

Improved Software Project Risk Assessment Using Bayesian Nets

Lukasz Radlinski

Submitted for the degree of Doctor of Philosophy

Queen Mary, University of London

2008

I certify that this thesis, and the research to which it refers, are the product of my own work, and that any ideas or quotations from the work of other people, published or otherwise, are fully acknowledged in accordance with the standard referencing practices of the discipline. I acknowledge the helpful guidance and support of my supervisor, Professor Norman Fenton. I also acknowledge the following individuals for their support: Professor Martin Neil, Dr. David Marquez, Dr. William Marsh, Peter Hearty (all with Queen Mary, University of London) and Dr. Barbara Zaborek (Pomeranian Medical University, Szczecin, Poland).

Lukasz Radlinski

20/09/2008

Date

Abstract

Empirical software engineering models typically focus on predicting development effort or software quality but not both. Using Bayesian Nets (BNs) as causal models, researchers have recently attempted to build models that incorporate relationships between functionality, effort, software quality, and various process variables. This thesis analyses such models and, as part of a new validation study, identifies their strengths and weaknesses. A major weakness is their inability to incorporate prior local productivity and quality data, which limits their applicability in real software projects. The main hypothesis is that it is possible to build BN models that overcome these limitations without compromising their basic philosophy. In particular, the thesis shows we can build BNs that capture known trade-offs and can be tailored to individual company needs.

The new model, called the Productivity Model, is developed by using the results of the new validation of the existing model, together with various other analyses. These include: the results of applying various statistical methods to identify relationships between a range of variables using publicly available data on software projects; analyses of other studies; expert knowledge. The new model is also calibrated using the results of an extensive questionnaire survey of experts in the area.

The thesis also makes a number of other novel contributions to improved risk assessment using BNs, including.

- A model which predicts the proportions of different types of defects likely to be left in software after testing. The model uses the results of statistical analysis on the past software projects. It can be combined with other defect prediction models to predict the number of residual defects of different categories.
- A learning model for predicting the number of defects found and fixed in successive testing iterations.

To Yvonne

Glossary of abbreviations

| | |
|----------|---|
| BN | Bayesian Network |
| CPD | Conditional Probability Distribution |
| CPT | Conditional Probability Table |
| DAG | Directed Acyclic Graph |
| DBN | Dynamic Bayesian Network |
| DD | Dynamic Discretisation |
| DTM | Defect Types Model |
| ELMIT | Enhanced Learning Model for Iterative Testing |
| FP | Function Point |
| GUI | Graphical User Interface |
| HMM | Hidden Markov Model |
| ID | Influence Diagram |
| KF | Kalman Filter |
| KLOC | Thousand (Kilo) Lines of Code |
| KW-ANOVA | Kruskal-Wallis one-way ANalysis Of VAriance |
| LMIT | Learning Model for Iterative Testing |
| LMITFD | Learning Model for Iterative Testing and Fixing Defects |
| LOC | Lines of Code |
| NBC | Naïve Bayesian Classifier |
| NPT | Node Probability Table |
| PDF | Probability Density Function |
| SPI | Software Process Improvement |
| SRCC | Spearman's Rank Correlation Coefficient |

Table of contents

| | |
|--|-----|
| Abstract | 3 |
| Glossary of abbreviations..... | 5 |
| Table of contents | 6 |
| List of figures | 9 |
| List of tables..... | 12 |
| 1 Introduction | 14 |
| 2 Background and Motivation..... | 20 |
| 2.1 Classic software engineering models | 20 |
| 2.2 Overview of project trade-offs | 21 |
| 2.3 Trade-offs in parametric models for software engineering..... | 22 |
| 2.4 Problems with classic approaches to predicting software project risk..... | 23 |
| 2.5 Methods addressing limitations of traditional software metrics approaches .. | 24 |
| 2.6 Summary | 32 |
| 3 Review of most recent empirical data | 33 |
| 3.1 Various data | 33 |
| 3.2 Trade-offs in software development | 35 |
| 3.3 Project management | 37 |
| 3.4 Defect prediction | 39 |
| 3.5 Requirements management | 43 |
| 3.6 Summary | 43 |
| 4 Review of Bayesian Networks for Software Project Risk Assessment | 45 |
| 4.1 Introduction to Bayesian nets..... | 45 |
| 4.2 Trade-off analysis in a simple BN | 48 |
| 4.3 Causal frameworks for risk | 49 |
| 4.4 Naïve Bayesian Classifier | 52 |
| 4.5 Ranked nodes | 53 |
| 4.6 Indicator and causal approach..... | 56 |
| 4.7 Reasoning over time..... | 60 |
| 4.8 MODIST Project-Level BN | 61 |
| 4.9 MODIST Phase-based Defect Prediction Model | 65 |
| 4.10 Revised Defect Prediction Model | 71 |
| 4.11 Comparison of BN models..... | 80 |
| 4.12 Summary | 82 |
| 5 Limitations of existing BN models | 83 |
| 5.1 Integration of models | 83 |
| 5.2 Incorporating new empirical data..... | 84 |
| 5.3 Using constants | 86 |
| 5.4 Dynamic discretisation..... | 91 |
| 5.5 Custom units of measurement..... | 99 |
| 5.6 Defect prediction | 100 |
| 5.7 Summary | 101 |
| 6 The Productivity Model | 102 |
| 6.1 Overview of the Productivity Model..... | 102 |
| 6.2 Model features..... | 103 |
| 6.3 Summary of model variables | 104 |
| 6.4 Model structure | 106 |

| | | |
|------|---|-----|
| 6.5 | Prior probabilities in ranked variables | 112 |
| 6.6 | Model calibration | 112 |
| 6.7 | Issues arising from model development..... | 116 |
| 6.8 | Future Enhancements to the Productivity Model..... | 119 |
| 6.9 | Summary | 122 |
| 7 | Validating Productivity Model..... | 123 |
| 7.1 | Constraints to external validation..... | 123 |
| 7.2 | Non-default productivity and defect rates..... | 124 |
| 7.3 | Meeting the target for product quality | 126 |
| 7.4 | Impact of uncontrollable project factors | 128 |
| 7.5 | Changed effort allocation..... | 130 |
| 7.6 | Explaining counterintuitive outcome | 132 |
| 7.7 | Influence of uncertainty in qualitative factors | 134 |
| 7.8 | Influence of controllable and uncontrollable factors | 134 |
| 7.9 | Using custom units of measurement | 137 |
| 7.10 | Setting interval target for numeric node..... | 139 |
| 7.11 | Sensitivity analysis..... | 141 |
| 7.12 | Summary | 144 |
| 8 | Modelling prior defect and productivity rates..... | 146 |
| 8.1 | The dataset | 146 |
| 8.2 | Statistical analysis of the dataset..... | 147 |
| 8.3 | Bayesian net for prior productivity and defect rates | 150 |
| 8.4 | Model validation | 153 |
| 8.5 | Summary | 163 |
| 9 | Model for predicting types of defects | 164 |
| 9.1 | The dataset | 164 |
| 9.2 | Statistical analysis of dataset..... | 164 |
| 9.3 | Bayesian net for estimating types of defects..... | 168 |
| 9.4 | Model validation | 172 |
| 9.5 | Summary | 182 |
| 10 | Learning Model for Iterative Testing and Fixing Defects | 183 |
| 10.1 | Background | 183 |
| 10.2 | The datasets | 184 |
| 10.3 | Basic learning model for iterative testing | 185 |
| 10.4 | Lessons learned from basic model creation and validation | 187 |
| 10.5 | Extended learning model for iterative testing | 190 |
| 10.6 | Model for iterative testing and fixing defects | 193 |
| 10.7 | Validating learning model for iterative testing and fixing defects..... | 196 |
| 10.8 | Summary | 204 |
| 11 | Summary and future work..... | 205 |
| 11.1 | Novel contributions..... | 205 |
| 11.2 | Future work | 208 |
| 11.3 | Final conclusions..... | 209 |
| | References | 210 |
| | Appendix A Summary of empirical data used indirectly | 226 |
| | A.1 Project management | 226 |
| | A.2 Software process improvement | 226 |
| | A.3 Software sizing..... | 232 |
| | A.4 Defect prediction | 233 |
| | Appendix B The Productivity Model | 235 |

| | | |
|------------|---|-----|
| B.1 | Definition of the Productivity Model..... | 235 |
| B.2 | Questionnaire about relationships between software project factors..... | 266 |
| B.3 | Raw results from questionnaire survey..... | 283 |
| Appendix C | Productivity and Defect Rates Model..... | 285 |
| C.1 | Statistical analysis of productivity and defect rates..... | 285 |
| C.2 | Definition of PDR model..... | 290 |
| Appendix D | Defect Types Model..... | 300 |
| D.1 | Statistical analysis of defect types..... | 300 |
| D.2 | Definition of defect types model..... | 305 |
| Appendix E | Learning Model for Testing and Fixing Defects..... | 310 |
| E.1 | Definition of Learning Model for Testing and Fixing Defects..... | 310 |
| E.2 | Defect Removal Model Manager..... | 322 |
| E.3 | Datasets used to validate the model..... | 326 |

List of figures

| | | |
|-------------|---|----|
| Figure 3-1 | Coding defect introduction ranges..... | 42 |
| Figure 3-2 | Number of defects found and fixed in each iteration in various NASA datasets..... | 42 |
| Figure 4-1 | Example of simple BN model | 45 |
| Figure 4-2 | Parametric-type model converted to a BN | 48 |
| Figure 4-3 | Predictions in the parametric-type model converted to a BN..... | 49 |
| Figure 4-5 | Causal taxonomy of risk..... | 51 |
| Figure 4-6 | Typical structure of NBC | 53 |
| Figure 4-8 | Predicted <i>process effectiveness</i> in an example model | 55 |
| Figure 4-9 | Example of using indicator approach in MODIST Project-level Model..... | 57 |
| Figure 4-10 | Influence of one indicator node on another one in MODIST Project-level Model..... | 57 |
| Figure 4-11 | Causal approach example in Revised Defect Prediction Model | 58 |
| Figure 4-12 | Example of indicator approach MODIST Phase-based Defect Prediction Model | 58 |
| Figure 4-13 | Common causes in Revised Defect Prediction Model | 60 |
| Figure 4-14 | Structure of a simple temporal model | 60 |
| Figure 4-15 | Schematic of the Project-level Model | 62 |
| Figure 4-16 | Structure of the key part of the MODIST Project-level Model..... | 63 |
| Figure 4-17 | Predicted distributions after setting ‘perfect’ quality for software..... | 64 |
| Figure 4-18 | Predicted distributions after setting fixed <i>project duration</i> | 64 |
| Figure 4-19 | Predicted distributions after resigning from ‘perfect’ software quality .. | 65 |
| Figure 4-20 | Schematic view of the whole phase net in Phase-based Defect Prediction Model | 67 |
| Figure 4-21 | Predictions for different <i>testing effort</i> scenarios | 68 |
| Figure 4-22 | Impact of very good development on defect prediction..... | 68 |
| Figure 4-23 | Example of sequence of joined models | 70 |
| Figure 4-24 | Risk framework in the MODIST Phase-based Defect Prediction Model..... | 71 |
| Figure 4-25 | Schematic view of Revised Defect Prediction Model..... | 72 |
| Figure 4-26 | Specification and documentation subnet in Revised Defect Prediction Model..... | 73 |
| Figure 4-27 | New functionality subnet in Revised Defect Prediction Model | 73 |
| Figure 4-28 | Existing code base subnet in Revised Defect Prediction Model | 74 |
| Figure 4-29 | Common influences subnet in Revised Defect Prediction Model..... | 74 |
| Figure 4-30 | Influence of quality of existing code on defect prediction in the current phase..... | 75 |
| Figure 4-31 | Predicted distributions after adding poor subcontractor and necessary integration with third party software | 75 |
| Figure 4-32 | Impact of particular qualitative factors on <i>residual defects post</i> | 77 |
| Figure 4-33 | Impact of the two most important qualitative factors on <i>residual defects post</i> | 79 |
| Figure 5-1 | Simple model parameterization with additional node | 87 |
| Figure 5-2 | Defining constants for <i>KLOC delivered</i> | 87 |
| Figure 5-3 | Predicted <i>residual defects post</i> with static discretisation | 92 |

| | | |
|-------------|---|-----|
| Figure 5-4 | Predicted probability distributions for <i>residual defects post</i> for original and revised models in selected scenario | 97 |
| Figure 5-5 | Scale of new functionality subnet in Phase-based Defect Prediction Model..... | 99 |
| Figure 6-1 | Main subnets of the Productivity Model | 102 |
| Figure 6-2 | Structure of the Productivity Model..... | 108 |
| Figure 6-3 | Structure of subnet for process and people quality | 110 |
| Figure 6-4 | Propagating observation for process factor the same in all development activities | 111 |
| Figure 6-5 | Impact of controllable and uncontrollable factors on productivity and software quality | 115 |
| Figure 7-1 | Predictions with custom prior rates | 125 |
| Figure 7-2 | Meeting the target for product quality..... | 127 |
| Figure 7-3 | Predictions with changed uncontrollable factors..... | 128 |
| Figure 7-4 | Entering soft evidence to <i>quality of input documentation</i> in AgenaRisk | 129 |
| Figure 7-5 | Predictions with default and changed effort allocation | 131 |
| Figure 7-6 | Predictions for variables influencing trade-off relationships | 133 |
| Figure 7-7 | Predictions for default and most probable scenarios..... | 134 |
| Figure 7-8 | Predictions for most and least favourable scenarios of controllable and uncontrollable factors | 136 |
| Figure 7-9 | Predictions for different combinations of controllable and uncontrollable factors | 137 |
| Figure 7-10 | Predictions with custom units of measurement..... | 138 |
| Figure 7-11 | Predictions for scenario with interval target for <i>revised defect rate</i> (approach 1)..... | 140 |
| Figure 7-12 | Predictions for scenario with interval target for <i>revised defect rate</i> (approach 2)..... | 141 |
| Figure 7-13 | Impact of factors on process effectiveness in different development activities..... | 143 |
| Figure 7-14 | Impact of uncontrollable project factors on productivity and defect rates..... | 143 |
| Figure 7-15 | Impact of process factors on productivity and defect rates | 144 |
| Figure 7-16 | Impact of individual process and people factors on overall <i>process and people quality</i> | 144 |
| Figure 7-17 | Impact of aggregated controllable and uncontrollable factors on productivity and defect rates | 144 |
| Figure 8-1 | Structure of a BN for predicting productivity and defect rates | 152 |
| Figure 8-2 | Predictions when no observations are entered..... | 154 |
| Figure 8-3 | Predicted <i>defect rate</i> for different <i>functional sizes</i> | 155 |
| Figure 8-4 | Predicted median values for <i>defect rate</i> depending on various <i>functional sizes</i> | 155 |
| Figure 8-5 | Predicted <i>defect rate</i> depending on CASE tool usage..... | 156 |
| Figure 8-6 | Predicted productivity and defect rates depending on <i>development platform</i> | 157 |
| Figure 8-7 | Predicted <i>productivity rate</i> depending on <i>development type</i> | 157 |
| Figure 8-8 | Predicted <i>productivity rate</i> depending on <i>language type</i> | 158 |
| Figure 8-9 | Predicted productivity and defect rates depending on <i>used methodology</i> | 158 |
| Figure 8-10 | Predicted <i>productivity rate</i> depending on number of user locations..... | 159 |

| | | |
|--------------|--|-----|
| Figure 8-11 | Predicted productivity and defect rates in most and least favourable scenarios | 160 |
| Figure 8-12 | Predictions after linking PDR Model with Productivity Model..... | 161 |
| Figure 8-13 | Tornado graphs illustrating sensitivity of dependent variables..... | 162 |
| Figure 9-2 | Prior proportions of each type of defects | 170 |
| Figure 9-3 | Linking a BN for estimating types of defects with another defect prediction model..... | 171 |
| Figure 9-4 | Predictions for different <i>functional sizes</i> | 172 |
| Figure 9-5 | Impact of <i>task complexity</i> on predicted proportions of defects..... | 172 |
| Figure 9-6 | Impact of <i>deadline pressure</i> on predicted proportions of defects | 173 |
| Figure 9-7 | Impact of <i>package customisation</i> on predicted proportions of defects . | 173 |
| Figure 9-8 | Impact of <i>GUI usage</i> on predicted proportions of defects | 174 |
| Figure 9-9 | Predictions for different combinations of uncontrollable factors..... | 175 |
| Figure 9-10 | Predictions for different states of controllable factors | 176 |
| Figure 9-11 | Predictions for different combinations of controllable factors..... | 177 |
| Figure 9-12 | Predictions for most and least favourable scenarios..... | 178 |
| Figure 9-13 | Predictions after entering soft evidence | 179 |
| Figure 9-14 | Predicted <i>number of defects</i> after linking with Productivity Model..... | 180 |
| Figure 9-15 | Tornado graphs illustrating sensitivity of dependant variables..... | 181 |
| Figure 10-1 | Different shapes of trend lines for number of defects found depending on time; adopted from..... | 185 |
| Figure 10-2 | General structure of the Learning Model for Iterative Testing | 186 |
| Figure 10-3 | Detailed structure of LMIT | 186 |
| Figure 10-4 | Initial prediction results from LMIT with PC1 dataset | 187 |
| Figure 10-5 | Detailed structure of the Extended Learning Model for Iterative Testing (single iteration)..... | 191 |
| Figure 10-6 | Prediction results from ELMIT with semi-randomly generated dataset | 192 |
| Figure 10-7 | Structure of the Learning Model for Iterative Testing and Fixing Defects (single iteration) | 193 |
| Figure 10-8 | Detailed structure of testing and fixing process in Learning Model for Iterative testing and Fixing Defects (single iteration) | 194 |
| Figure 10-9 | Prediction results for the LMITFD with semi-randomly generated dataset | 197 |
| Figure 10-10 | Predicted total number of <i>defects found</i> and <i>defects fixed</i> depending on different number of iterations used to learn the model..... | 198 |
| Figure 10-11 | Relative errors in predictions for total number of <i>defects found</i> and <i>defects fixed</i> depending on number of iterations used to learn the model | 199 |
| Figure 10-12 | Impact of parameter <i>maximum number of iterations</i> for DD algorithm..... | 200 |
| Figure 10-13 | Predictions from LMITFD with testing and fixing <i>effort</i> halved and doubled after 5 iterations used to learn the model..... | 201 |
| Figure 10-14 | Predictions from LMITFD with very high <i>process and people quality</i> and medium <i>other factors</i> starting from iteration 6..... | 202 |
| Figure 10-15 | Predictions from LMITFD with very high and very low <i>fixing process and people quality</i> after 5 iterations used to learn the model... | 203 |

List of tables

| | | |
|------------|--|----|
| Table 1-1 | Research hypotheses verified in this thesis | 15 |
| Table 1-2 | List of papers with a reference to the this PhD thesis chapter..... | 17 |
| Table 2-1 | Examples of parametric models for predicting development effort or software quality | 21 |
| Table 2-2 | Summary of applications of modern methods in software engineering .. | 26 |
| Table 2-3 | Comparison of features of modern methods used in software project risk assessment | 32 |
| Table 3-1 | Summary of advantages and disadvantages of most popular publicly available software engineering repositories | 34 |
| Table 3-2 | Major results from Putnam's research | 35 |
| Table 3-3 | Cost/schedule/defect trade-off report | 36 |
| Table 3-4 | Selected descriptive statistics for projects analyzed in..... | 36 |
| Table 3-5 | Simple correlations with model performance..... | 37 |
| Table 3-6 | Most common risk factors in depending on application sector..... | 37 |
| Table 3-7 | Main results of research to find optimum team size; based on data from | 38 |
| Table 3-8 | Performance of the best and worst performing software organizations in Europe; adopted from..... | 38 |
| Table 3-9 | Productivity for object-oriented projects..... | 38 |
| Table 3-10 | COCOMO II multipliers..... | 39 |
| Table 3-11 | Estimated defect rates per function point depending on CMM level..... | 40 |
| Table 3-12 | Cumulative percentages of defects removed by development phase for organizations at CMM Level 4 | 40 |
| Table 3-13 | Software size versus defect potential, defect removal efficiency, and defect density..... | 40 |
| Table 3-14 | Comparison of defect potentials and removal effectiveness | 41 |
| Table 3-15 | Estimated defect removal effectiveness for organizations at different CMM levels | 41 |
| Table 4-1 | Advantages and disadvantages of BNs..... | 48 |
| Table 4-2 | Summary of pros and cons of implementing different approaches | 59 |
| Table 4-3 | Results of global sensitivity analysis..... | 78 |
| Table 4-4 | Values of model accuracy evaluation measures | 80 |
| Table 4-5 | Comparison of pros and cons of analyzed BN models..... | 81 |
| Table 5-1 | Difficult expressions in existing models | 84 |
| Table 5-2 | Original and adjusted defect rates for different states of <i>quality delivered</i> | 85 |
| Table 5-3 | Types of adjustments of existing models | 86 |
| Table 5-4 | Current and adjusted partitioned expressions for <i>KLOC delivered</i> | 87 |
| Table 5-6 | Current and adjusted partitioned expressions <i>total effective effort</i> | 89 |
| Table 5-7 | Current and adjusted expressions for <i>project duration and average number of people full time</i> | 89 |
| Table 5-8 | Implementing constants in other existing models | 90 |
| Table 5-9 | Intervals for nodes <i>new functionality</i> and <i>KLOC (new)</i> | 93 |
| Table 5-10 | Node states for <i>defects found</i> in MODIST Phase-based Defect Prediction Model | 94 |
| Table 5-11 | Numeric node types in original and revised models | 96 |

| | | |
|------------|---|-----|
| Table 5-12 | Comparison of calculation times for selected scenarios in original and revised model..... | 98 |
| Table 6-1 | Summary of main variables in the Productivity Model..... | 104 |
| Table 6-2 | Summary of the questionnaire survey results..... | 114 |
| Table 6-3 | Weights for controllable and uncontrollable factors | 115 |
| Table 6-4 | Example of effort allocation..... | 117 |
| Table 6-5 | Summary of code reuse statistics obtained from questionnaire survey. | 121 |
| Table 7-1 | Combinations of factors in most and least favourable scenarios for productivity and defect rate | 135 |
| Table 8-1 | Summary of descriptive statistics for dependant variables | 147 |
| Table 8-2 | Descriptions of potential predictors used to build the model..... | 148 |
| Table 8-3 | Predictors kept after each step of statistical analysis..... | 150 |
| Table 8-4 | Summary of predictors in PDR Model..... | 151 |
| Table 8-5 | Transforming <i>functional size</i> to meaningful intervals..... | 151 |
| Table 8-6 | Impact of predictors on dependent variables reflected in the model..... | 153 |
| Table 8-7 | Most and least favourable scenarios..... | 159 |
| Table 9-1 | Summary of descriptive statistics for dependant variables | 165 |
| Table 9-2 | Summary of potential predictor variables used to build the model..... | 165 |
| Table 9-3 | Predictors kept after each step of statistical analysis..... | 168 |
| Table 9-4 | Summary of predictors in Defect Types Model | 169 |
| Table 9-5 | Impact of predictors on dependent variables reflected in the model..... | 171 |
| Table 9-6 | Summary of analyzed sample projects | 174 |
| Table 9-7 | Summary of two analyzed sample projects | 176 |
| Table 9-8 | Most and least favourable scenarios..... | 178 |
| Table 10-1 | Expressions in the LMIT | 187 |
| Table 10-2 | Expressions in the Extended Model for Iterative Testing | 191 |
| Table 10-3 | Comparison of calculation times for LMITFD depending on different <i>maximum number of iterations</i> | 200 |

1 Introduction

As the demands for developing good software cheaply and quickly continue to grow, so does the challenge of assessing risk in software projects. Much effort has been spent on developing predictive models providing useful information to support the decision-making process. These models typically aim to estimate required resources for software projects or to estimate the quality of delivered software. Yet, few models have addressed the ultimate objective of software metrics, which is to provide software managers support for decision-making and risk assessment based on quantification. Such an objective requires a combination of both the resource and quality perspectives of a project, and would allow software managers to answer the following types of questions:

- If we have insufficient effort for this project how much functionality must we lose to deliver the best quality?
- What are the relationships between functionality, effort and number of defects?
- How does the change in the process and people quality affect the functionality and the quality of the software delivered?
- What functionality and quality should we expect if our project is more complex than the previous ones?
- How does a change in inherent project factors impact on estimated effort and software quality?
- What will be the impact on functionality and quality if the allocated effort is changed?
- How many defects will be found and fixed during the testing process?
- How severe will the defects be for future software users?

The main objective of this thesis is to develop models to answer such questions.

One approach that has shown considerable promise in meeting such aims is Bayesian nets (BNs). From one perspective a BN is a graphical model, which makes it easy to understand, interpret and use. From another perspective a BN is a set of model variables defined as probability distributions; this enables us to reason under uncertainty, a feature that becomes especially relevant in software projects where missing and uncertain data is very common. Although BNs have been successfully used in software projects they have some limitations preventing them from providing

relevant estimates in many cases. This thesis proposes new BNs which overcome their limitations. Specifically, this thesis addresses the four research hypotheses listed in Table 1-1.

Table 1-1 Research hypotheses verified in this thesis

| No. | Hypothesis | Chapter of the thesis |
|--|--|-------------------------|
| H1 | It is possible to build a BN for software project risk assessment which overcomes key limitations of existing BNs without compromising their basic philosophy. In particular, such BNs enable trade-off analysis between key project factors (effort, functionality and product quality) and can be tailored to individual company needs. | 2, 3, 4, 5, 6, 7 |
| H2 | It is possible to build a BN for estimating team productivity rate and software defect rate which incorporates environmental factors describing the nature of the developed software, and includes factors describing the development process identified through the statistical analysis of empirical data and experts' knowledge. | 3, 5, 8 |
| H3 | It is possible to build a BN for proportions of different types of defects categorized by their severity which incorporates environmental factors describing the nature of the developed software, and includes factors describing the development process identified through the statistical analysis of empirical data and experts' knowledge. | 3, 5, 9 |
| H4 | It is possible to build a BN for predicting the number of defects likely to be found and fixed in future testing and fixing iterations which learns the values of its parameters using observations from past testing and fixing iterations. | 3, 5, 10 |
| The main chapters in which each hypothesis is verified are indicated in bold | | |

This thesis is structured as follows:

Chapter 2 contains a review of existing relevant models and demonstrates their advantages and limitations.

Chapter 3 describes recent important empirical data which were used directly both to inform the existing BN models and also the new models proposed.

Chapter 4 introduces the theory behind BNs and includes new material that highlights the advantages and limitations of existing models.

Chapter 5 formulates the research challenges which arise from the limitations of existing BN models. It also focuses on implementing the relatively simple solutions to some of the problems with existing BNs. Implementing the most important research challenges requires developing new models discussed in subsequent chapters.

Chapter 6 is devoted to the main new contribution of this thesis – a newly developed Productivity Model which can be used for risk assessment in software projects. This model addresses key limitations of existing models. This chapter discusses the structure of this model, the steps performed to calibrate it using

questionnaire survey data, the steps required to prepare it for use and the problems that arose during model creation and validation.

Chapter 7 discusses how the Productivity Model captures known relationships between its most important variables.

Chapter 8 discusses development of a new model for estimating prior productivity and defect rates (PDR Model) based on a set of uncontrollable environmental factors.

Chapter 9 introduces a new model for predicting proportions of different types of defects depending on their severity (DTM Model).

Chapter 10 is devoted to developing a new Learning Model for Iterative Testing and Fixing Defects (LMITFD).

Chapter 11 summarizes the main results achieved during this study together with discussion on future work extending the results achieved so far.

In addition there are the following appendices:

- Appendix A – a summary of recent empirical data used indirectly to inform the developed models and which can be used in future work extending the newly developed models;
- Appendix B – additional information on the Productivity Model
 - B.1: definition of Productivity Model's structure in XML format;
 - B.2: questionnaire used to calibrate the Productivity Model;
 - B.3: raw results from the questionnaire survey performed to calibrate the Productivity Model;
- Appendix C – additional information on the PDR Model
 - C.1: results of the statistical analysis;
 - C.2: definition of PDR Model's structure in XML format;
- Appendix D – additional information on the DTM
 - D.1: results of the statistical analysis;
 - D.2: definition of DTM's structure in XML format;
- Appendix E – additional information on LMITFD
 - E.1 – definition of LMITFD's structure in XML format;
 - E.2 – definition of Defect Removal Model Manager – a software tool I developed to generate dynamic models, automate entering observations, running the model and exporting the results;

- E.3 – datasets used in validating dynamic models discussed in Chapter 10;

Much of the work in this thesis is based on papers that have been (or are soon to be) published. Table 1-2 contains the list of such papers together (with a statement of my individual contribution where there are co-authors) and references to the chapters of this thesis where they are exploited.

Table 1-2 List of papers with a reference to the this PhD thesis chapter

| No. | Paper | Chapter of the thesis |
|-----|---|-----------------------|
| 1 | Fenton N., Hearty P., Neil M., Radliński Ł., Software Project and Quality Modelling Using Bayesian Networks, manuscript submitted to: Meziane F., Vadera S. (eds.), <i>Artificial Intelligence Applications for Improved Software Engineering Development: New Prospects</i> , 2008. I summarized some of the new BN models and discussed scenarios demonstrating their potential in providing information to decision-makers. | 6, 7, 8, 10 |
| 2 | Radliński Ł., Fenton N., Marquez D., Estimating Productivity and Defect Rates Based on Environmental Factors, <i>Information Systems Architecture and Technology: Models of the Organisation's Risk Management</i> , Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław, Poland, 2008, pp. 103–113. I performed the statistical analysis aiming to identify the most influential factors on productivity and defect rates. I built the BN model, called the PDR model, estimating productivity and defect rates which act as inputs to the Productivity Model built earlier. In the PDR model I used the results of this analysis as adjusted by other reported results and expert knowledge. I performed the model validation, sensitivity analysis and discussed the results. | 8 |
| 3 | Fenton N., Neil M., Marsh W., Hearty P., Radliński Ł., Krause P., On the effectiveness of early life cycle defect prediction with Bayesian Nets, <i>Empirical Software Engineering</i> , vol. 13 no. 5, pp. 499–537, Oct. 2008. This is the extension of paper 10. I described details of the structure of the BN model used in the analysis. I extended the validation by using other accuracy measures and I extended the discussion about the results. I performed the sensitivity analysis for the analyzed model. | 4 |
| 4 | Radliński Ł., Fenton N., Neil M., A Learning Bayesian Net for Predicting Number of Software Defects Found in a Sequence of Testing, <i>Polish Journal of Environmental Studies</i> , vol. 17 no. 3B, pp. 359–364, 2008. I developed a learning BN for predicting the number of defects found in specific future testing iterations. The model learns what values should be predicted by using the number of defects found in the past testing iterations. | 10 |
| 5 | Radliński Ł., A Review of Publicly Available Databases of Software Projects, accepted to: <i>Studia Informatica</i> , vol. 22, (in Polish – original title: Przegląd publicznie dostępnych baz danych przedsięwzięć informatycznych). | 3, 8, 9, 10 |
| 6 | Radliński Ł., A Survey of Bayesian Networks for Risk Assessment in Software Engineering, accepted to: <i>Studia Informatica</i> , vol. 21, (in Polish – original title: Przegląd sieci Bayesa do szacowania ryzyka w inżynierii oprogramowania). | 2, 4 |

| No. | Paper | Chapter of the thesis |
|-----|---|-----------------------|
| 7 | <p>Radliński Ł., Fenton N., Neil M., Marquez D., Improved Decision-Making for Software Managers Using Bayesian Networks, <i>Proc. 11th IASTED Int. Conf. Software Engineering and Applications</i>, Cambridge, MA, 2007, pp. 13–19.</p> <p>I developed a BN model for analysis of trade-offs in a software project. I demonstrated how the model can be used to support decision-making by the managers in some typical scenarios.</p> | 1, 6, 7 |
| 8 | <p>Radliński Ł., Fenton N., Marquez D., Hearty P., Empirical Analysis of Software Defect Types, <i>Information Systems Architecture and Technology. Information Technology and Web Engineering: Models, Concepts & Challenges</i>, Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław, Poland 2007, pp. 223–231.</p> <p>I performed the statistical analysis of the empirical data which led to identification of the most important factors influencing the proportions of different types of software defects. I also proposed a conceptual version of the BN model incorporating the results of this statistical analysis.</p> <p>The paper won the ‘Best Paper Award’ at the Information Systems Architecture and Technology conference (in Szklarska Poręba, Poland, 2007) based on the reviews and the presentation.</p> | 9 |
| 9 | <p>Radliński Ł., Fenton N., Neil M., Marquez D., Modelling Prior Productivity and Defect Rates in a Causal Model for Software Project Risk Assessment, <i>Polish Journal of Environmental Studies</i>, vol. 16 no. 4A, pp. 256–260, 2007.</p> <p>I performed the statistical analysis using the empirical data which led to developing a BN for estimating productivity and defect rates depending on a set of nominal and ranked variables.</p> | 8 |
| 10 | <p>Fenton N., Neil M., Marsh W., Hearty P., Radliński Ł., Krause P., Project Data Incorporating Qualitative Factors for Improved Software Defect Prediction, <i>Proc. 3rd Int. Workshop on Predictor Models in Software Engineering. Int. Conf. on Software Engineering</i> (May 20–26, 2007), IEEE Computer Society, Washington, DC, 2.</p> <p>I made enhancements to sections 2, 6 and 7 of the paper.</p> | 4 |
| 11 | <p>Radliński Ł., Comparative Analysis of Software Quality Models, <i>Studia Informatica</i>, vol. 19, 2006, pp. 131–150, (in Polish – original title: Analiza porównawcza modeli jakości oprogramowania).</p> | 11 |
| 12 | <p>Radliński Ł., Modelling Complex Nodes in Bayesian Nets for Software Project Risk Assessment, <i>Polish Journal of Environmental Studies</i>, vol. 15 no. 4C, pp. 149–152, 2006.</p> | 4, 5 |
| 13 | <p>Fenton N., Radliński Ł., Neil M., Improved Bayesian Networks for Software Project Risk Assessment Using Dynamic Discretisation, in: Sacha K. (ed.) <i>Software Engineering Techniques: Design for Quality</i>, IFIP International Federation for Information Processing, vol. 227, Springer, Boston 2006, pp. 139–148.</p> <p>I converted the numeric nodes in the original model to the simulation nodes. I ran a series of tests to identify the differences in predicted results and calculation times between the original and revised models.</p> | 5 |
| 14 | <p>Radliński Ł., Software reliability estimation and prediction, in: <i>Advanced Information Technologies for Management. Research Papers</i>, no. 1044, Publishing House of the University of Economics, Wrocław, Poland, 2004, pp. 82–90.</p> | 2 |
| 15 | <p>Radliński Ł., Software Process Improvement in Small Software Companies, in: Kisielnicki J. (ed.), <i>Informatics as Tool in Modern Management</i>, Wydawnictwo Polsko-Japońskiej Wyższej Szkoły Technik Komputerowych, Warsaw, Poland, 2004, pp. 437–446, (in Polish – original title: Poprawa procesów programowych w małych organizacjach programistycznych; book title: Informatyka narzędziem współczesnego zarządzania).</p> | Appendix A |

| No. | Paper | Chapter of the thesis |
|-----|--|-----------------------|
| 16 | Radliński Ł., Selected Problems of User Requirements Specification Quality Assessment, <i>Advanced Information Technologies for Management. Research Papers</i> , no. 1044, Publishing House of the University of Economics, Wrocław, Poland, 2003, pp. 69–77, (in Polish – original title: Wybrane problemy zapewnienia jakości specyfikacji wymagań użytkowników). | 3, 6 |
| 17 | Radliński Ł., Software Quality Measurement – Opportunities and Limitations, <i>Firma i Rynek</i> , no. 2–4 (27–29), pp. 136–139, 2003, (in Polish – original title: Pomiar jakości oprogramowania – możliwości i ograniczenia). | 11 |

2 Background and Motivation

This chapter discusses various approaches to developing predictive software engineering models, most often used to predict development effort and software quality. These approaches include the most popular and widely known parametric-type models, as well as those based on system dynamics, machine learning or Bayesian methods. Comparison of different modelling methods suggests that Bayesian nets is the best modelling approach to develop models for software project risk assessment. The new contribution of this chapter is the analysis of applicability of various modelling approaches to software engineering. Section 2.5 is partially based on [201].

2.1 *Classic software engineering models*

Most current software engineering models predict development effort or software quality. Development effort models [9, 12, 18, 21, 22, 23, 39, 43, 55, 56, 67, 93, 149, 161, 162, 173, 190, 191, 197, 212, 215, 219, 227, 230, 233, 235, 237, 246, 249, 254] typically estimate the amount of resources (duration and people) required to develop a project or part of a project. Some models also enable estimation of project duration [21, 22, 23, 58, 67, 212]. A comparison of these effort models is provided in [255].

Predicting software quality is more complicated. Since software quality is an aggregation of several attributes, several models for assessing overall software quality have been developed [24, 25, 59, 78, 86, 91, 117, 159, 247]. These models are compared in [202]. Current efforts for estimating and predicting software quality mainly refer to one of its attributes at a time:

- total number of defects or defect rate [6, 13, 15, 42, 48, 75, 77, 67, 68, 69, 79, 83, 90, 97, 101, 108, 150, 183, 254],
- software reliability (time between failures, number of failures in a given testing interval) [7, 11, 52, 58, 88, 123, 147, 152, 175, 176, 198, 216, 231, 256, 257, 262],
- identification of fault-prone software components (modules, classes etc.) [29, 34, 62, 95, 102, 103, 104, 112, 135, 163, 171, 185, 226],
- user satisfaction [41, 44, 67, 251],

A comparison of quality prediction models is provided in [73, 211].

Table 2-1 summarizes the examples of parametric predictive models for software engineering.

Table 2-1 Examples of parametric models for predicting development effort or software quality

| Model/Author | Year | Dependent variable |
|------------------------------|------|--|
| Akiyama [6] | 1971 | number of defects |
| Jelinski-Moranda [123] | 1972 | time between failures |
| Ferdinand [79] | 1974 | number of defects |
| Halstead [101] | 1977 | number of defects |
| Goel and Okumoto [87] | 1978 | time between failures |
| SLIM [197] | 1978 | effort |
| Goel and Okumoto [88] | 1979 | number of failures in given testing interval |
| Littlewood [151] | 1981 | time between failures |
| COCOMO [23] | 1981 | effort, duration |
| Lipow [150] | 1982 | defect rate |
| Goel [89] | 1982 | number of failures in given testing interval |
| Yamada, Ohba and Osaki [256] | 1983 | number of failures in given testing interval |
| Gaffney [83] | 1984 | number of defects |
| Musa and Okumoto [176] | 1984 | number of failures in given testing interval |
| Ohba [181] | 1984 | number of failures in given testing interval |
| UNISYS [48] | 1990 | number of defects |
| COCOMO II [21, 22] | 1995 | effort, duration |
| Ohlsson and Alberg [182] | 1996 | number of trouble reports |
| COQUALMO [42] | 1999 | defect rate |

2.2 Overview of project trade-offs

One of the most demanding tasks for project managers is to appropriately balance the key project constraints. Trade-off analysis enables us to estimate how much you need to sacrifice one or more project constraints if you wish to achieve better performance in one or more other constraints. For example, how much more *effort* is required to develop software of a given size with a fixed *process quality* and with higher target for *quality delivered*? Or: how much less *functionality* can be delivered with fixed *process quality*, fixed target *quality delivered* but less *effort* spent on the project.

Several researchers have analysed trade-offs between the key project variables. Swink et al. [239] have developed a theory of project efficiency and performance trade-offs based on empirical data. They found that trade-offs are more strongly reflected in efficient projects compared to inefficient projects. Several authors performed pairwise analyses of relationships between project variables :

- speed-quality [33, 105],
- time-cost [92, 143, 194],

- time-quality [131, 169].

Ruhe et al. [218] used AHP (Analytical Hierarchical Process) to analyze trade-offs between effort, time and quality in evaluating, prioritizing and selecting candidate requirements in software projects. Pollack-Johnson and Liberatore [193] analyzed trade-offs between project time and cost, together with product quality incorporated in their model. They also developed a model based on AHP enabling it to find the best combination of project duration and cost for a target product quality. Babu and Suresh [10] developed linear programming models used in analysis of trade-offs between time, cost and quality.

The list of factors which are traded for another in a project are not fixed. For example, Smith and Reinertsen [232] identify four such factors: project timeliness, product performance, development expense, and product cost. This thesis focuses on trade-off analysis between:

- *development effort* (in some places separating *project duration* and *number of people*),
- *process quality*,
- *functionality delivered*,
- *quality delivered*.

2.3 Trade-offs in parametric models for software engineering

There are two main requirements for a model to enable a trade-off analysis:

1. Incorporating trade-off factors in a single model – very few parametric models capture effort and quality together in a single model.
2. Supporting both forward and backward inference – parametric-type models do not support backward inference nor can they be converted in a statistically meaningful way to switch one of their predictors to a dependent variable.

For example, a model for predicting development *effort* is typically expressed as a function of *functionality* (or more generally *size*) and other quantitative and/or qualitative variables (X) as shown in Equation 1.

$$\text{effort} = f(\text{functionality}, X) \quad (1)$$

Let us try to convert such a model to estimate how much *functionality* it is possible to develop using limited *effort* and with adjustment of other quantitative and

qualitative variables. Using arithmetic operations we could probably produce a model such as the one presented in Equation 2.

$$\text{functionality} = f(\text{effort}, X) \quad (2)$$

From a mathematical point of view both models would contain the same information. But from a statistical point of view such a conversion is not acceptable. Traditional regression-based models assume a strong correlation between each predictor and each predictive variable. They also assume weak, ideally no, correlation between particular predictors. The original model was built with the assumption that *functionality* and other predictors (X) are not correlated. But in the second model *functionality* becomes a predictive variable, which by definition should be correlated with the set of predictor variables. Unfortunately, *functionality* is only correlated with *effort*. Therefore, the second model does not seem to be correct.

2.4 Problems with classic approaches to predicting software project risk

Parametric approaches generally suffer from the following disadvantages:

- They do not incorporate trade-offs between resources, cost, quality and functionality. Rather, one of them is used as a predictor variable to predict another one of them.
- Results are often expressed only as point values ignoring the inevitable uncertainty in such predictions.
- They cannot incorporate qualitative judgement.
- Being based on past data the models cannot handle process changes.
- The formulaic approach leads to over-simplistic relationships.
- They do not enable backward inference.
- They may be critically dependent on variables for which no data is available.
- Models often fail to take into account the type of software development life-cycle.

More on the limitations of classic parametric models can be found in [73, 76].

2.5 **Methods addressing limitations of traditional software metrics approaches**

To address the disadvantages of the parametric models, numerous other methods have been used. We consider these in three categories: system dynamics, ‘learning’ methods, and Bayesian methods:

System dynamics

System dynamics (SD) was introduced in the 1960s by Forrester [80]. It is a method for “understanding how complex systems change over time” [243]. The key concept is to use feedback loops to model the impact of variables from one time slice to the next. Because system dynamics models are built by domain experts they often reflect causal relationships between variables. An extensive literature survey on system dynamics and its applications in various disciplines is contained in [223, 245].

‘Learning’ methods

The numerous learning methods (both machine learning and otherwise) that have been applied in software engineering include:

- *Artificial neural networks (NN)* [19, 96, 107, 186] which aim to simulate the behaviour of biological neural networks to solve various artificial intelligence problems.
- *Fuzzy sets (FS)* [60, 160, 258]. They *extend* the classical concept of sets by degrees of membership.
- *Rough sets (RS)* introduced by Pawlak [187, 188] as an extension to classic set theory to be used with incomplete knowledge. They can be treated as sets with *fuzzy* boundaries. That means that these boundaries are not precisely described by a set of their attributes but are approximated.
- *Estimation by analogy (EA)* – also called *instance-based learning*: There are two main methods: k-nearest neighbour (K-NN) [54, 228] and case-based reasoning (CBR) [1, 139, 224]. Their aim is to classify a new object depending on the similarity of the new object’s properties to properties of *k* sample objects in the dataset.
- *Classification and Regression Trees (CART)* [28]: This is a method which, for a categorical dependent variable, produces a classification tree, and for a numeric dependent variable produces a regression tree.

- *Inductive logic programming (ILP)* [16, 144, 174]: This approach integrates induction techniques with logic programming.
- *Genetic algorithms (GA)* [51, 82, 113, 167]: These are stochastic algorithms used in solving various optimisation problems.
- *Support vector machines (SVM)* [50, 109, 225]: These non-linearly map the input data vectors into a high dimensional feature space. Then, in classification tasks, an optimal hyperplane in this space is built to separate the classes of objects. In the regression task a linear regression is performed in this space with the cost function ignoring training data which are close to the model prediction.

Bayesian Approaches

Bayesian analysis (BA) [17, 85, 145, 252] is a process of inductive reasoning. One of its most important features is that it can combine both sample data and prior information (expert judgement) to perform inferences. To achieve this Bayes' Theorem is used to produce posterior probability distributions for model parameters during the model learning process. The next level in Bayesian analysis is a Bayesian network (BN) [124, 177, 189, 252]. BNs are graphical models where pairs of nodes (representing model variables) are linked together (using conditional probability). BNs perform analyses without the need to provide a complete set of values for all predictors. BNs can be developed by learning its structure from the dataset using a learning algorithm. Alternatively, the model structure can be built by a domain expert and the unconditional and conditional probabilities are calibrated using a dataset. Finally, both the model structure and the probability distributions can be defined by expert judgement. BNs are discussed in detail in Chapter 4.

Table 2-2 summarizes the application of the above methods to various software engineering problems. The following abbreviations are used when referring to the analysed methods: BA – Bayesian analysis, BN – Bayesian nets, CART – classification and regression trees, EA – estimation by analogy, EJ – expert judgement, FS – fuzzy sets, GA – genetic algorithms, ILP – inductive logic programming, NN – neural networks, PM – parametric models, RS – rough sets, SD – system dynamics, SVM – support vector machines.

Table 2-2 Summary of applications of modern methods in software engineering

| Author | Dependent variable or problem analysed Main conclusions | Method |
|------------------------------|---|-------------------------------------|
| Dai et al. [52] | reliability | BA |
| Moses [172] | assessing qualitative software attributes expressed by human-raters | BA |
| Moses et al. [173] | productivity, effort | BA |
| Ramírez Cid and Achcar [216] | reliability | BA |
| Seok and Simmons [227] | effort | BA |
| Yang [257] | reliability | BA |
| Zhang and Tsai [260] | <p>various aspects of development effort and software quality</p> <p>The authors assess each of the methods analyzed using various criteria, e.g.: required domain knowledge (DK), training data (TD), advantages (A), disadvantages (D):</p> <ul style="list-style-type: none"> • BA – DK: Required in terms of prior probabilities. TD: Incrementally decrease or increase the estimated probability. A: Flexible in learning target function. Probabilistic prediction. D: Requiring many initial probabilities. Computational cost. • CART – DK: Not required. TD: Adequate data needed to avoid overfitting. Missing values tolerated. A: Robust to noisy data. Capable of learning disjunctive expressions. D: Overfitting. • EA – DK: Not required. TD: Plenty data needed. A: Training is fast. Can learn complex functions. Do not lose information. D: Slow at query time. Curse of dimensionality. • GA – DK: Not required. TD: Not needed (some test data may be needed for fitness evaluation). A: Suited to tasks where functions to be approximated are complex. Algorithms can be easily parallelized. D: Crowding. Bloating. • NN – DK: Not required. TD: Need to have plentiful data. A: Robust to errors in training data. Can learn complex functions (non-linear, continuous functions). Parallel, distributed learning process. D: Slow training and convergence process. Multiple local minima in error surface. Overfitting. • ILP – DK: Required. TD: Organized as positive and negative examples. A: Expressive and human readable representation of learned function. Induction formulated as inverse of deduction. Background knowledge guided search. D: Not robust to noisy data. Intractable search space in general case. Increased background knowledge results in increased complexity of hypothesis space. No guarantee to find the smallest or best set of rules. <p>“When a given task is data-rich, methods of inductive learning can be considered. If there exists a well-defined model for a task, then we can adopt analytical learning methods. Two paradigms can be combined to form a hybrid inductive-analytical learning approach. We can utilize hybrid methods in situations where both data and domain theory are less than desirable. Methods of either paradigm will be good candidates if a task has both an adequate domain theory and plenty of data.”</p> | BA, CART, EA, GA, NN, ILP and other |
| Hewett [111] | <p>defect repair time</p> <p>The author found that predictions obtained using CART were the most accurate.</p> <p>“It is surprising that SVM performs poorly; it is one of the best performance learners and has increasingly gained popularity in recent years, particularly in Bioinformatics. Perhaps SVM is designed to do well on high dimensional data but not on large sample sets.”</p> | BA, CART, EA, NN, SVM |

| Author | Dependent variable or problem analysed Main conclusions | Method |
|----------------------------|---|----------------|
| Stewart [237] | <p>effort</p> <p>For the analysed models “the results show a wide variation in predictive accuracy, depending on the data set used”.</p> <p>“Neural networks achieved superior performance in almost all experiments on training data. [...] Neural networks are prone to over-fitting and very high accuracy on training data may indicate over-fitting”.</p> <p>“Although the predictive performance achieved in our experiments is far from satisfactory for practical applications, it needs to be stressed that the data sets used were relatively small and contained a wide variety of projects from different countries, business areas, and application types”.</p> <p>“Unlike decision trees Naive–Bayes is not sensitive to noise in the data”.</p> <p>“Naive–Bayes is less affected by the problem of over-fitting than neural networks”</p> <p>“Our empirical experiments have shown that the Naive–Bayes classifier is a valuable tool for analysis of software engineering data which can be used as an alternative to other more widely used approaches such as decision trees and neural network”.</p> | BA, CART, NN |
| Challagulla et al. [37] | <p>number of defects, fault-prone modules (4 datasets analysed)</p> <p>“Naive Bayes performed the best for 3 datasets and was third in the remaining data sets”.</p> <p>EA “performed the best for one data set, while being second in two other data sets and third in the remaining data sets”.</p> <p>“Neural networks was second and third in two data sets, while it did not finish in the top three for the other two datasets”.</p> <p>“No particular learning technique that performs the best for all the data sets”.</p> | BA, EA, PM, NN |
| Chulani and Boehm [43, 56] | <p>effort</p> <p>The authors analysed how applying BA can mitigate some of the problems of multiple regression; “the Bayesian approach was better and more robust than the multiple regression approach”</p> | BA, PM |
| Van Koten [246] | <p>effort</p> <p>The authors found that a clear advantage of Bayesian models is lower volume of data required to calibrate the models than for regression models. For example, BMVN (Bayesian multivariate normal distribution) was significantly more accurate than linear regression model after training using just five or fewer number of past projects.</p> | BA, PM |
| Bibi and Stamelos [18] | effort | BN |
| Cockram [45] | software inspection effectiveness | BN |
| Fenton et al. [67, 170] | effort, functionality, quality delivered, user satisfaction | BN |
| Fenton et al. [68, 69, 75] | number of defects | BN |
| Pai et al. [184] | process of software independent verification and validation | BN |
| Wooff et al. [254] | number of defects, degree at which test cases match given situation, effort for testing | BN |
| Bai et al. [11] | <p>reliability</p> <p>Authors compared the results from their model with the results from classic reliability models: Jelinski-Moranda (JM) and Goel-Okumoto (GO). This comparison indicated that JM and GO models are not able to predict time between failures for the early failures. Additionally they pointed out that with the increase of amount of available data</p> | BN, PM |

| Author | Dependent variable or problem analysed Main conclusions | Method |
|-----------------------------|---|-----------------------|
| | about subsequent failures JM and GO models are overestimating while predictions from their model are stable with small variance. | |
| Boehm [26] | decision whether to perform independent validation and verification | CART |
| Francis et al. [81] | classifying failures if they result of the same or similar causes “Examining the individual decisions made by a classification tree used to group failures is not very helpful for diagnosing the cause of the failures”. | CART |
| Khoshgoftaar et al. [135] | fault-proneness modules | CART |
| Huang and Chiu [114] | effort “Analogy-based software effort estimation models are intolerant of noise and irrelevant effort drivers.” “Applying GA to analogy-based software effort estimation models is a feasible approach that can provide objective weights for software effort drivers rather than the subjective weights assigned by experts. It also reveals that the analogy-based estimation model with nonlinearly weighted similarity measures produces superior prediction accuracy compared to results from the linearly weighted analogy and unequally weighted similarity measures.” The authors found that combining GA with EA produced more accurate results than using CART, NN or PM (ordinary least squares). | CART, EA + GA, NN, PM |
| Ceylan et al. [36] | fault-prone code “All of the learning algorithms used in our research have similar prediction performances having similar mean square error values. However, the radial basis function (RBF) method has slightly smaller prediction errors than the other two methods” (CART and Multi Layer Perceptron). | CART, NN |
| Srinivasan and Fisher [233] | effort “techniques are competitive with traditional estimators on one dataset, but also illustrate that these methods are sensitive to the data on which they are trained” | CART, NN |
| Gokhale and Lyu [90] | number of faults The authors found that predictions provided by regression trees were more accurate. | CART, PM |
| Schröter et al. [226] | fault-prone components The authors found that SVMs provided the most accurate results, regression trees were only slightly less accurate while both linear regression and ridge regression were least accurate | CART, PM, SVM |
| El Emam et al. [62] | fault-prone components The authors compared the performance of 30 various classifiers varying in size measures, standardization techniques, use or non-use of weights and the number of nearest neighbours. They stated that “there is no difference in prediction performance when using any combination of parameters”. However they suggest using classifier with Euclidean distance, z-score standardization, no weighting scheme and a single nearest neighbour. Such classifier is more intuitive for non-specialists and performs as well as other more complicated classifiers | EA |
| Shepperd and Kadoda [229] | effort The authors proposed a set of rules suggested to be taken into consideration when deciding on selection of specific method. For example: <ul style="list-style-type: none"> • stepwise regression is preferred for continuous, especially normally distributed, dependent variable when a dataset does not contain outliers or collinearity. • CBR is preferred for discretised dependent variable and when the | EA, CART, PM, NN |

| Author | Dependent variable or problem analysed Main conclusions | Method |
|------------------------------|---|-----------------|
| | dataset contains outliers and collinearity. “ML approaches always benefited from having larger training sets”. | |
| Angelis and Stamelos [9] | effort | EA |
| Mendes et al. [161, 162] | effort “The best predictions were obtained for the data set that presented a continuous <i>cost</i> function, reflected as a strong linear relationship between size and effort, and that was more <i>unspoiled</i> (no outliers, small collinearity)” [161] | EA |
| Jørgensen et al. [129] | effort Software estimation by analogy by incorporating the concept of ‘regression toward the mean’ (RTM) where estimates were adjusted by more ‘average’ projects. This can be useful when selected projects have unusually low or high productivity. Their results confirmed that introducing RTM improved produced estimates. They also compared their results with estimates provided by software professionals (expert judgement) and observed that the latter were to some extent inherently RTM-adjusted | EA, EJ |
| Chiu and Huang [39] | effort The authors used a GA to adjust reused effort based on similarity distances between pairs of projects. Their empirical results showed that “applying a suitable linear model to adjust the analogy-based estimations is a feasible approach to improving the accuracy of software effort estimates”. | EA, GA |
| Kirsopp and Shepperd [137] | effort | EA |
| Kirsopp et al. [136] | effort | EA |
| Mair et al. [154] | effort “ANNs seem to be the most accurate technique and there is little to choose between CBR and LSR” (least square regression). ILP is “consistently the least accurate technique”. “One of the benefits of rule-induction is that it makes explicit the rules that are being used by the prediction system. This, it is argued, can lead to insights about the data being used. However, the partitions or branches can sometimes appear to be rather arbitrary and reliance upon them as genuinely meaningful indicators may be unwise. In addition, our experience of rule-induction methods suggests that they can be unstable predictors, and possibly less accurate than other techniques.” “CBR or estimation by analogy also has potential explanatory value, since projects or ordered in degree of similarity to the target project.” “Considerable effort was required to configure the ANN and that this compared very unfavourably with the other techniques, particularly CBR and LSR”. The authors ranked the analysed methods using three criteria (form best to worst): • accuracy: (1) NN, (2 equal) PM and CBR, (4) ILP • explanatory value: (1 equal) CBR, PM and ILP, (4) NN • configurability: (1 equal) CBR and PM, (3) ILP, (4) NN | EA, ILP, NN, PM |
| Shepperd and Schofield [230] | effort Analogy based approach outperformed algorithmic models based on stepwise regression in term of predictive accuracy in all cases using 9 analyzed datasets. | EA, PM |
| Walkerden and | effort | EA, PM |

| Author | Dependent variable or problem analysed Main conclusions | Method |
|--------------------------------|---|------------|
| Jeffery [249] | Estimates based on people' selections of analogues were more accurate than those based on tools' selections. They were also more accurate than produced by simple regression models | |
| Li and Ruhe [149] | effort "Proposed attribute weighing method using RSA [rough set analysis] can improve the estimation accuracy of EBA [estimation by analogy] method AQUA+ according to the empirical studies over six data sets" | EA, RS |
| Bajaj et al. [12] | effort | FS |
| Kumar et al. [141] | effort | FS |
| Gray and Macdonell [93] | effort FS model "shows good performance, being out-performed in terms of accuracy only by the neural network model with considerably more input variables" | FS, NN, PM |
| Emer and Vergilio [64] | test case selection | GA |
| Evetts et al. [65] | number of defects | GA |
| Greer and Ruhe [94] | trade-off between cost and functionality | GA |
| Zhang et al [261] | next release problem, trade-off between cost and functionality | GA |
| Cohen and Devanbu [46] | defect density | ILP |
| Khoshgoftaar et al. [133] | testability | NN |
| Quah and Thwin [198] | reliability, maintainability | NN |
| Hochman et al. [112] | fault-proneness modules | NN + GA |
| De Barcelos Tronto et al. [55] | effort NNs and stepwise regression which the authors also used are "competitive" to APF, SLIM and COCOMO methods. | NN, PM |
| Dohi et al. [58] | reliability NNs achieved better predictive performance than parametric methods using software reliability growth models. | NN, PM |
| Khoshgoftaar et al. [134] | faults NNs performed better both in terms of quality of fit and predictive quality. | NN, PM |
| Sitte [231] | reliability | NN, PM |
| Ramanna [215] | effort "Predictive capability of the rough neural network is comparable with RSES" (one of RS techniques). | NN, RS |
| Laplante and Neill [142] | uncertainty in software engineering | RS |
| Peters and Ramanna [190] | effort | RS |
| Ruhe [219] | criticality of software modules, effort | RS |
| Stefanowski [235] | effort | RS |
| Becker-Kornstaedt and Neu [13] | software inspections | SD |
| Christie and Stalley [40] | requirements specification process | SD |
| Collofello et al. [47] | testing process | SD |

| Author | Dependent variable or problem analysed Main conclusions | Method |
|----------------------------|--|--------|
| Pfahl and Ruhe [192] | organisational learning | SD |
| Pfahl et al. [191] | stability of software release plans | SD |
| Raffo et al. [212] | project performance (schedule and resources) | SD |
| Tvedt and Collofello [244] | effectiveness of process improvements | SD |

Most of the studies on applications of learning methods in software engineering conclude that the best method (often based on prediction accuracy) is the one proposed by the authors. Independent analyses were performed in order to determine the most accurate method [36, 37, 111, 154, 260]. However, there is no single machine learning method which would produce the best results for all known datasets.

Table 2-3 shows a comparison of the features of modern methods for software project risk assessment based on the studies summarized in Table 2-2.

One of the most important factors in selecting an appropriate method is how much data it requires to produce a model. There are known publicly-accessible databases of past software projects (summarized in [200]). However, they all have some disadvantages, which limit their potential applicability. The most common weaknesses of these datasets are:

- low volume of data,
- low number of variables,
- missing values,
- lack or short descriptions.

Usually even a single one of the above weaknesses in a given dataset can make it unusable for developing a model. Therefore, we believe a critical criterion for a method is that it does not rely strongly on empirical data but allows expert knowledge.

BNs appear to provide a method with the highest potential in performing various types of analyses, in particular trade-off analysis. This is mainly due to their ability to: capture causal relationships, model intuitiveness, run with an incomplete list of predictors and perform both forward and backward inference.

Table 2-3 Comparison of features of modern methods used in software project risk assessment

| Method \ Feature | System Dynamics | Neural Networks | Fuzzy Sets | Rough Sets | Estimation by analogy | Classification & Regression Trees | Inductive Logic Programming | Genetic Algorithms | Support Vector Machines | Bayesian Approaches |
|---|-----------------|-----------------|------------|------------|-----------------------|-----------------------------------|-----------------------------|--------------------|-------------------------|---------------------|
| Ability to model causal/influential relationships | VH | No | No | H | No | H | VH | No | No | VH |
| Reasoning from cause to effect (forward) and from effect to cause (backward) | No | Yes | Yes | No | Yes | No | Yes | No | No | Yes |
| Ability to incorporate uncertainty (for dependent variable) | No | M | VH | VH | M | VH | M | No | Yes | VH |
| Ability to combine heterogeneous data (numerical and nominal - not ordered) | No | No | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| Required volume of the past data | M | M | M | M | L | L | L | L | M | L |
| Intuitiveness (understanding) | H | L | H | H | M | M | H | M | M | H |
| Ease of use (by non-expert) | H | M | M | M | H | H | H | M | M | H |
| Model adaptability | M | M | M | L | M | M | H | M | M | H |
| Model execution time | M | M | M | M | L | L | M | M | M | H |
| Ability to build models with continuous dependent variable | Yes | Yes | Yes | No | Yes | Yes | Yes | Yes | Yes | Yes |
| Ability to perform different types of analyses (what-if, goal-seeking, trade-off) | M | H | M | M | M | H | VH | M | M | VH |
| Ability to run the model with incomplete set of predictors | No | No | No | No | Yes | No | No | No | No | Yes |
| Ability to create a model without any empirical data (only by expert) | Yes | No | Yes | Yes | No | Yes | Yes | Yes | No | Yes |
| 'VH' – very high, 'H' – high, 'M' – medium, 'L' – low | | | | | | | | | | |

2.6 Summary

We have analysed the limitations of classic parametric-type models and have considered the various modern approaches that attempt to overcome these limitations. However, most of the modern approaches suffer from the problem of low availability of relevant and reliable empirical data. Although BNs do not come out best under all the various requirements, they best satisfy the most important requirements and are thus highly suitable as a modelling method. The next chapter discusses the most relevant publicly available empirical data which were used to inform existing and new BNs.

3 Review of most recent empirical data

This chapter discusses various empirical results which we used to:

- identify relevant factors in our models (Chapters 4, 6, 8, 9, 10),
- identify relationships which helped in defining proper relationships between variables in our models (Chapters 4, 6, 7, 8, 9, 10),
- define prior probabilities in our models (Chapters 6, 8, 9, 10),
- validate our models (Chapters 8, 9, 10).

Apart from this data we have also used other sources indirectly. They are listed in Appendix A. We believe that both the data presented in this chapter and in Appendix A is essentially a summary of the state-of-the-art of publicly available empirical data which can be used by other researchers and software companies. Most of the data summarized in this chapter is categorized by area of applicability (effort, defects etc.). However, there is some data which does not fall into any other categories or which contains data from multiple areas. These are discussed in the Section 3.1, which is based on a previously published paper [200].

The new contribution of this chapter is the survey and classification of publicly available empirical data.

3.1 *Various data*

In [200] I performed a review of publicly available repositories with software engineering datasets. Table 3-1 summarizes the main advantages and disadvantages of the analyzed repositories. Based on the analysis of these datasets we can point to the following areas in which specific repositories and databases can be used:

- software size estimation – datasets containing detailed data on each module (mainly NASA datasets),
- development effort estimation – datasets containing data on effort (ISBSG and some from PROMISE),
- defect estimation – datasets containing data on number of defects, occurrence of defects in specific modules, history of defects and other issues (almost all databases except some from PROMISE),

- analysis of relationships between size, effort and defects – datasets containing all of these parameters (ISBSG and some from PROMISE),
- analysis of impact of process and people quality – datasets containing process and people data (some from PROMISE).

Table 3-1 Summary of advantages and disadvantages of most popular publicly available software engineering repositories

| Repository | Advantages | Disadvantages |
|------------------------------|---|--|
| ISBSG [121] | <ul style="list-style-type: none"> • high project variation • unified list of parameters describing a project | <ul style="list-style-type: none"> • high number of missing values • lack of factors describing process and people quality |
| PROMISE [27] | <ul style="list-style-type: none"> • high variation of subject databases – areas of applications for described projects and parameters describing projects • additional information about the data and their use – publications, models • low number of missing values | <ul style="list-style-type: none"> • lack of unified list of parameters describing a project • common appearance of parameters from one area only (size, defects, effort) |
| NASA MDP [165] | <ul style="list-style-type: none"> • detailed data on defects down to the level of problem report • high number of parameters describing software modules and defects | <ul style="list-style-type: none"> • difficult usage of source data caused by saving them in multiple files • high number of missing data about defects in some databases • lack of data about effort • lack of parameters describing process and people quality • projects only from NASA operation area |
| using Bugzilla [241, 242] | <ul style="list-style-type: none"> • detailed data on defects down to the level of problem report • various ways of data presentation: reports, charts, lists | <ul style="list-style-type: none"> • focus on defects only – lack of effort data and parameters describing process and people quality • lack of easy access to raw data |

Only two of the analyzed datasets contain data about software size, effort, defects and process and people quality:

- PROMISE: qqdefects – data about 31 projects,
- PROMISE: ivvbayes – data about 4 projects.

Some limitations of publicly available databases are summarized in Table 3-1, but another limitation is the anonymity of the data. In most cases it is not possible to find the methodology by which data was collected for a specific project, so it is not possible to discover such issues as: who provided the data, what sources were used to provide the data, what criteria were used in assigning values for categorical variables and what was the environment in which the data were collected. This causes datasets to be less reliable and interpretation of the data may be incorrect without additional information. Some of these issues are discussed in subsequent analyses where datasets are used,

notably: in Section 4.10 (the 'qqdefects' dataset from PROMISE); in Chapters 8 and 9 (subsets of the ISBSG dataset); and Chapter 10 (the NASA MDP datasets).

3.2 Trade-offs in software development

Putnam tried to answer the question: “how much software can a typical IT shop create in 12 months?” [195]. The research was based on data from 281 projects completed in 2000 and 2001: 119 of them were Department of Defence (DOD) projects and 162 were from the private sector. All the data was from the QSM, Inc. database. The primary application domain for those projects was Business (IT) projects. Those projects included “new development, major enhancements, and minor enhancements; package modifications; object-oriented projects; and web developments”. Project team sizes ranged from less than 1 full-time equivalent person to 129 people [195]. Table 3-2 presents the major findings of this research.

Table 3-2 Major results from Putnam's research

| Factors \ Projects | DOD projects | Industry projects | All projects |
|--|--------------|-------------------|--------------|
| Number of projects | 119 | 162 | 281 |
| Percent of projects delivering more than 75 EKLOC in 12 months | 8% | 11% | 10% |
| Staff required for projects exceeding 75 EKLOC | | | 20-100 |
| Average staff required for projects less than 75 EKLOC | | | 5-10 |
| Maximum EKLOC in less than 12 months | 335.1 | 227.91 | |
| Average EKLOC in more than 12 months | 72.508 | 59.937 | |
| Average EKLOC in less than 12 months | 23.178 | 21.469 | |
| Percent of projects in less than 12 months | 71.4% | 81.5% | |
| Schedule from average trend at 75 EKLOC | 12.4 months | 10 months | |

Putnam stated that the practical upper limit for a 1-year project is about 180 KLOC, using 70-100 people. The costs of such a project should be 2-4 times higher than similar ones without the 12 months deadline. Putnam also states that it is possible to build software of 50 to 75 KLOC in 12 months with a team of 3-10 people. Building software larger than 75 KLOC in a year is possible but requires:

- “significantly greater staffing and cost,
- a relatively low level of complexity in the developed software,
- recognition that the risk of exceeding 12 months is high,
- recognition that the risk of low reliability is even greater” [195].

Galorath et al. [84] also analyzed trade-offs between key project variables in software projects. Table 3-3 summarizes their trade-off report achieved in a software project. They state that applying more testing increased total development effort by 18% which also led to an increase in total development cost of 18%. It also increased the peak number of people needed on the project by 11% but caused a lower number of residual defects after software release of 45%.

MacCormack et al. [153] performed empirical research on trade-offs between productivity and quality. They used data from 29 projects between summer of 2000 and winter of 2001. Table 3-4 illustrates selected descriptive statistics for those projects. Table 3-5 summarizes the results of their statistical analysis – correlations between productivity and defect rates with selected predictors.

Table 3-3 Cost/schedule/defect trade-off report [84]

| | More testing | Less testing | Difference |
|-----------------------------|--------------|--------------|------------|
| Development Schedule Months | 31.54 | 29.89 | 6% |
| Development Effort Months | 1422.39 | 1210.15 | 18% |
| Development Base Year Cost | \$20909062 | \$17789243 | 18% |
| Defect Prediction | 33 | 60 | -45% |
| Constraints | MIN TIME | MIN TIME | |
| Peak Staff | 63.59 | 57.10 | 11% |

Table 3-4 Selected descriptive statistics for projects analyzed in [153]

| Variable | Description | Mean | Median | Standard deviation | Min. | Max. |
|--------------------------|---|------|--------|--------------------|------|------|
| Defect rate | Average no. of customer-reported defects per month per million lines of new code over first 12 months | 18.8 | 7.1 | 23.1 | 0.0 | 80.0 |
| Productivity | New LOC developed per person-day | 26.4 | 17.6 | 24.0 | 0.7 | 85.0 |
| Functional specification | Percentage of functional specification that was complete before team started coding | 55% | 55% | 32% | 0% | 100% |
| Design specification | Percentage of detailed design specification complete before team started coding | 20% | 10% | 26% | 0% | 80% |
| Design review | Binary: 1 if design reviews were performed during development, 0 if not | 0.79 | 1 | 0.41 | 0 | 1 |
| Code review | Binary: 1 if the number of people who typically reviewed another person's code was one or more, 0 if none | 0.52 | 1 | 0.51 | 0 | 1 |

Table 3-5 Simple correlations with model performance [153]

| Control variables | Defect rate | Productivity |
|--|--------------------|---------------------|
| Systems projects | 0.436** | 0.030 |
| Applications | -0.058 | -0.018 |
| Embedded | 0.014 | 0.066 |
| Size: Ln(LOC) | -0.562*** | 0.514*** |
| Process variables | | |
| Functional specification | 0.035 | 0.473** |
| Design specification | -0.403* | -0.014 |
| Early prototype [†] | 0.742**** | -0.624*** |
| Subcycles | -0.418* | 0.021 |
| Daily builds | -0.026 | 0.316 |
| Regression test | -0.383* | 0.180 |
| Design review | -0.545** | -0.227 |
| Code review | -0.255 | 0.104 |
| [†] : Implies less functionality in the first prototype * p < 10%, ** p < 5%, *** p < 1%, **** p < 0.1% Correlations in bold are significant (p < 0.05) | | |

3.3 Project management

Jones [125] identified the top five most common risks that threaten projects in various application sectors. These risks are summarized in Table 3-6.

Table 3-6 Most common risk factors in depending on application sector

| Project Sector | Risk Factor | Percent of Projects at Risk |
|--------------------------------|----------------------------------|-----------------------------|
| Management information systems | Creeping user requirements | 80% |
| | Excessive schedule pressure | 65% |
| | Low quality | 60% |
| | Cost overruns | 55% |
| | Inadequate configuration control | 50% |
| Commercial | Inadequate user documentation | 70% |
| | Low user satisfaction | 55% |
| | Excessive time to market | 50% |
| | Harmful competitive actions | 45% |
| | Litigation expense | 30% |

Putnam [196] tried to answer the question: “what is the optimum team size for medium sized software projects?” He defined medium sized software as containing from 35 to 95 new or modified KLOC. His research was based on data from 491 projects. The average size of software taken into research was 57,412 EKLOC. He measured team productivity using a “Productivity Index” (a measure dependant on size, schedule time and development effort). Table 3-7 displays the main results of his research. The author concludes that the optimum team size is 3-5 people. Although this group did not win in any assessment categories, it was very good in all areas. A 5-7 person team was second but was very close to the winning group.

Table 3-7 Main results of research to find optimum team size; based on data from [196]

| Team size | Productivity Index | Schedule in months | Effort |
|-----------|--------------------|--------------------|--------|
| 1.5 – 3 | 16.36 | 13.6 | 31 |
| 3 – 5 | 16.29 | 11.9 | 48 |
| 5 – 7 | 16.18 | 11.6 | 69 |
| 9 – 11 | 13.72 | 17.1 | 167 |
| 15 – 20 | 13.03 | 16.29 | 283 |

Zahran [259] analyzed differences in productivity and achieved software quality between best and least performing software companies in Europe. Table 3-8 summarizes the main results of his research. He observed that the difference in achieved quality is significantly higher than in achieved productivity.

Table 3-8 Performance of the best and worst performing software organizations in Europe; adopted from [259 p. 393]

| Area of comparison | Leaders (best performing) | Laggers (worst performing) | Relative performance Leaders versus Lagers |
|---|---------------------------|----------------------------|--|
| Development productivity | >25 FPs per person-month | <5 FPs per person-month | >5 times better |
| Defect Removal (before delivery) | >95% | <50% | >30 times better |
| Estimate consistency (of cost and duration) | <10% | >40% | >30 times better |
| Defect correction after delivery (percent of development effort on defect correction in the first 12 months after delivery) | <1% | >10% | >10 times better |

Table 3-9 summarizes the productivity achieved in 6 software projects developed using an object-oriented approach and with different programming languages.

Table 3-9 Productivity for object-oriented projects [130 p. 344]

| Project \ Metric | Project A (C++) | Project B (C++) | Project C (C++) | Project D (IBM Smalltalk) | Project E (OTI Smalltalk) | Project F (Digitalk Smalltalk) |
|--------------------------|-----------------|-----------------|-----------------|---------------------------|---------------------------|--------------------------------|
| Number of Classes | 5,741 | 2,513 | 3,000 | 100 | 566 | 492 |
| Person-Years | 100 | 35 | 90 | 2 | na | 10 |
| Classes per Person-Year | 57.4 | 71.8 | 33.3 | 50 | na | 49.2 |
| Classes per Person-Month | 4.8 | 6 | 2.8 | 4.2 | na | 4.1 |
| Methods per Person-Month | 8 | 18 | 20 | 71 | na | 86 |

Table 3-10 summarizes COCOMO II model multipliers [43 p. 582]. The differences in these values between state ‘VL’ (very low) and ‘VH’ (very high) or ‘XH’ (extra high) – where applicable – indicate the strength of the impact of a particular factor on development effort as it is captured in this model. The last column ‘PR’ reflects the ratio of the highest and the lowest values of parameter. Thus it reflects the importance of each parameter.

Table 3-10 COCOMO II multipliers [43 p. 582]

| COCOMO II Parameter | | VL | L | N | H | VH | XH | PR |
|---|---|------|------|------|------|------|------|------|
| PREC | Precendentness | 6.20 | 4.96 | 3.72 | 2.48 | 1.24 | 0.00 | 1.33 |
| FLEX | Development Flexibility | 5.07 | 4.05 | 3.04 | 2.03 | 1.01 | 0.00 | 1.26 |
| RESL | Architecture and Risk Resolution | 7.07 | 5.65 | 4.24 | 2.83 | 1.41 | 0.00 | 1.38 |
| TEAM | Team Cohesion | 5.48 | 4.38 | 3.29 | 2.19 | 1.10 | 0.00 | 1.29 |
| PMAT | Process Maturity | 7.80 | 6.24 | 4.68 | 3.12 | 1.56 | 0.00 | 1.43 |
| RELY | Required Software Reliability | 0.82 | 0.92 | 1.00 | 1.10 | 1.26 | | 1.53 |
| DATA | Data Base Size | | 0.90 | 1.00 | 1.24 | 1.28 | | 1.42 |
| CPLX | Product Complexity | 0.73 | 0.87 | 1.00 | 1.17 | 1.34 | 1.74 | 2.39 |
| RUSE | Develop for Reuse | | 0.95 | 1.00 | 1.07 | 1.15 | 1.24 | 1.31 |
| DOCU | Documentation Match to Life-cycle Needs | 0.81 | 0.91 | 1.00 | 1.11 | 1.23 | | 1.52 |
| TIME | Time Constraint | | | 1.00 | 1.11 | 1.29 | 1.63 | 1.63 |
| STOP | Storage Constraint | | | 1.00 | 1.05 | 1.17 | 1.46 | 1.46 |
| PVOL | Platform Volatility | | 0.87 | 1.00 | 1.15 | 1.30 | | 1.50 |
| ACAP | Analyst Capability | 1.42 | 1.19 | 1.00 | 0.85 | 0.71 | | 2.00 |
| PCAP | Programmer Capability | 1.34 | 1.15 | 1.00 | 0.88 | 0.76 | | 1.77 |
| AEXP | Applications Experience | 1.22 | 1.10 | 1.00 | 0.88 | 0.81 | | 1.51 |
| PEXP | Platform Experience | 1.19 | 1.09 | 1.00 | 0.91 | 0.85 | | 1.40 |
| LTEX | Language and Tool Experience | 1.20 | 1.09 | 1.00 | 0.91 | 0.84 | | 1.43 |
| PCON | Personnel Continuity | 1.29 | 1.12 | 1.00 | 0.90 | 0.81 | | 1.59 |
| TOOL | Use of Software Tools | 1.17 | 1.09 | 1.00 | 0.90 | 0.78 | | 1.50 |
| SITE | Multi-Site Development | 1.22 | 1.09 | 1.00 | 0.93 | 0.86 | 0.80 | 1.52 |
| SCED | Required Development Schedule | 1.43 | 1.14 | 1.00 | 1.00 | 1.00 | | 1.43 |
| Multiplicative Effort Calibration Constant (A) = 2.94 | | | | | | | | |
| Exponential Effort Calibration Constant (B) = 0.91 | | | | | | | | |

3.4 Defect prediction

Jones [126 cited after 130 p. 96] performed a study of a large body of empirical results. The average number of defects found and measured in software developed in the U.S. throughout the whole software life cycle (including defects reported by users) is 5 per FP. Jones has also estimated the defect removal efficiency of software organizations depending on their CMM level. After applying it to the overall defect rate per function point (FP), defect rates for the maintenance life of the software were estimated (Table 3-11).

**Table 3-11 Estimated defect rates per function point depending on CMM level
[126 cited after 130 p. 96]**

| CMM Level | Estimated defect rates per FP |
|-----------|-------------------------------|
| 1 | 0.75 |
| 2 | 0.44 |
| 3 | 0.27 |
| 4 | 0.14 |
| 5 | 0.05 |

Table 3-12 summarizes the effectiveness of various stages of software development in CMM level 4 companies in terms of the cumulative percent of defects removed after each of these stages.

Table 3-12 Cumulative percentages of defects removed by development phase for organizations at CMM Level 4 [130 p. 181]

| Development phase | Cumulative percent of defects removed through acceptance test |
|--------------------|---|
| Requirements | 94% |
| Top-level design | 95% |
| Detailed design | 96% |
| Code and unit test | 94% |
| Integration test | 75% |
| System test | 70% |
| Acceptance test | 70% |

Table 3-13 illustrates various measures of software quality and process efficiency for software of various sizes. We can observe that as project size increases *defect potential* and *defect density* also increase and *defect removal efficiency* decreases.

**Table 3-13 Software size versus defect potential, defect removal efficiency, and defect density
[127 cited after 222]**

| Size (FPs) | Defect potential (development) | Defect removal efficiency | Defects density (released) |
|------------|---|---------------------------|----------------------------|
| | (defect potential and density expressed in terms of defects per FP) | | |
| 1 | 1.85 | 95% | 0.09 |
| 10 | 2.45 | 92% | 0.20 |
| 100 | 3.68 | 90% | 0.37 |
| 1000 | 5.00 | 85% | 0.75 |
| 10000 | 7.60 | 78% | 1.67 |
| 100000 | 9.55 | 75% | 2.39 |
| Average | 5.02 | 86% | 0.91 |

Table 3-14 summarizes defect potentials depending on defects injected in various stages of software development for leading, average and lagging software companies.

Table 3-14 Comparison of defect potentials and removal effectiveness [128 cited after 222]

| Task | Leading | Average | Lagging |
|--------------|---|---------|---------|
| | (data expressed in terms of defects per FP) | | |
| Requirements | 0.55 | 1.00 | 1.45 |
| Design | 0.75 | 1.25 | 1.90 |
| Coding | 1.00 | 1.75 | 2.35 |
| User Manuals | 0.40 | 0.60 | 0.75 |
| Bad fixes | 0.10 | 0.40 | 0.85 |
| Total | 2.80 | 5.00 | 7.30 |
| Removal % | 95% | 85% | 75% |
| Delivered | 0.14 | 0.75 | 1.83 |

In a study comparing software defects in open and closed software, the authors found that the defect density for Apache v2.1 was 0.53 defects per KLOC while the commercial defect density was about 0.51 defects per KLOC of source code. The author states that “there is a correlation between code inspection/peer review and the resulting defect density” but there is no numerical data on the efficiency of inspection/reviews [148].

Humphrey estimated that experienced software developers normally inject 100 or more defects per KLOC [115].

Table 3-15 summarizes defect removal effectiveness throughout a software project for organizations at different CMM levels.

Table 3-15 Estimated defect removal effectiveness for organizations at different CMM levels [130 p.181]

| CMM Level | 1 | 2 | 3 | 4 | 5 |
|------------------------------|-----|-----|-----|-----|-----|
| Defect Removal Effectiveness | 85% | 89% | 91% | 93% | 95% |

Figure 3-1 illustrates the relative code defect introduction ranges for the 21 defect drivers used in the COQUALMO model [212]. Names of these drivers are provided in Table 3-10. As an example of how to interpret this figure, consider PMAT (process maturity). The difference in the predicted number of defects between very low and extra high process maturity is 2.5 times (assuming all other factors are constant). Thus, the higher value for each defect driver indicates the higher relative importance of the specific driver.

We have analyzed the NASA MDP datasets containing data about the number of defects found and fixed. The dataset is organized in such way that each defect has assigned dates when it was discovered and fixed. Some defects do not have assigned dates of detecting and fixing. We did not use datasets that contained large numbers of

such defects. Figure 3-2 illustrates the numbers of defects found and fixed grouped by month. Numeric values are summarized in Appendix E – Section E.3.

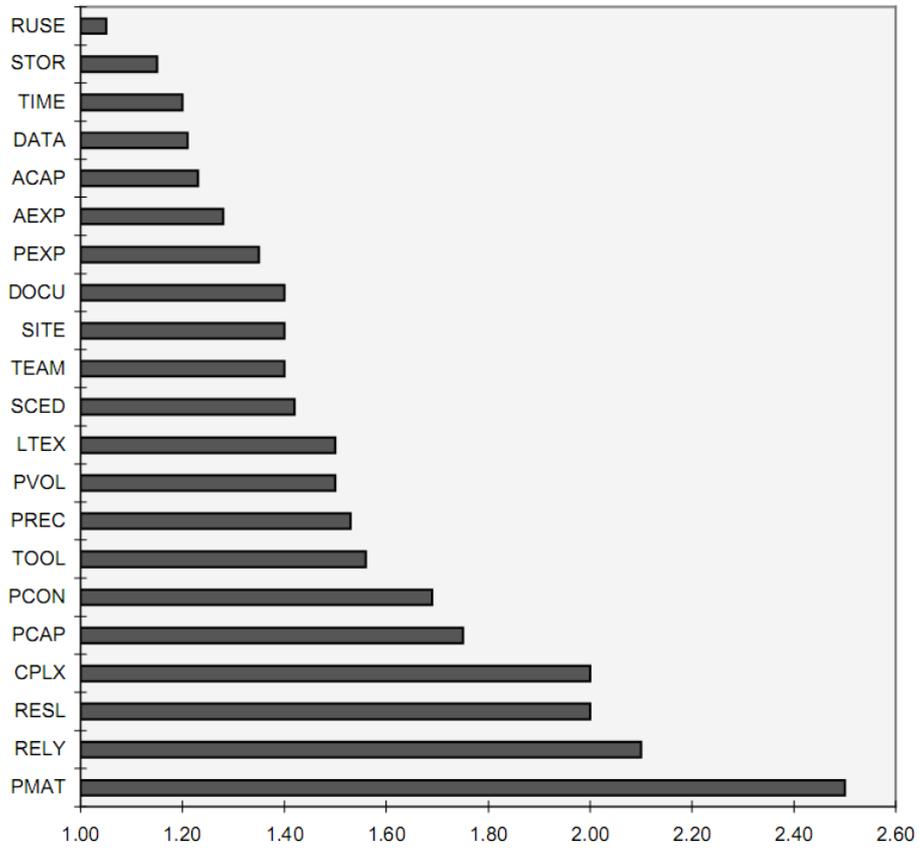


Figure 3-1 Coding defect introduction ranges [212]

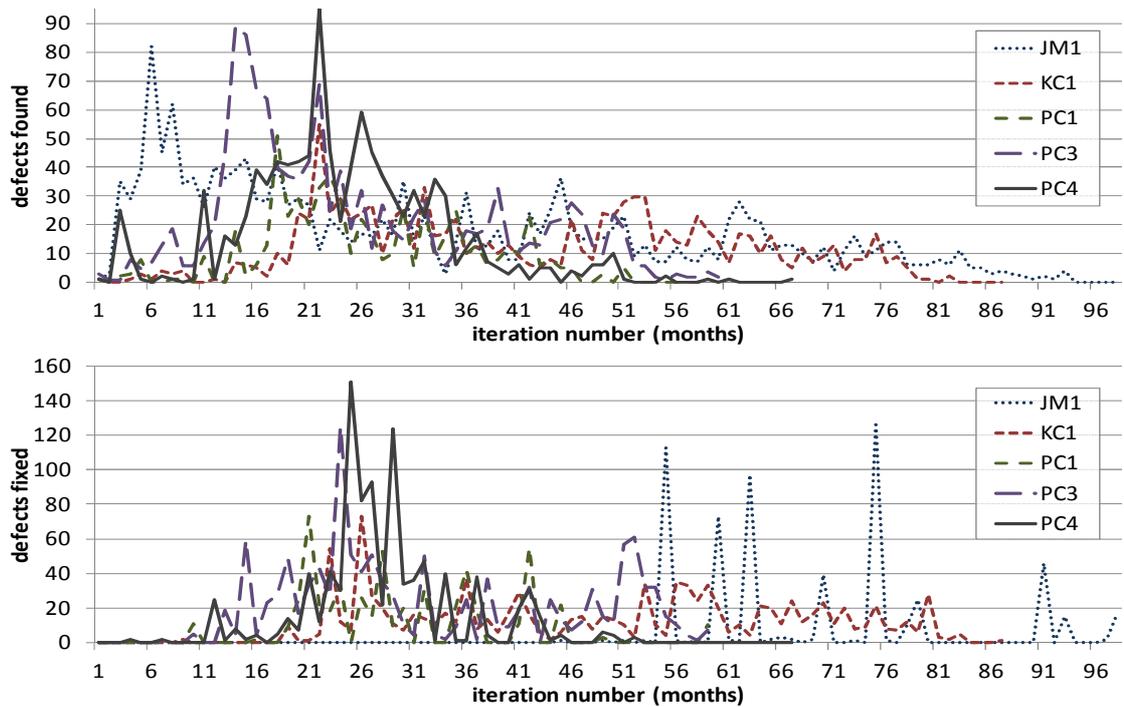


Figure 3-2 Number of defects found and fixed in each iteration in various NASA datasets

3.5 Requirements management

The most common symptoms of errors in a requirements specification are [35, 208]:

- system does not pass acceptance tests despite passing tests during system development,
- project schedule overrun,
- budget overrun,
- system efficiency lower than expected by users,
- designers and programmers forced to repeating their work,
- end-users use system functions in unexpected way, do not use many available functions or the whole system at all,
- many suggestions about system improvement appear shortly after its delivery.

The positive effects of applying concepts like CMMI, various ISO norms, rapid application development (RAD), joint application development (JAD), quality function deployment (QFD) or Six-Sigma have been observed. But none of them can be regarded as a 'silver-bullet' [132 pp. 591-600]. Success in applying these methods depends on other conditions [49, 208]:

- creating a strategic plan for a company based on which the aim for the system is clearly stated,
- assigning an appropriate budget and duration for preparing the requirements specification,
- realising the weaknesses in the organisation,
- constant user education,
- standardising the methods and techniques used,
- a professional approach covering fair cost estimation, preparing gathering, requirements, analysis and controlling requirements.

3.6 Summary

The empirical data discussed in this chapter can be used in various areas of software engineering. However, the main problem with such data is that very often they are incomplete in the sense that they are typically focused on only one aspect of software development. Empirical data covering various size metrics, development effort

and defects is usually not available. This causes a difficulty in building models for software project risk assessment relying only or mostly on the data. Available empirical data can still be used in combination with expert judgement in developing predictive models. The data discussed in this chapter was used to inform some of the existing BNs as well as the new BNs. The next chapter provides a brief introduction to the theory of BNs and discusses the BNs which are the basis for developing new models.

4 Review of Bayesian Networks for Software Project Risk Assessment

This chapter introduces the basic theory of BNs and modelling project risk. Then it focuses on the review of extensively tested BNs to which the author had access. It demonstrates how such models can be used to provide information useful for decision-makers and also discusses the advantages and disadvantages of these models. The new contribution of this chapter is a compact introduction to the BNs and a detailed review of selected existing BN models for software project risk assessment. Section 4.6 is based on a previous paper [207], Sections 4.8-4.11 on [201], and Section 4.10 additionally on [68, 69].

4.1 Introduction to Bayesian nets

A Bayesian net (BN) (also called: Belief network, Bayesian belief network, causal model) is a probabilistic model consisting of two main elements:

1. A Directed Acyclic Graph (DAG) containing:
 - a. A set of nodes reflecting a set of random variables.
 - b. A set of directed links connecting pairs of variables.
2. Probability distributions assigned to the nodes:
 - a. For root nodes (without parents) – unconditional probabilities.
 - b. For nodes having at least one parent node – conditional probabilities.

A simple BN is shown in Figure 4-1.

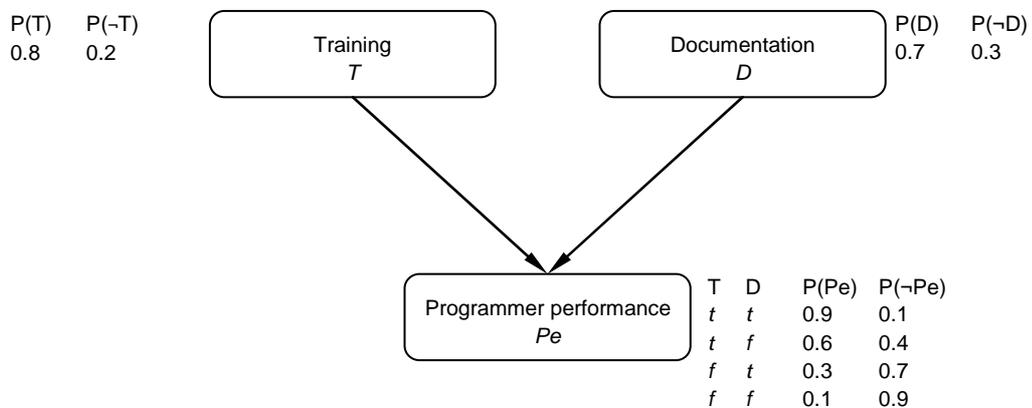


Figure 4-1 Example of simple BN model

Each node in a BN is of a specific type. For example, common types are:

- Boolean – two states “True” and “False”;
- numeric: discrete – the states are point real numbers, e.g.: ‘-1’, ‘0’, ‘0.5’, ‘20.33’, etc.;
- numeric: continuous – the states are intervals between real numbers, e.g.: ‘-10–0’, ‘0–0.5’, ‘0.5–3’, ‘3–15.5’, etc. or point real numbers;
- labelled (nominal) – the states are expressed as words which cannot be transformed to a numeric scale, e.g.: ‘blue’, ‘green’, ‘red’, ‘yellow’.

The conditional probabilities can be defined in the form of:

- Conditional Probability Tables (CPT) – for the discrete and labelled variables,
- Conditional Probability Distributions (CPD) – for the continuous variables.

A BN which contains both discrete and continuous variables is called a ‘Hybrid Bayesian Network’ [179, 220 p. 501].

Usually domain experts can relatively easily determine the important variables to be included in the model as well as the links between them. However, specifying unconditional and conditional probabilities does not seem to be an easy task. In models with many states per variable and nodes with many parents it is usually a difficult or impossible task to elicit consistent CPTs or CPDs manually. Some BN tools, like AgenaRisk [3] which we use in this study, support defining CPTs and CPDs using expressions which often simplifies the task of model preparation and reduces the time needed to build a model. Once the model is built inference in the model is performed based on Bayes’s Theorem:

$$P(A | B) = \frac{P(B | A)P(A)}{P(B)} \quad (3)$$

where:

- $P(A)$ is prior probability of event A ,
- $P(B)$ is prior probability of event B ,
- $P(B/A)$ is conditional probability of B given A ,
- $P(A/B)$ is conditional probability of A given B .

To illustrate this we return to the simple BN in Fig 4-1. In this model we estimate *programmer performance* based on the *training* and *documentation* quality.

Without entering any observations, based on the prior probabilities, the model predicts that the probability that the programmer will achieve good performance is

0.696. This is calculated as shown in Equation 4 (each combination of T and D multiplied by the appropriate conditional probability in Pe).

$$\begin{aligned}
P(Pe) &= P(T) * P(D) * P(Pe|T, D) + \\
&P(T) * P(\neg D) * P(Pe|T, \neg D) + \\
&P(\neg T) * P(D) * P(Pe|\neg T, D) + \\
&P(\neg T) * P(\neg D) * P(Pe|\neg T, \neg D) = \\
&0.8 * 0.7 * 0.9 + 0.8 * 0.3 * 0.6 + 0.2 * 0.7 * 0.3 + 0.2 * 0.3 * 0.1 = \\
&0.504 + 0.144 + 0.042 + 0.006 = 0.696.
\end{aligned} \tag{4}$$

Let us now assume that we know that the programmer receives good *documentation*. In such a case, the probability of achieving good performance increases to 0.78 (Equation 5). As we expected this is a higher probability than the initial one, because we are now sure that good *documentation* was received. Observation entered into *documentation* does not cause any change in the *training* node because they are conditionally independent given *programmer performance*.

$$\begin{aligned}
P(Pe | D) &= P(T) * P(Pe | T, D) + P(\neg T) * P(Pe | \neg T, D) = \\
&0.8 * 0.9 + 0.2 * 0.3 = 0.72 + 0.06 = 0.78.
\end{aligned} \tag{5}$$

Let us now further assume that although good *documentation* was received, a bad *programmer performance* has been observed. In such cases the model revises prediction of the *training* quality. By applying the Bayes's rule (Equation 3) the model predicts that the likelihood of good *training* was actually around 0.364 – much lower than initially assumed (0.8).

$$P(T | D, \neg Pe) = \frac{P(\neg Pe | T, D) * P(T)}{P(\neg Pe | D)} = \frac{0.1 * 0.8}{1 - 0.78} \approx 0.364 \tag{6}$$

In the more complex real-life BN models that we use, it is impossible to perform the Bayesian inference calculations manually. Fortunately, there are various efficient algorithms implemented in BN toolkits to do this. They can be either:

- exact, e.g.: variable elimination, clique tree propagation, recursive conditioning, enumeration; or
- approximate, e.g.: direct sampling, Markov chain sampling, variational methods, loopy propagation.

More on theoretical aspects of BNs can be found in [124, 177, 189, 220, 252].

Table 4-1 summarizes the most important advantages and disadvantages of BNs in modelling software project risks.

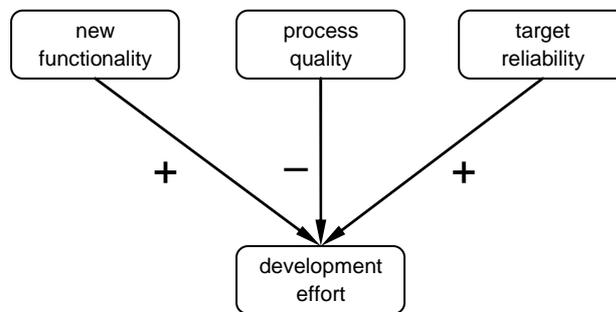
Table 4-1 Advantages and disadvantages of BNs

| Advantages | Disadvantages |
|---|---|
| <ul style="list-style-type: none"> • No two fixed separate lists of predictive and predictor variables. • Forward and backward inference. • Explicit incorporation of uncertainty. • Ability to run with missing data. • Ability to combine qualitative and quantitative data. | <ul style="list-style-type: none"> • Difficulty in obtaining reasonable prior knowledge. • Difficulty in empirically validating model estimates in models built only on expert knowledge. • Reduced accuracy in models calculated with approximate inference algorithm. • Calculation time. |

4.2 Trade-off analysis in a simple BN

As stated earlier, BNs appear to be the most promising method to analyze trade-offs in software projects. However, incorporating key project variables does not automatically ensure that relationships between them will be reflected correctly. It all depends on the structure of a particular model.

For example, we created a simple BN (Figure 4-2) with a structure derived from a parametric type model like COCOMO. It is intended to predict the *effort* required to complete a software project of given *functionality* with some constraints. ‘Plus’ and ‘minus’ signs indicate the positive or negative correlation between the linked variables.

**Figure 4-2 Parametric-type model converted to a BN**

The main problem with this type of model is that it is not a causal model. A proper causal model ought to predict increased *functionality* when *process quality* increases. As illustrated on Figure 4-3 we can see that this model incorrectly does not predict any change in *functionality* but predicts decreased *effort*. Now we enter an observation into development *effort* ('2242' – the median value predicted in the 'Baseline' scenario) and keep previously entered observation for *process quality*. The model again ought to predict an increased *functionality*. In Figure 4-4 we can observe that this time the model correctly predicts increased *functionality*. Hence we can

conclude that with this over-simplistic model we can perform a trade-off analysis when there is an observation entered to development *effort* but we cannot do such analysis when *effort* is left without observation. To reflect the trade-offs properly in more scenarios we need a BN with links reflecting causal relationships between these factors. To lay the foundations for such causal models we next need to consider a proper causal framework for risk.

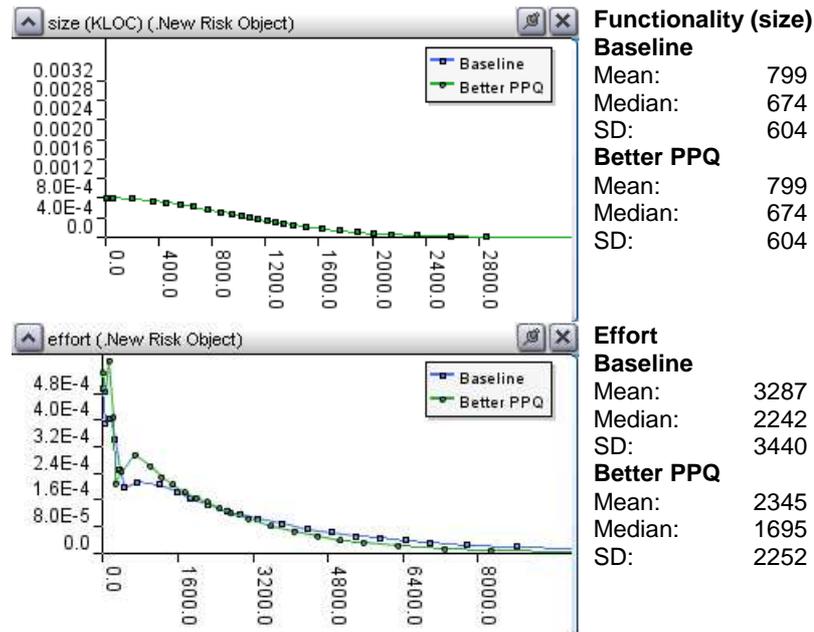


Figure 4-3 Predictions in the parametric-type model converted to a BN

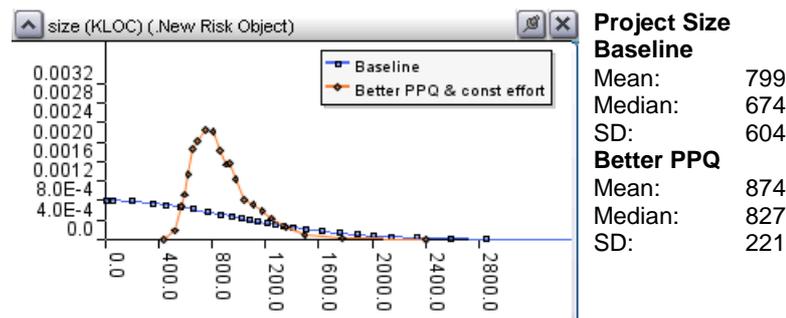


Figure 4-4 Predictions in the parametric-type model converted to a BN with effort constraint

4.3 Causal frameworks for risk

The simplest definitions of risk are:

- a source of danger [250];
- a possibility of loss or injury [164];

The first of these definitions highlights the qualitative sense of risk while the second one highlights the quantitative sense of risk. A quantitative approach often captures risk as defined by the risk score approach in Equation 7.

$$risk = probability\ of\ event * impact\ of\ event \quad (7)$$

This way of capturing risk has important disadvantages [70]:

- It is often both difficult and meaningless to estimate the probability of an event without analyzing various factors that determine the occurrence of the event and the possible actions undertaken to prevent the event from occurring.
- It is also often difficult and meaningless to estimate the impact of an event without analyzing various factors that can limit the negative consequences.
- The risk score can be used in prioritising risks but otherwise is meaningless.

BNs provide an ideal mechanism to overcome these limitations, providing a rational way of defining risk. In particular, the causal risk approach in [71] proposes 5 types of nodes:

- trigger – an event that may or may not occur; the stakeholder has no impact on this node,
- control – an event which can block or reduce a trigger's impact on the risk event,
- risk event – the key node observed in the model,
- mitigant – an event reducing impact of a risk event on further consequences,
- consequence – the result of a risk event occurring.

An example of this is shown in Fig 4-5. *Requirements creep* is a risk event which is triggered by *poor requirements specification* and controlled by *constrained user expectations*. *Requirements creep* can be mitigated by *additional funding from customer* but this may lead to *project over budget*. More examples on using this approach can be found in [70, 71].

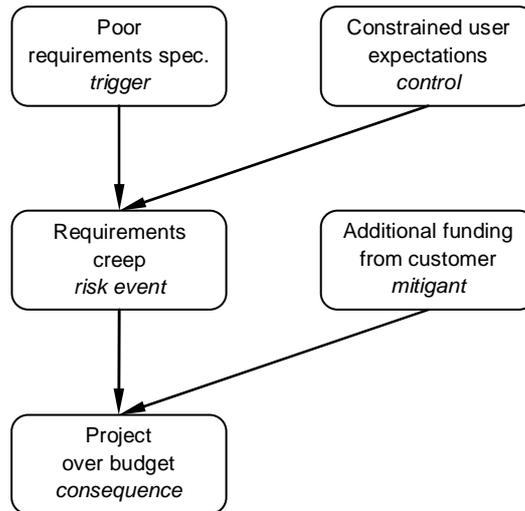


Figure 4-5 Causal taxonomy of risk [71]

We can extend the types of nodes when necessary. That might be especially useful for ‘consequences’. For example we may define three levels of consequences in models for business:

- internal consequence – the result of a risk event for the current company or process state,
- end-product consequence – the impact on end-product features,
- future company consequence – the impact on the company state in the future.

Distinguishing three types of consequences enables us to create BNs in which the nodes are joined in this order: trigger → risk event → internal consequence → end-product consequence → future company consequence. We may also skip some of these nodes. For example, if a particular risk event has no impact on internal and end-product consequences then we may have this order of nodes: trigger → risk event → future company consequence.

The advantages of using this approach are [70]:

- ensuring that every aspect of risk measurement is meaningful in the context,
- quantifying uncertainty (probability values associated with any node),
- providing visual and formal mechanism for recording and testing subjective probabilities,
- simplifying the creation of models from the beginning.

There is one weakness in this approach, which in some cases may prevent it from achieving its benefits. Simple and small models may be created for single users. However, such BNs are usually created for several different users. If they wish to see

the BN from different perspectives it may happen that for different users the model's nodes play different roles. The problem here is to assign the risk types to the nodes including their diverse user perspectives.

BN models built from this approach can also benefit from applying more layers of risk classification. Various types of risks should be used in software project risk assessment [240]:

- mission and goals,
- program management,
- decision drivers,
- organisation management,
- customer/user,
- project parameters,
- product content,
- deployment,
- development process,
- development environment,
- project management,
- project team,
- technology,
- maintenance.

The causal framework for risk described here has been highly influential in informing the structure of the various BN models reviewed and proposed in this thesis.

4.4 Naïve Bayesian Classifier

A Naïve Bayesian Classifier (NBC) is a special type of BN. It does not have a causal structure. Very often the links between variables are in the opposite direction to those we would expect when describing cause and effect. The typical structure of an NBC is illustrated in Figure 4-6. The dependent variable (the one to be estimated) is formally independent of the other nodes in the model. Only the prior unconditional probability distribution needs to be provided for this variable. The dependent variable is calculated using back-propagation – observations are entered into the leaf nodes

(predictors) and, by applying Bayes' theorem, the posterior probability for the dependent variable is updated.

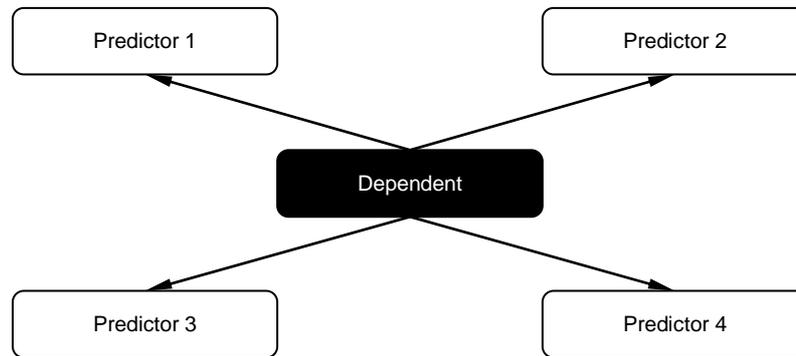


Figure 4-6 Typical structure of NBC

In a typical NBC, model predictors are assumed to be independent of each other as this simplifies the model structure and requires less data to build the model. Predictors' conditional probabilities are defined separately for each variable depending on the states of the variable to be classified. These probabilities also take into account the unconditional probabilities for each variable which reflects the frequency of each state of each predictor. Furthermore, NBC models “can work surprisingly well, even when the independence assumption is not true” [220 p. 482].

NBC models can be learned either with unsupervised or supervised training. In unsupervised training the model parameters are generated automatically from a dataset of observations from the past. Supervised training requires the participation of an expert in the domain to determine the outputs which the model should produce given a set of predictors.

NBC models are widely used, such as for example, in email spam classification software. We use this technique in creating:

- the PDR model estimating the productivity and defect rates which are the inputs to the main part of the Productivity Model (Chapter 8),
- the DTM model for estimating the proportions of different types of defects likely to be found after software testing (Chapter 9).

4.5 *Ranked nodes*

Existing models reviewed in this chapter and the new models discussed in chapters 6-10 extensively use the notion of *ranked* nodes [2, 74]. Ranked nodes are used to define qualitative variables which can be expressed on a ranked scale, that is where

the states can be ordered depending on the degree of intensity of a given feature. Most often we use either 5-point (from ‘very low’ to ‘very high’) or 7-point scale (from ‘lowest’ to ‘highest’).

Consider the simple example in Figure 4-7. We assume that all variables are ranked nodes measured on a 5-point scale. Normally, we would need to define 125 probability values in the CPT for *process effectiveness* (25 combinations of parents’ states for each state of *process effectiveness*). However, it turns out that, in most situations, using ranked nodes makes things much simpler.

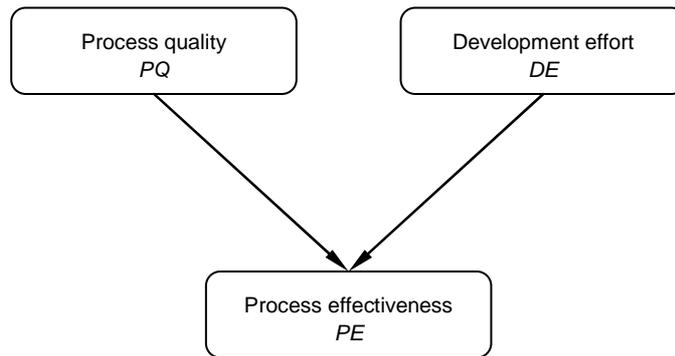


Figure 4-7 Simple example of using ranked nodes

To understand why, it is important to note that in a ranked node the whole range of possible values is internally defined as the interval $[0, 1]$. This range is divided in as many equal-width intervals as there are labelled states. Each interval is associated with one label. Users refer to node states by their labels (e.g. ‘very low’, ‘very high’), not by intervals.

For example, if a ranked node contains five states from ‘very low’ to ‘very high’, the following underlying intervals are automatically created:

- ‘very low’ – $[0, 0.2]$,
- ‘low’ – $[0.2, 0.4]$,
- ‘medium’ – $[0.4, 0.6]$,
- ‘high’ – $[0.6, 0.8]$,
- ‘very high’ – $[0.8, 1]$.

During the calculation process the ranked nodes are treated as continuous nodes – their intervals are used. This enables various arithmetic expressions, such as *sum*, *multiplication*, etc. and statistics such as mean, median, standard deviation etc. to be calculated. In particular, the ability to use various weighted expressions drastically simplifies the process of defining CPTs. For example, the CPT for *process effectiveness*

from the example above can be defined as the expression shown on Equation 8. This expression tells us that *process effectiveness* is a weighted mean of *process quality* and *development effort*, with the former having twice the impact than the latter.

$$PE = \text{wmean}(2, PQ, 1, DE) \quad (8)$$

TNormal distribution is frequently used in combination with the ranked nodes to capture the uncertainty of the relationship between the dependent variable and its predictors. TNormal distribution is similar to the Normal distribution but is defined over a fixed interval and has the following parameters:

- μ – the mean; for ranked nodes this is a weighted expression;
- σ^2 – the variance; for ranked nodes this reflects the uncertainty in the weighted expression;
- a and b – the start and endpoint of the range; in ranked nodes this range is always [0, 1].

In the CPT of the example we add the variance of 0.001:

$$PE = \text{TNormal}(\text{wmean}(2, PQ, 1, DE), 0.001, 0, 1) \quad (9)$$

The effect of these assumptions is shown for the following two scenarios in Figure 4-8:

- Scenario 1: *process quality* = ‘very high’ and *development effort* = ‘very low’,
- Scenario 2: *process quality* = ‘very low’ and *development effort* = ‘very high’.

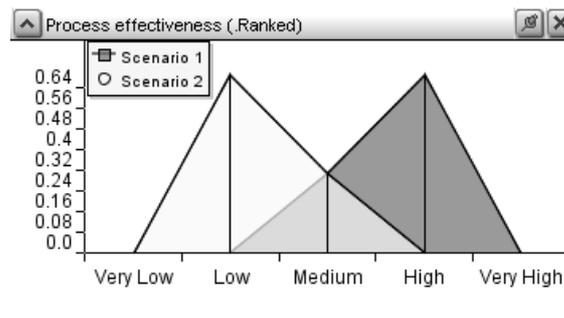


Figure 4-8 Predicted *process effectiveness* in an example model

To use these expressions it is sufficient to provide the weights w_i for each parent variable X_i without the need to manually analyze all combinations of states of parent variables. The following types of weighted expressions can be used in AgenaRisk:

- weighted mean,
- weighted min,

- weighted max,
- min-max mixture.

The *Weighted mean* function calculates an arithmetic average of a set of variables adjusted by the weights reflecting the importance of each variable used in the expression.

The *Weighted min* function assumes that when the values of variables are different, the aggregated value of them is lower than their average. The variation from the average depends on the weights assigned. When all weights are large this function produces a result close to the normal MIN function – selecting the lowest value from the set provided. When all weights are ‘1’ then the weighted min is a simple average. The *Weighted max* function works in a similar way to weighted min function, except that the higher, not lower, value is selected from the set provided.

The *min-max mixture* function is a weighted mixture of min and max functions. Here the weights are not assigned to individual variables but to the whole min and max functions.

4.6 Indicator and causal approach

Some model variables cannot be directly observed but can be estimated based on the values of variables that can be observed. For example, *overall staff quality* (unobserved) is an aggregation of variables like *staff motivation*, *staff turnover*, *relevant program language experience*, *general level of staff experience* and *general level of staff training*. Existing BN models contain various aggregated nodes, like *process quality*, *quality delivered*, *spec and doc process effectiveness*, *development process quality*, *testing process quality* and other. Such aggregations can be modelled using either indicators or causal relationships.

Indicators are the observable nodes, which in a BN model are defined as children of an aggregated node. Figure 4-9 provides an example of using indicator nodes in the MODIST Project-level Model. This model and other existing BN models are discussed in detail in Sections 4.8-4.11. It is quite difficult to assess the *overall staff quality* in isolation. Instead, the end user enters observations on the indicator nodes because they are easier to observe. However, such structure does not usually reflect the causal relationships – the links between the indicators and the aggregated node are often in the opposite direction to the true causal relationships.

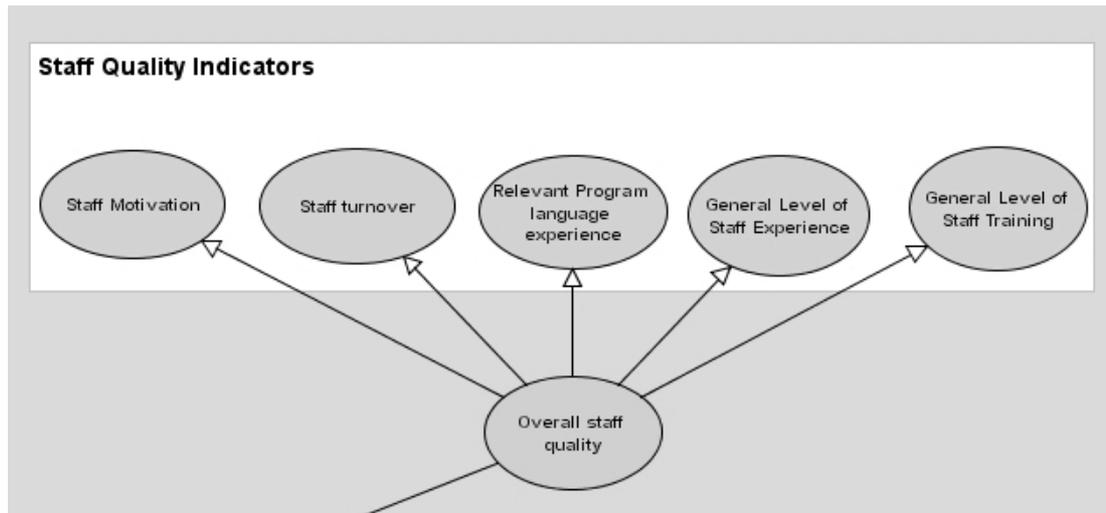


Figure 4-9 Example of using indicator approach in MODIST Project-level Model

A feature of this approach is that entering an observation into one of the indicator nodes leads to changes in the distributions of the other indicator nodes. If we enter the observation 'very high' to the *relevant program language experience* we notice a shift in the distributions of the other indicator nodes. We then might ask the question: what happens in reality if we believe that *relevant program language experience* is very high? Can we conclude that the staff is also more highly motivated or that the staff turnover will be lower than normal as the model suggests (Figure 4-10)? Indeed, there are relationships between some of these process factors. For example, Hall et al. found that there is a correlation between *staff motivation* and *staff turnover*: “staff who stay in their job are likely to be motivated” [98].

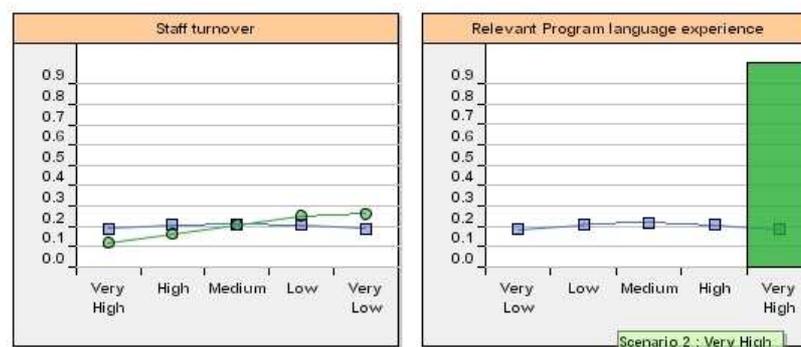


Figure 4-10 Influence of one indicator node on another one in MODIST Project-level Model

What if, in reality, some of the indicators are conditionally independent? We then use a second idea – the ‘causal approach’. Here the links are in the opposite direction than in the indicator approach – they are pointing to the single aggregated node such as *testing process quality* illustrated on Figure 4-11. In this case, the causal nodes are truly

independent. If we enter an observation into one of those detailed nodes it will affect only the aggregated node. Other causal nodes will remain unchanged.

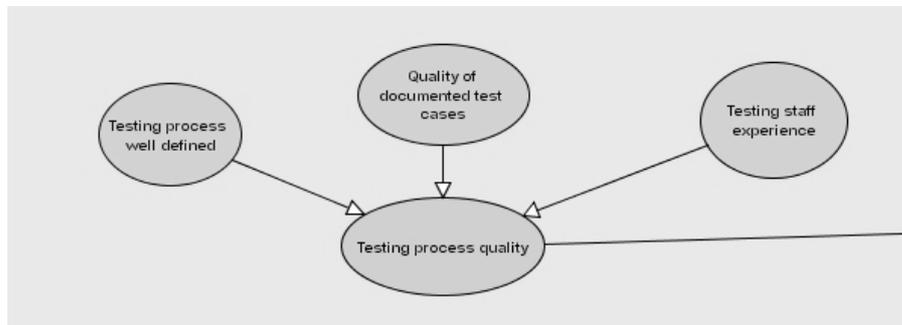


Figure 4-11 Causal approach example in Revised Defect Prediction Model

A traditional weakness in using this causal approach is the complexity of the NPT in the aggregated node due to the large number of combinations of states of parent nodes. If all the nodes can be defined as ranked nodes and if the causal relationship can be approximated by a weighted expression as defined in the previous section, then this traditional weakness can be overcome as discussed there. Alternatively, we can create ‘dummy’ nodes, which are intermediate hidden nodes grouping two or three parent nodes.

Examples illustrating the above different approaches for estimating *testing process quality* are presented in Figure 4-11 (indicator approach from the MODIST Phase-based Defect Prediction Model) and Figure 4-12 (causal approach from the Revised Defect Prediction Model).

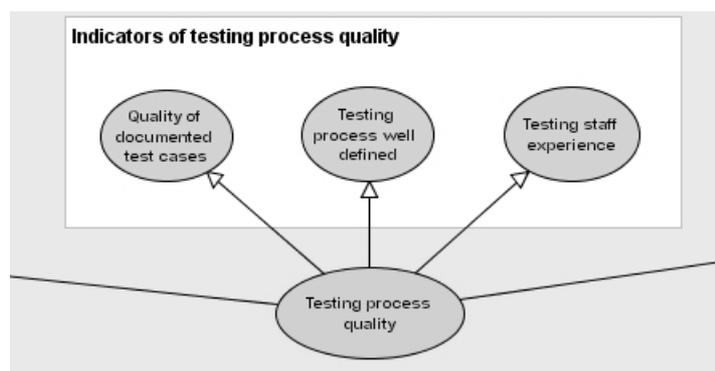


Figure 4-12 Example of indicator approach MODIST Phase-based Defect Prediction Model

Table 4-2 illustrates the advantages and disadvantages of implementing the different approaches to solve modelling of complex nodes in BNs.

Table 4-2 Summary of pros and cons of implementing different approaches [207]

| Approach | Pros | Cons |
|--------------------|--|--|
| Indicator approach | <ul style="list-style-type: none"> • fast calculation since no node has more than one parent • simple to add or remove indicators – since other indicators and their NPTs are unaffected | <ul style="list-style-type: none"> • may not reflect true causal relationship • possible undesired dependency between indicator nodes • need to define an NPT for each indicator node |
| Causal approach | <ul style="list-style-type: none"> • nodes connected more ‘naturally’ • only one NPT had to be defined | <ul style="list-style-type: none"> • potentially complex NPT if ranked nodes/weighted function cannot be used • slower calculation because of large NPT • adding/removing causal nodes requires change to the whole NPT |

To more precisely reflect relationships from the real world it is possible to combine the causal and indicator approaches, especially when more observable nodes are used than in the examples discussed so far. Those variables which in reality influence each other should be modelled using the indicator approach, while those which are independent should use the causal approach.

If we want to introduce only a small degree of dependency between observable variables we may add those influencing factors as ‘common causes’ (Figure 4-13) used in defect prediction models. Normally ‘specification and documentation’ and ‘design and development’ processes are independent in the sense that they do not influence each other directly. However, *overall management effectiveness* influences both of them.

A fair conclusion from this analysis of the applicability of the indicator and causal approaches is that neither of them is the best in every situation. It is not even possible to say which approach makes it easiest to create a model as this depends on the aims of the model, the availability of the data and the availability of domain knowledge. In fact we have used both of these approaches in the newly created models described later:

- The main part of the Productivity Model together with the Process and People Subnet use the causal approach and ‘common causes’ (Chapter 6),
- The model for predicting prior productivity and defect rates (the PDR Model) and the Defect Types Model both use the indicator approach (Chapters 8 and 9).

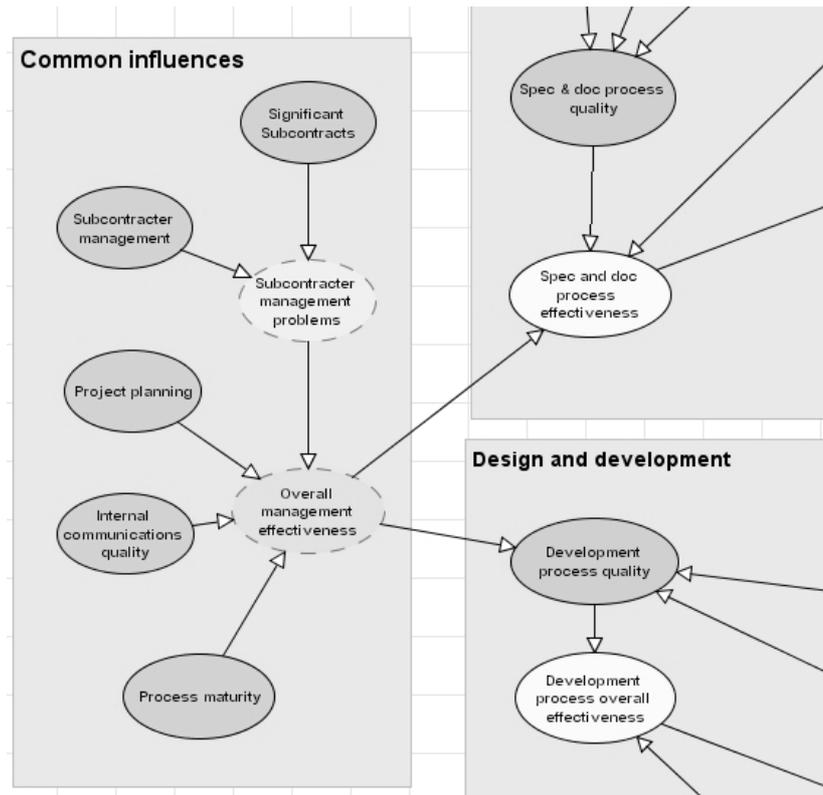


Figure 4-13 Common causes in Revised Defect Prediction Model

4.7 Reasoning over time

Apart from performing estimates in a single time slice, BNs allow us to build models enabling reasoning over time. Such models predict the future states of variables given the observed states of these variables in the past, usually with other explanatory variables also observed in the past. Figure 4-14 illustrates the structure of a simple temporal model. By entering observations for variables Y_i , the states of variables X_i are updated and used in predicting their future states.

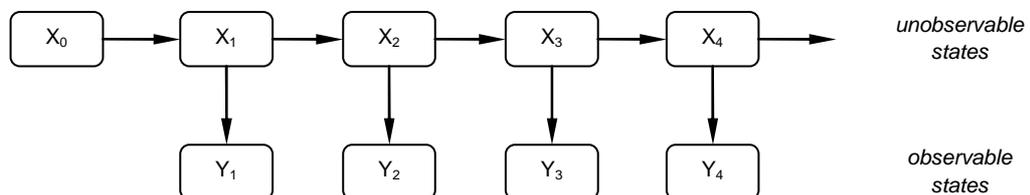


Figure 4-14 Structure of a simple temporal model

There are three main families of such temporal models [220]:

1. *Hidden Markov Models* (HMMs) – where unknown states of discrete variables are learnt by observing discrete or continuous variables,

2. *Kalman Filters* (KFs) – where unknown states of continuous variables are learnt by observing other continuous variables,
3. *Dynamic Bayesian Nets* (DBNs) – which extend both HMMs and KFs by allowing both discrete and continuous variables, more complex dependencies between variables, non-linear relationships, non-Gaussian distributions and other extensions.

Such models are based on two main assumptions that reduce the model complexity:

1. Stationery process – all changes are caused by a stationery process which “is governed by laws that do not themselves change over time” [220 p. 539]. This solves a problem with a potentially unbounded number of CPTs – different for each X_i variable in each time slice.
2. Markov assumption – “the current state depends only on a finite history of previous states” [220 p. 539]. This solves a problem with a potentially infinite number of parents for a variable whose state is changed during a process. The simplest and the most frequently used case is a first-order Markov process. Here the current state X_i depends only on the previous state X_{i-1} and does not depend on any earlier states.

More on the theoretical aspects of temporal models can be found in [156, 157, 177, 220].

We analyze the applicability of DBNs to predict the number of defects likely to be found and fixed in future testing iterations given the observed number of defects found and fixed in past testing iterations (Chapter 10).

4.8 MODIST Project-Level BN

Model structure

The aim for the Project-level BN Model was to predict and assess the overall risk/quality status of a large software project. The target user group of the project level model is project managers.

Figure 4-15 illustrates the schematic for the Project-level Model. There are six subnets in the model [4, 67]:

- distributed communication and management – with variables capturing the nature and scale of the distributed aspects of the project and the extent to which

these are well managed (e.g. *scale of distributed communications, communication management quality, scale of subcontractors, subcontract management quality, internal management quality, overall management quality*);

- requirements and specification – with variables relating to the extent to which the project is likely to produce accurate and clear requirements and specifications (e.g. *requirements novelty, requirements complexity, stakeholder involvement, requirements stability, specification accuracy*);
- process quality – variables relating to the quality of the development processes used in the project (e.g. *specification process quality, specification clarity, development and testing quality, CMM level, overall process quality*);
- people quality – variables relating to the quality of people working on the project (e.g. *staff motivation, staff turnover, general level of staff experience, overall staff quality, overall people quality*);
- functionality delivered – variables relating to the amount of the new functionality delivered on the project, including the resources assigned to the project (e.g. *new functionality delivered, KLOC delivered, project duration, average number of people full time*);
- quality delivered – variables relating to both the final quality of the system delivered and extent to which it provides user satisfaction (e.g. *defects per KLOC post release, quality delivered, user satisfaction*).

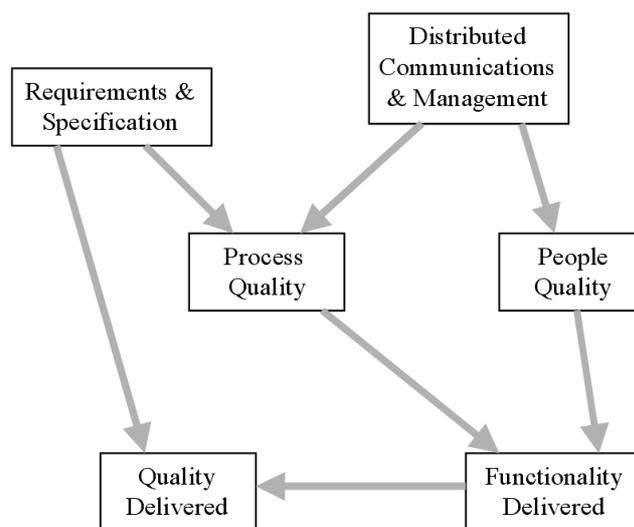


Figure 4-15 Schematic of the Project-level Model [4]

The key part of the model (Figure 4-16) is a component for analysing project trade-offs between:

- quality – represented by objective *quality delivered* and subjective *user satisfaction*,
- effort – represented by the *average number of people full time* working for the project,
- time – represented by the *project duration*,
- functionality – meaning *functionality delivered*.

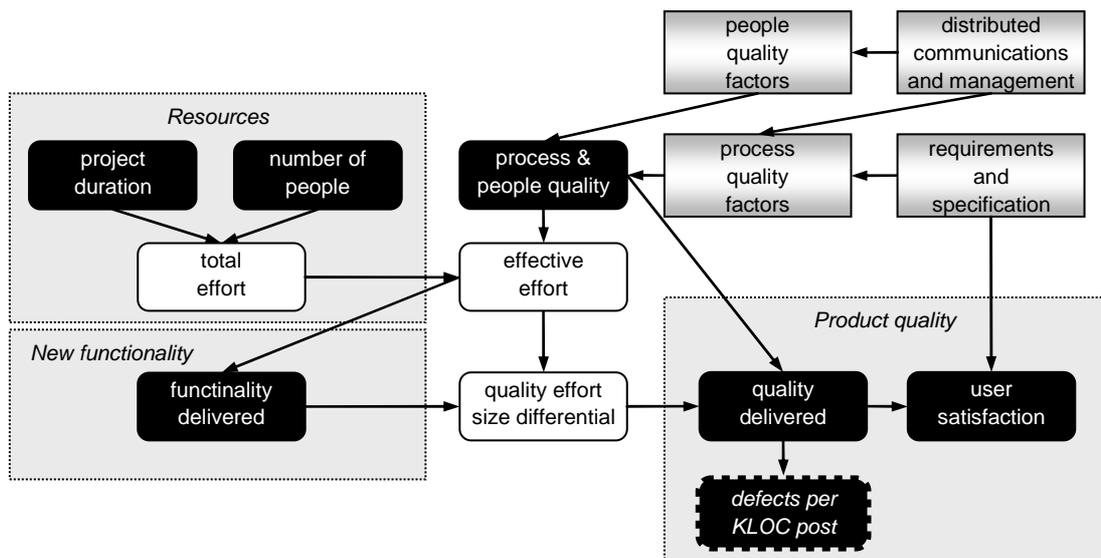


Figure 4-16 Structure of the key part of the MODIST Project-level Model

Using the model for prediction and trade-off analysis

Suppose we want to develop a system containing 7000 function points (FPs). The model will show us the predicted distributions for the effort and time necessary to develop the software (Figure 4-17– ‘baseline’ scenario). If we add a constraint that the quality delivered must be perfect we can observe that we will need more people for the project and that it will last longer than without such a constraint (Figure 4-17 – ‘fixed quality’ scenario). The model also suggests that we can probably only achieve the target if we have high *process and people quality*.

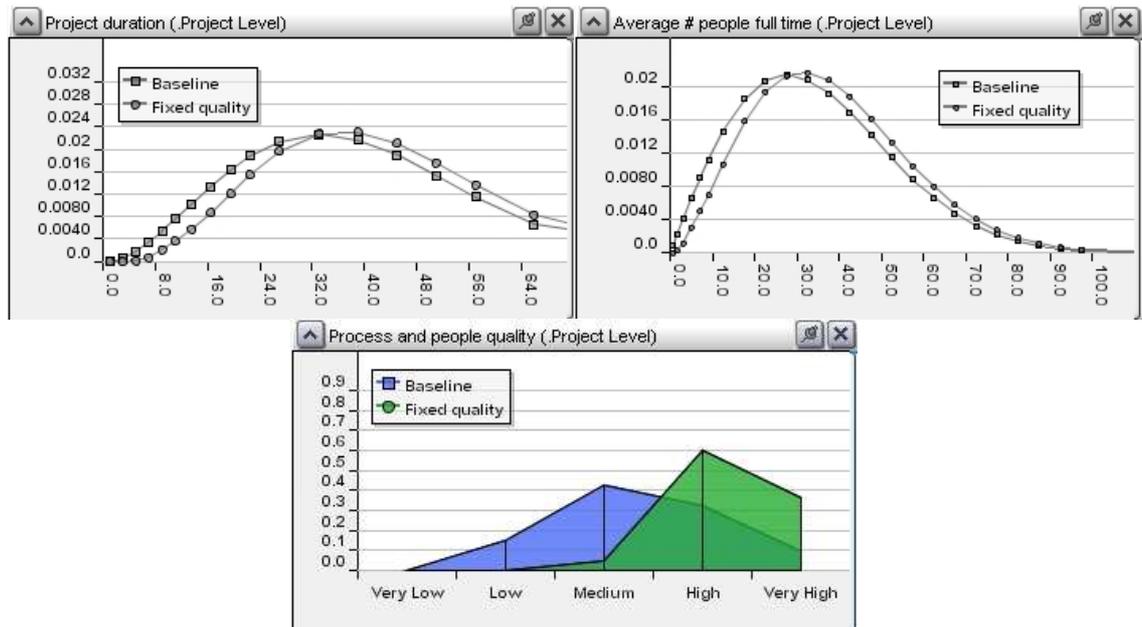


Figure 4-17 Predicted distributions after setting ‘perfect’ quality for software

It may happen that we do not have the amount of time for development that the model predicts. Suppose we have only 24 months for software delivery (Figure 4-18 – ‘fixed quality and schedule’ scenario). In this case we will only succeed if we significantly increase the number of people working on the project. We also need a very high level for *process and people quality*.

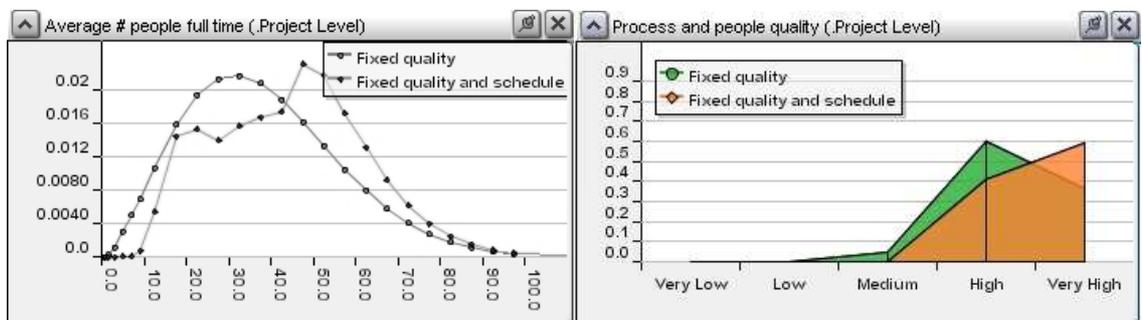


Figure 4-18 Predicted distributions after setting fixed *project duration*

If we can dispense with the perfect quality, because it may be difficult to achieve, we would not need so many people. The *process and people quality* do not need to be as high (Figure 4-19 – ‘fixed schedule’). However, those resources and *process and people quality* still need to be higher than without the project duration constraint.

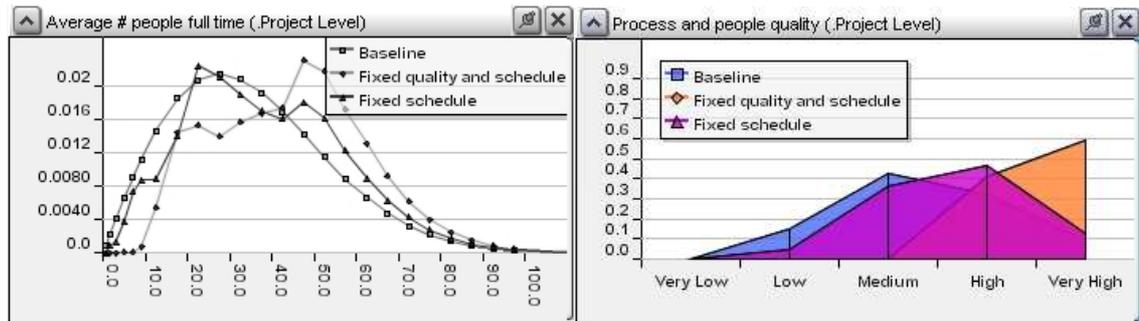


Figure 4-19 Predicted distributions after resigning from ‘perfect’ software quality

The MODIST Project-level Model appears to be the first BN that enables trade-off analysis. However, it was not built with the intention of reflecting all possible relationships between project factors. For example, in the following scenarios the model provides unexpected results:

- Normally increased *effort* should lead to an increase in either *functionality* or *quality delivered* or both. The model correctly predicts the increase in *functionality* but predicts a very small decrease in the *quality delivered*. The underlying logic is that nearly all additional *effort* will be allocated in developing more *functionality*. However, delivering more *functionality* also requires spending more *effort* on the quality assurance and testing. Since the model does not assume the allocation of enough additional *effort* on quality assurance and testing, it predicts that the *quality delivered* will be slightly lower compared with the scenario with the initial level of *effort*.
- We enter the single observation of an increase in *quality delivered* in the model. The model correctly predicts that this target can be achieved with a higher *process quality*. But, unexpectedly, the model does not predict an increase in the total *effort*. It predicts that the only way to achieve higher *quality delivered* with *effort* unchanged is to reduce the *functionality*. This means that less *effort* will be spent on building a product but more on the quality assurance and testing.

4.9 MODIST Phase-based Defect Prediction Model

Model structure

The Phase-based Model enables detailed defect prediction to be performed at a lower level of granularity, not for the whole project but for individual teams. Joining together copies of the phase-based model enables us to form complex multi-phase

models that reflect any particular lifecycle used. It can be classic waterfall, spiral, incremental, or even a custom lifecycle developed in a particular company.

A single-phase model is responsible for predictions or assessments in a single software development phase. Any phase in the model is a single activity or a combination of the following activities [4]:

- Specification/documentation – The aim is to understand and/or describe existing or proposed functionality. It includes gathering requirements and preparation of documentation. The result of this activity may be specification documents, design documents and user manuals.
- Development (coding) – The result of this activity is executable code which was created based on some pre-defined requirements.
- Testing and rework – This activity deals with activities involving executing code to find and note defects and then to fix known defects.

Figure 4-20 illustrates the schematic view of the whole phase net. The rectangles in the figure indicate that they are not a single nodes but more complex subnets. There are five main parts in the model:

1. specification and documentation – contains variables related to:
 - specification process quality (e.g. *specification process quality* and its indicators),
 - scale of new specification and documentation work (including *quality of overall system specification and documentation PRE* and *quality of overall system specification and documentation POST*),
 - analyzing if specification process quality (adjusted by *specification and documentation effort*) is enough for the planned scale of new specification and documentation work (*adequacy of doc for new functionality* and its indicators);
2. design and development – contains variables related to coding (e.g. *development process quality* and its indicators, *development process effort*, *development process effectiveness*);
3. testing and rework – contains variables related to testing activities (*testing process quality* and its indicators, *testing process effort*, *testing process overall effectiveness*) and rework – fixing known defects (e.g. *rework process quality*, *rework effort*, *rework process overall effectiveness*);

4. scale of new functionality implemented – contains variables describing the size and complexity of software which needs to be implemented in this phase (*new functionality implemented this phase* and its indicators);
5. defect insertion and recovery – contains variables related to defect prediction based on the effectiveness of different development activities and the scale of new functionality (e.g. *total potential defects*, *new defects in*, *total defects in*, *defects found*, *defects fixed*); the subnet takes into account the amount of residual defects predicted for the previous phases.

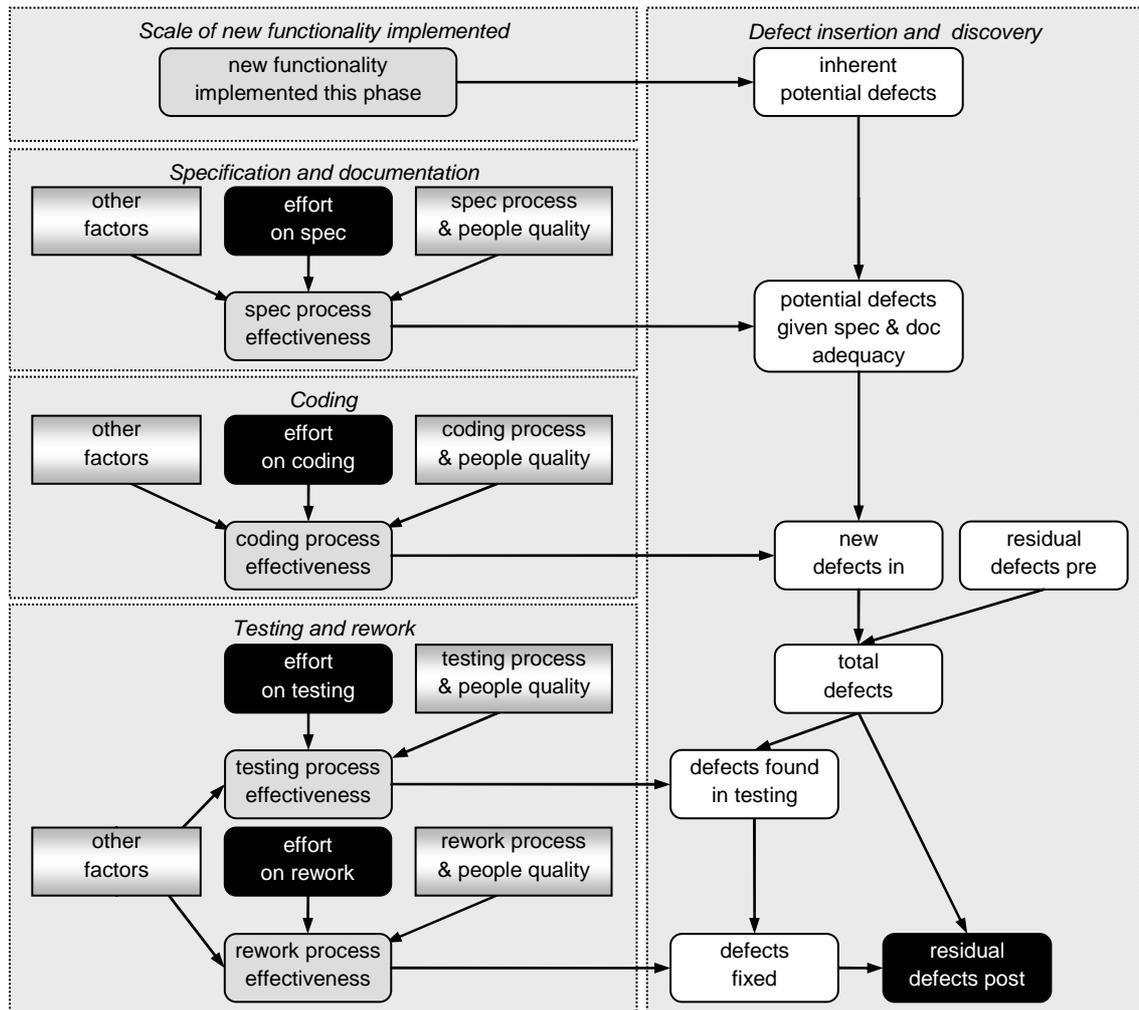


Figure 4-20 Schematic view of the whole phase net in Phase-based Defect Prediction Model

Model predictions

We can use the Phase-based Model to predict the amount of defects depending on the quality/effort of development activities. For example, suppose we need to create a part of a system of 1000 FPs. We can observe the predicted number of *defects found in*

testing and *residual defects*. We will analyze how these predictions change when we enter more data about development activities.

Suppose we want to analyze the impact of testing and rework effort on defect prediction. Figure 4-21 illustrates this case. When testing and rework effort is ‘very high’ we can observe that we should expect many more defects to be found in testing compared to ‘very low’ testing and rework effort. But that will change the prediction for *residual defects* remaining after testing and rework. The model predicts that when testing and rework effort is ‘very low’ we should expect to have more *residual defects* (median=2946) compared to ‘very high’ testing and rework effort (*residual defects*: median=1065).

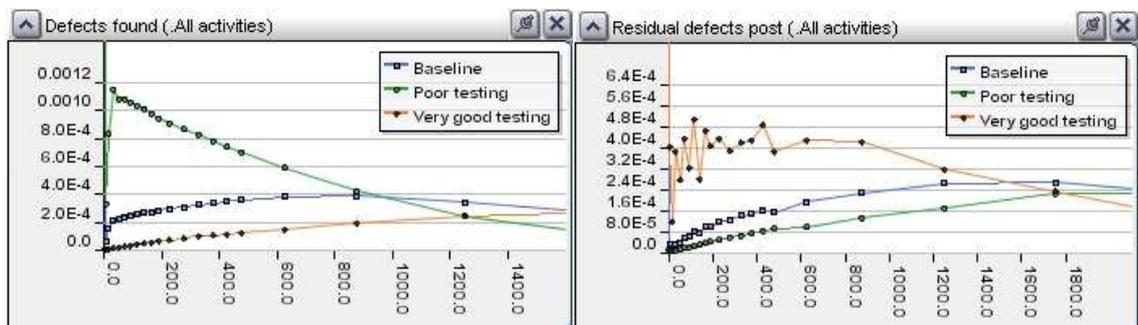


Figure 4-21 Predictions for different *testing effort* scenarios

We can observe that even if we have ‘very low’ testing and rework effort we can still improve the software quality. If we have ‘very high’ effort in development and very good quality of development process we can still succeed in improving software quality. We should then expect a lower number of *defects found* in testing (Figure 4-22). That is simply because there are fewer defects in the software. Even poor testing and rework will not affect the predicted defects that much. As a consequence we should also expect less *residual defects*. It will be even lower than if we set testing and rework effort to ‘very high’ without specifying *development effort* and *development process quality*.

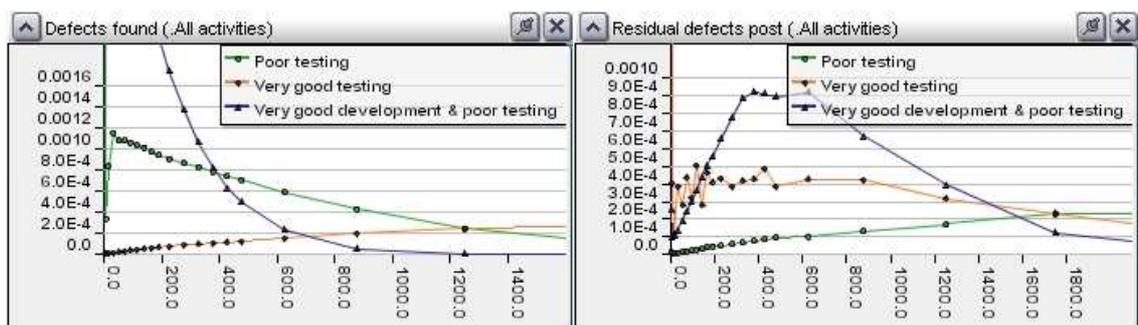


Figure 4-22 Impact of very good development on defect prediction

Reflecting various development lifecycles

The Phase-based model is flexible in that it can be used in different types of phases. For example, we may have a phase whose aim is to add some more code, and test the software – without the need to produce any type of documentation. In this case we could set both *specification and documentation effort* and *scale of new specification and documentation work in this phase* to zero. But we then have an unnecessarily complex net – with unneeded subnets. This is the motivation for creating seven models containing seven possible combinations of activities in a single phase:

- all activities (specification/documentation, development, testing/rework);
- specification/documentation and development, without testing;
- specification/documentation and testing, without development;
- development and testing, without specification/documentation;
- only specification/documentation;
- only development;
- only testing.

We can join several Phase-based models to reflect the development process of a large part of a software development or the whole project. This is possible because there are:

- two nodes for the quality of system specification/documentation:
 - *quality of overall system specification and documentation pre* – reflects all documentation available before the current phase,
 - *quality of overall system specification and documentation post* – reflects all documentation available after the current phase,
- two nodes for residual defects prediction:
 - *residual defects pre* – reflects amount of residual defects in existing code (before the current phase),
 - *residual defects post* – reflects amount of residual defects remaining in code after the current phase.

Connecting those pairs of nodes in successive phases we ensure that documentation created in one phase (*post*) will be the base (*pre*) for the second phase. Similarly residual defects left in one phase (*post*) will be the base for the next phase (*pre*). For example, we may create a sequence of joined models in an integrated model which reflects the waterfall lifecycle (Figure 4-23).

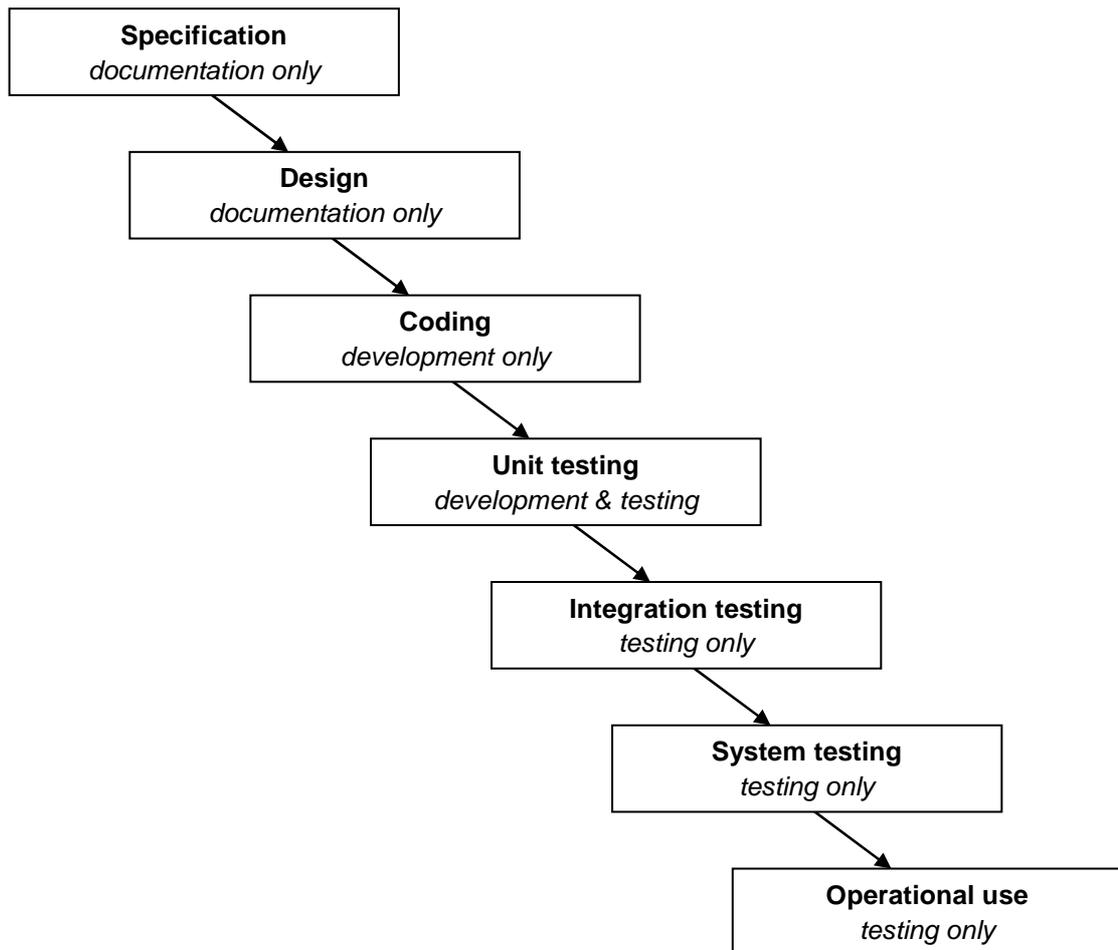


Figure 4-23 Example of sequence of joined models (based on [4])

It is also possible to join Phase-based Models in different ways, for example:

- using separate models for specification, design, development and testing for each software module, joined together for phases such as integration work and system testing;
- joining several ‘all activities’ models to reflect spiral (incremental) lifecycle where each ‘basic’ model indicates a single increment;
- joining several models to reflect development process with milestones or reviews defined, for example, every 3 months where each phase reflects a single development period.

Compatibility with causal risk approach

Although no part of the MODIST models was built explicitly using the causal risk approach discussed in Section 4.3 it turns out that key parts in the model seem to be compatible with this concept. For example, the coding team may be interested in the number of defects inserted in a coding process, seen as a risk event (Figure 4-24). This

number is influenced by the number of *total potential defects* (capturing project size and adequacy of specification process). From a programmers' point of view this is a trigger initiating their work but is out of their control. They can control the extent to which *total potential defects* impacts on defects inserted through allocating appropriate (or not) *coding process quality* and *coding effort*. In the case of inadequate *coding process and effort* there is a high risk of an increased number of defects inserted by the programmers (*defects in*). Testing and rework activities then follow to mitigate this and to ensure that fewer *residual defects* are left after product release.

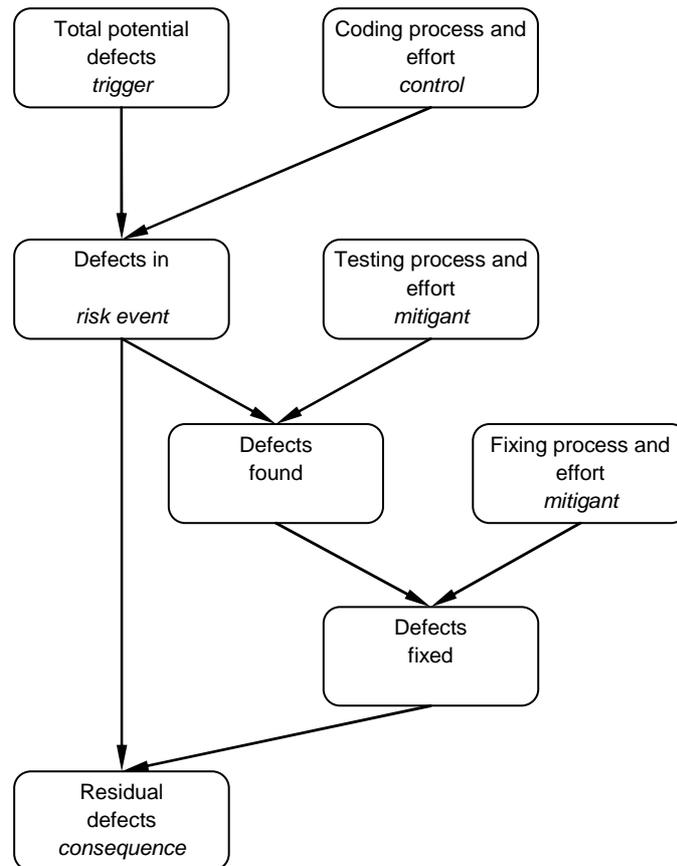


Figure 4-24 Risk framework in the MODIST Phase-based Defect Prediction Model

4.10 Revised Defect Prediction Model

Model structure

The main aim of this model, as its name says, is to predict defects in developed software. This model is based on the MODIST Phase-based Model with some changes introduced. Figure 4-25 illustrates the schematic of this model.

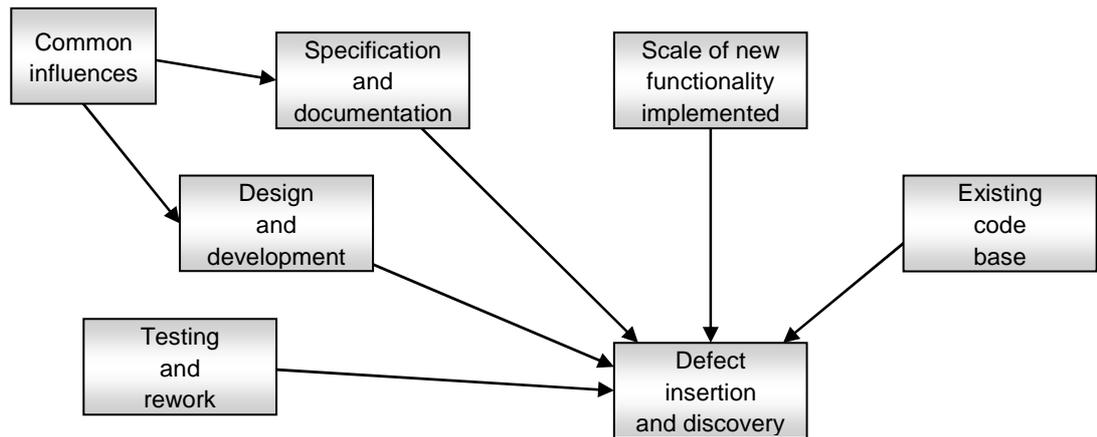


Figure 4-25 Schematic view of Revised Defect Prediction Model [69]

The users of the Phase-based Defect Prediction Model wished to have a single model for predicting defects in the whole project. They wanted to use this model in a similar way to the Project-level Model, that is without the need to create a complex multi-phase model.

As was the case in the Phase-based Model, the general idea of splitting the development activities into three groups remains. In this Revised Defect Prediction Model there are also subnets related to:

- specification and documentation,
- design and development,
- testing and rework.

The main improvements in the Revised Defect Prediction Model are:

- Simplification of specification and documentation subnet – causes and effects are more intuitive here (Figure 4-26).
- A different scale is used in the new functionality implemented subnet – the model uses the *effective KLOC implemented this phase* variable instead of function points for software measurement. It depends on real software size (KLOC) which is adjusted by *complexity of new functionality*, *scale of distributed communication* and *integration with third-party software*. So it tells us how complex is this portion of code (Figure 4-27).
- It uses a ‘causal approach’ for defining complex nodes (aggregations of several qualitative factors) rather than the ‘indicator approach’ used in both MODIST models (details in Section 4.6).
- Existing code base – a new subnet containing variables describing any existing portion of the code (e.g. *complexity of existing code base*, *KLOC existing code*

base, overall process and testing quality of existing code base). They influence total defects in via residual defects pre (Figure 4-28).

- Common influences – a new subnet containing factors influencing specification/documentation and development activities which reflect overall management effectiveness, for example: *subcontractor management, project planning, internal communications quality, process maturity* (Figure 4-29).

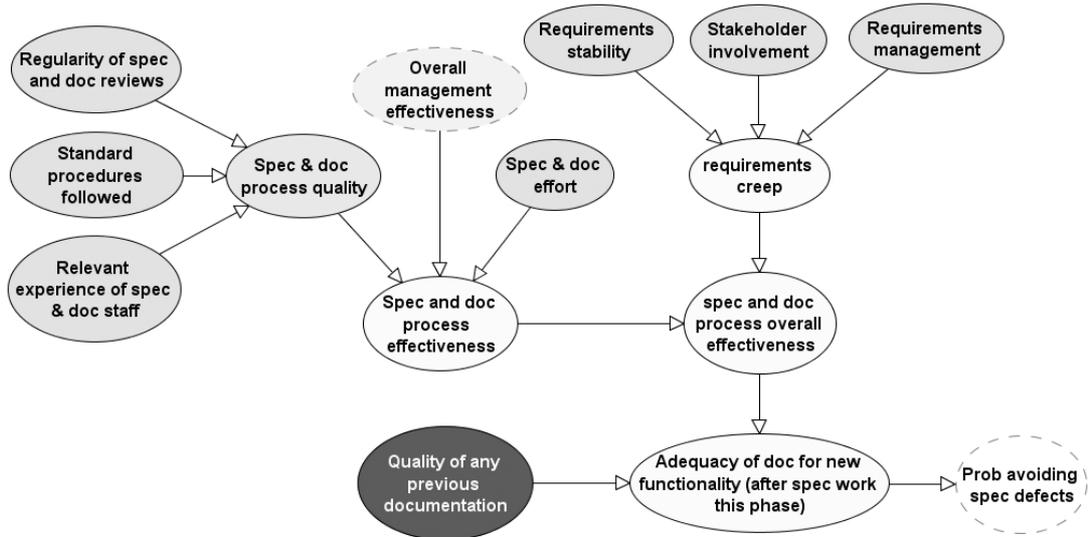


Figure 4-26 Specification and documentation subnet in Revised Defect Prediction Model

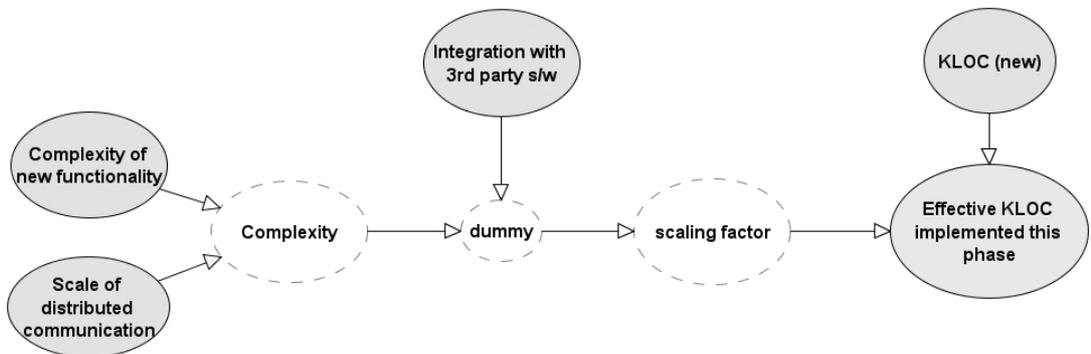


Figure 4-27 New functionality subnet in Revised Defect Prediction Model

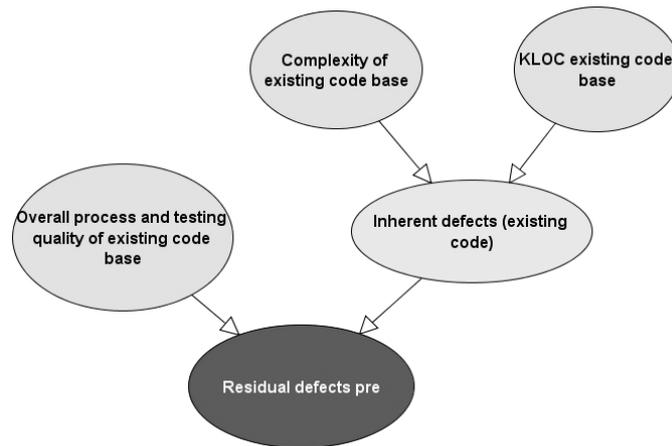


Figure 4-28 Existing code base subnet in Revised Defect Prediction Model

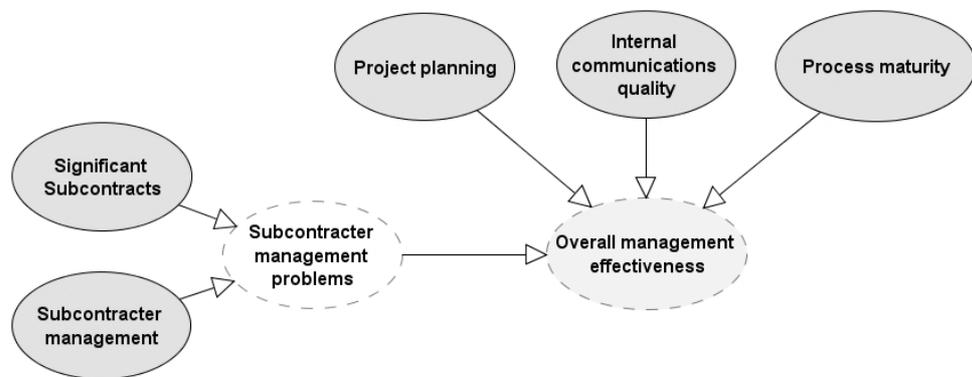


Figure 4-29 Common influences subnet in Revised Defect Prediction Model

Model predictions

Since the Revised Defect Prediction Model is an improved version of the MODIST Phase-based Model, we can run similar predictions. But we are also able to analyze factors which are not present in the Phase-based Model. For example, suppose we have to develop a very complex portion of a system which we estimate as 150 KLOC. We wish to add new code to the existing code of the same size but of ‘low’ complexity. The model predicts a difference for *residual defects* depending on whether the existing code was badly or well tested. We should expect to find more defects in testing when *overall process and testing quality of existing code* was ‘very low’ (Figure 4-30).

Suppose that we actually know that existing code was well tested. But we also know that we need to cooperate with a subcontractor who has poor management. In addition we now have to integrate with third party software. We can observe that we should now expect more *defects found in testing* and that we leave more *residual defects* than without such changes (Figure 4-31).

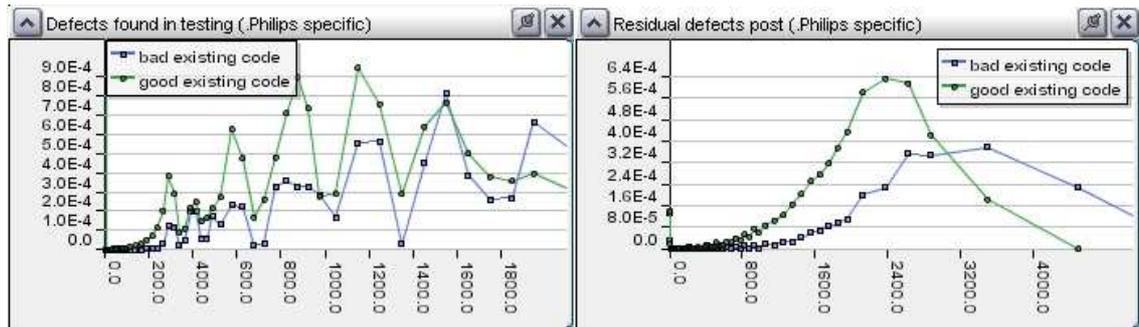


Figure 4-30 Influence of quality of existing code on defect prediction in the current phase

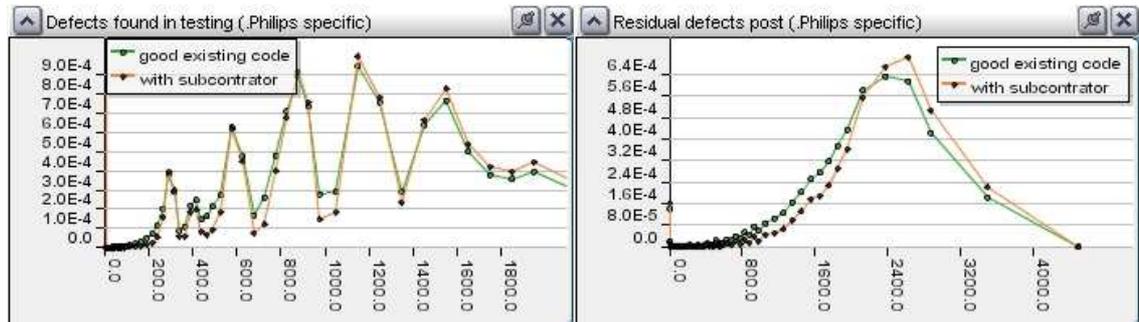


Figure 4-31 Predicted distributions after adding poor subcontractor and necessary integration with third party software

Trade-off analysis

Neither defect prediction model was developed with the main aim of enabling trade-off analysis between the key project variables. As their names say, they were intended to predict the number of defects in developed software of known size. These models originally did not contain the variable *defect rate* but they both contain *product size* and *number of defects*. We added a new variable *defect rate* reflecting delivered software quality (Equation 10).

$$\text{defect rate} = \frac{\text{number of defects}}{\text{functionality}} \quad (10)$$

The problem of analysing trade-off is related to the way that both defect prediction models capture *effort*. In both models *effort* does not reflect the actual *effort* on any absolute scale. Rather, it reflects the appropriateness of effort for a project of a given size, developed in a particular environment. So, for example, increased *functionality* normally requires more *effort*. This cannot be reflected in the model because of the definition used for *effort* in these models. In this case the model does not predict any change in *effort*. Because of this interpretation most examples of trade-off analysis are not relevant in these models.

Furthermore, these models are not fully causal. For example, in these models *process quality* does not impact *functionality*. Therefore, the model predicts that the increase in *process quality* does not cause any change in *functionality*.

However, the model provides correct prediction in some scenarios. For example, let us assume that with fixed *process quality* more *functionality* has to be delivered. In this case the model correctly predicts that to maintain *quality delivered* at the initial level more *effort* is required. Another example assumes that a fixed *functionality* needs to be delivered using fixed resources but with a higher target for *quality delivered*. In this case the model predicts that achieving the target for *quality delivered* is possible with a higher level of *process quality*.

Sensitivity analysis

For the Revised Defect Prediction Model we performed a sensitivity analysis with the aim of determining which variables in the model have the strongest influence on *residual defects post*. We applied two types of analyses:

- basic sensitivity analysis – where we analyze the impact on *residual defects post* of one or two qualitative model input variables at a time in different combinations .
- global sensitivity analysis – where the sensitivity estimates of specific factors are evaluated incorporating changes in all other factors and which also incorporate the probability distributions associated with the variables [221, 248].

Figure 4-32 illustrates the impact of qualitative model variables on residual defects post for a project of 10 KLOC. Clearly, there are three factors significantly more important than the others:

- *complexity of new functionality*,
- *scale of distributed communication*,
- *integration with 3rd party software* – the actual variation in number of defects caused by change in this variable ranged from 132 to 529 (the high boundary is truncated on the graph to improve its overall clarity).

It is also worth noting the influence of the last two factors in Figure 4-32, namely *complexity of existing code base* and *overall process and testing quality of existing code base*. It turns out that the influence of these factors increased with the increase in the proportion of size of reused code to the size of the new code. So, not surprisingly, the

model suggests that the more code (proportionally) we reuse the more important it becomes to determine how this existing code was developed.

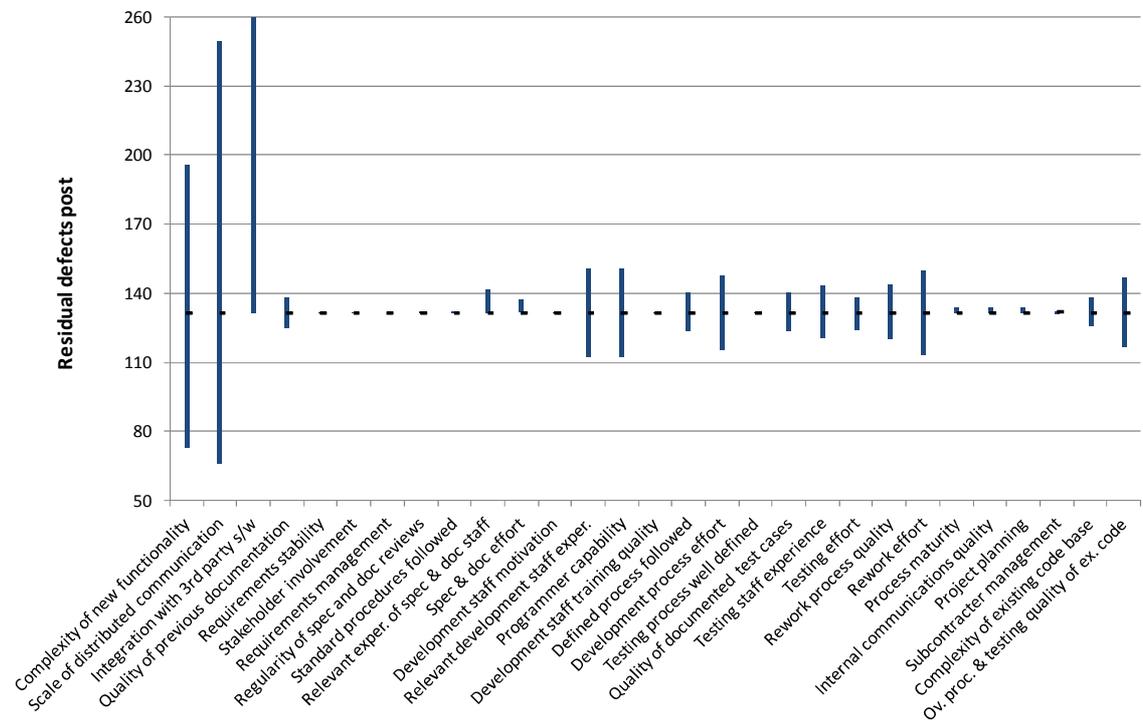


Figure 4-32 Impact of particular qualitative factors on *residual defects post*

Our analysis indicated that the variables covering the requirements process had little impact on the number of defects. Yet, poor requirements processes are often the first focus of any improvement activities. Given the relatively informal requirements processes of the projects being studied, *scale of distributed communication* acted to a certain extent as a proxy for the requirements process.

Tests performed for other project sizes confirmed similar proportions for the impact of qualitative factors. However, as the project size increases the impact of the 3 most important factors decrease in relation to other factors. Such behaviour is caused by the fact that the model has an upper bound for project size adjusted by these three most important factors which adjust the *effective KLoC implemented this phase*. This limit was introduced due to the lack of empirical data for yet bigger projects.

Table 4-3 illustrates the results of the global sensitivity analysis. Based on these results we can identify six of the most influential factors on the number of residual defects:

- *KLoC (new)*;
- *scale of distributed communication*;

- *complexity of new functionality;*
- *KLOC existing code base;*
- *testing staff experience;*
- *rework effort.*

Table 4-3 Results of global sensitivity analysis

| Sensitivity measure \ Input variable | Spearman's rank correlation coefficient SPEA | Standardised rank correlation coefficient SRCC | Partial rank correlation coefficient PRCC |
|---|--|--|---|
| KLOC (new) | 0.670* <i>1</i> | 0.663* <i>1</i> | 0.853 <i>1</i> |
| Complexity of new functionality | 0.353* <i>3</i> | 0.339* <i>3</i> | 0.640 <i>3</i> |
| Scale of distributed communication | 0.454* <i>2</i> | 0.455* <i>2</i> | 0.746 <i>2</i> |
| Integration with 3rd party s/w | 0.041* <i>18</i> | 0.056* <i>12</i> | 0.136 <i>12</i> |
| Quality of any previous documentation | -0.038 <i>22</i> | -0.043 <i>17</i> | -0.105 <i>17</i> |
| Requirements stability | -0.022 <i>25</i> | -0.010 <i>31</i> | -0.024 <i>31</i> |
| Stakeholder involvement | 0.038* <i>21</i> | -0.001 <i>32</i> | -0.002 <i>32</i> |
| Requirements management | -0.019 <i>27</i> | -0.018 <i>26</i> | -0.043 <i>26</i> |
| Regularity of spec and doc reviews | -0.010 <i>29</i> | -0.017 <i>28</i> | -0.041 <i>28</i> |
| Standard procedures followed | -0.053 <i>13</i> | -0.014 <i>30</i> | -0.034 <i>30</i> |
| Relevant experience of spec & doc staff | -0.050 <i>14</i> | -0.055 <i>13</i> | -0.133 <i>13</i> |
| Spec & doc effort | -0.007 <i>31</i> | -0.016 <i>29</i> | -0.039 <i>29</i> |
| Development staff motivation | -0.029 <i>23</i> | -0.018 <i>25</i> | -0.043 <i>25</i> |
| Relevant development staff experience | -0.040 <i>20</i> | -0.065 <i>9</i> | -0.158 <i>9</i> |
| Programmer capability | -0.048 <i>16</i> | -0.067 <i>8</i> | -0.163 <i>8</i> |
| Development staff training quality | -0.023 <i>24</i> | -0.031 <i>22</i> | -0.075 <i>22</i> |
| Defined process followed | -0.085 <i>7</i> | -0.056 <i>11</i> | -0.137 <i>11</i> |
| Development process effort | -0.061 <i>10</i> | -0.071 <i>7</i> | -0.171 <i>7</i> |
| Testing process well defined | -0.011 <i>28</i> | -0.018 <i>24</i> | -0.045 <i>24</i> |
| Quality of documented test cases | -0.019 <i>26</i> | -0.032 <i>20</i> | -0.077 <i>20</i> |
| Testing staff experience | -0.088 <i>6</i> | -0.089 <i>5</i> | -0.213 <i>5</i> |
| Testing effort | -0.040 <i>19</i> | -0.033 <i>19</i> | -0.080 <i>19</i> |
| Rework process quality | -0.068 <i>8</i> | -0.053 <i>14</i> | -0.129 <i>14</i> |
| Rework effort | -0.092 <i>5</i> | -0.083 <i>6</i> | -0.199 <i>6</i> |
| Process maturity | -0.050 <i>15</i> | -0.043 <i>16</i> | -0.105 <i>16</i> |
| Internal communications quality | 0.001 <i>32</i> | -0.020 <i>23</i> | -0.050 <i>23</i> |
| Project planning | -0.053 <i>12</i> | -0.031 <i>21</i> | -0.077 <i>21</i> |
| Subcontractor management | -0.010 <i>30</i> | -0.017 <i>27</i> | -0.042 <i>27</i> |
| Significant Subcontracts | 0.059* <i>11</i> | 0.043 <i>15</i> | 0.106 <i>15</i> |
| KLOC existing code base | 0.105* <i>4</i> | 0.089* <i>4</i> | 0.214 <i>4</i> |
| Complexity of existing code base | 0.043* <i>17</i> | 0.038 <i>18</i> | 0.094 <i>18</i> |
| Overall process and testing quality of existing code base | -0.062 <i>9</i> | -0.065 <i>10</i> | -0.157 <i>10</i> |
| * for SPEA and SRCC – values significant at p=0.95 values in italics indicate the rank of the specific factor for each measure | | | |

Box-plots presented on Figure 4-33 illustrate the impact of the two most influential qualitative model inputs on the number of *residual defects*. Analysis performed by Hall et al. [100] also confirms the very high importance of communication quality within development teams.

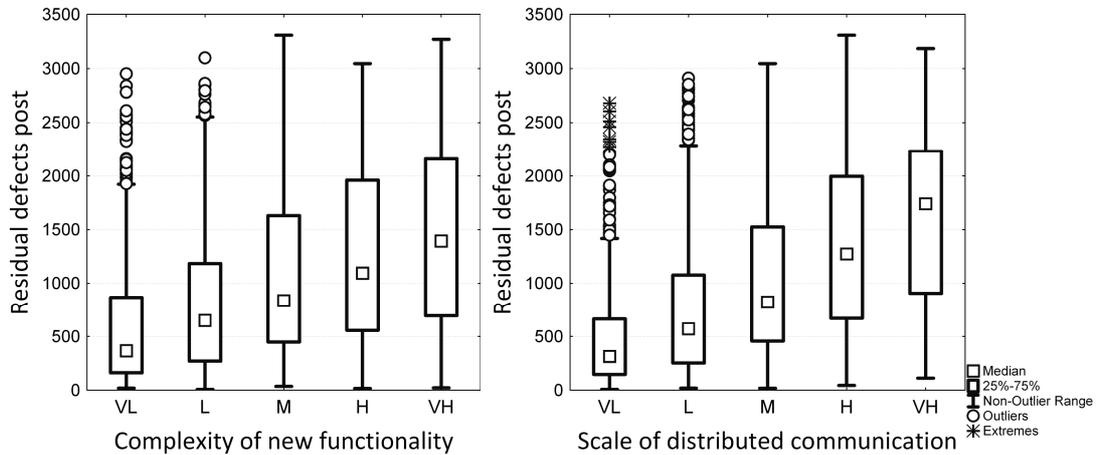


Figure 4-33 Impact of the two most important qualitative factors on *residual defects post*

We have validated this model's predictive accuracy against the empirical data available in [68, 69]. In this validation we used various evaluation measures. The results are summarized in Table 4-4. We can observe that the predictive accuracy of the model, expressed by the different measures presented in Table 14, increases with the size of the project and achieves the following highly desirable values in these types of model:

- Measures based on relative error (MMRE, MdMRE, BMMRE, MMER, MdMER) decrease significantly as project size increases. Such relationships between these measures and project size have also been reported in [236].
- Pred with different l levels increases (with one exception for Pred10 for medium-sized projects).

The lower prediction accuracy for smaller projects can be explained by the fact that the prior knowledge which we incorporated into the model based on expert opinions from the partner companies did not involve smaller projects. The validation using the data provided in previous sections shows that using the model with data outside the original model scope does not ensure accurate predictions.

We can observe a high value for MMRE compared to other measures based on the magnitude of relative error both for the dataset as a whole and the subsets depending on the project size. This was due to the fact that in 6 projects the magnitude of relative error was above 1 (the absolute prediction error was higher than the actual value) – even

several times higher than typical relative errors in other projects. These 6 projects caused the shift in MMRE but not so much in other measures based on relative error, which were all on a similar level.

Table 4-4 Values of model accuracy evaluation measures

| Evaluation measures \ Dataset for validation | Projects < 10 KLoC n = 7 | Projects ≥ 10 and < 50 KLoC n = 13 | Projects ≥ 50 KLoC n = 9 | All projects n = 31 |
|--|-----------------------------|--|----------------------------------|------------------------|
| R ² coefficient of determination | 0.003 | 0.523 | 0.984 | 0.931 |
| MMRE mean magnitude of relative error | 2.55 | 0.77 | 0.09 | 0.96 |
| MdMRE median magnitude of relative error | 0.49 | 0.28 | 0.06 | 0.27 |
| BMMRE balanced mean magnitude of relative error | 0.53 | 0.33 | 0.08 | 0.30 |
| MMER mean magnitude of relative error relative to estimate | 0.62 | 0.36 | 0.08 | 0.34 |
| MdMER median magnitude of relative error relative to estimate | 0.70 | 0.25 | 0.06 | 0.24 |
| Pred30 prediction at level 30 | 0.29 | 0.54 | 1 | 0.58 |
| Pred10 prediction at level 10 | 0.14 | 0.08 | 0.67 | 0.26 |

4.11 Comparison of BN models

Table 4-5 illustrates the main advantages and disadvantages of models discussed in this chapter and also in Section 2.5. When comparing the three discussed models developed by Fenton et al., which are the basis for this research, we can observe that:

- The Revised Defect Prediction Model is an improved version of the MODIST Phase-based Model with some added elements covering overall project management and the existing code base, and a revised new functionality subnet,
- The Project-level Model is intended for a different user group (project managers) than both of the previously discussed Defect Prediction Models (used by individual teams) as it uses data with a higher level of granularity,
- We can join different variations of the Phase-base Models to suit any type of development process,

- The Project-level and Phase-based models are complementary but we cannot use them together as one integrated model.

When analyzing models developed by Fenton et al. we not only used information provided by authors in published materials but also tested these models in various scenarios. We analyzed the models developed by other authors (Section 2.5) only using the information provided by those authors. We did not have direct access to their models nor did the authors provide the necessary details in their publications that would enable us able to rebuild and test their models.

Table 4-5 Comparison of pros and cons of analyzed BN models

| Model | Advantage | Disadvantage |
|---|---|--|
| MODIST Project-level by Fenton et al. [4, 67] | <ul style="list-style-type: none"> • contains trade-off component • overall project management ability • quality (defects) and user satisfaction prediction | <ul style="list-style-type: none"> • no defect prediction • need to use function points • difficult incorporation of new empirical data |
| MODIST Phase-based Defect Prediction by Fenton et al. [4, 75] | <ul style="list-style-type: none"> • ability to join models according to specific lifecycle • separate models for different combinations of activities | <ul style="list-style-type: none"> • focus on defect prediction only • no trade-off component • no reference to project schedule • weak reference to project management quality • need to use function points • effort expressed only in ranked type |
| Revised Defect Prediction by Fenton et al. [68, 69] | <ul style="list-style-type: none"> • improved new functionality subnet • more intuitive specification and documentation subnet • more detailed project management data than in phase model • reference to the metrics describing existing code base | <ul style="list-style-type: none"> • no trade-off component • focus on defect prediction only • no reference to project schedule • effort expressed only in ranked type |
| Software reliability by Bai et al. [11] | <ul style="list-style-type: none"> • ability to use the model with lack of knowledge some model parameters • positive verification of model predictions (but on small amount of data) | <ul style="list-style-type: none"> • reflects only one area in software engineering • small number of factors in a model |
| Estimation of inspection effectiveness by Cockram [45] | <ul style="list-style-type: none"> • many factors affecting model predictions • using variables known while running the model as parameters • positive verification of model predictions | <ul style="list-style-type: none"> • reflects only one area in software engineering |
| Independent V&V by Pai et al. [184] | <ul style="list-style-type: none"> • precise reflection of factors describing requirements specification | <ul style="list-style-type: none"> • reflects only one area in software engineering • lack of dependencies between specification factors (with one exception) |

| Model | Advantage | Disadvantage |
|---|--|---|
| Effort estimation by Bibi and Stamelos [18] | <ul style="list-style-type: none"> • capturing effort on various stages of software development • ability to reflect various development lifecycles • intentional avoiding too complex model structure leading to time-consuming calculations | <ul style="list-style-type: none"> • focus only on effort and skipping other development factors |
| Software testing by Wooff et al. [254] | <ul style="list-style-type: none"> • reflecting multiple aspects of software development • included utility measures | <ul style="list-style-type: none"> • reflects only testing process |

4.12 Summary

This chapter provided a short introduction to BNs and discussed different types of BN structure. The detailed review of selected existing BNs for software project risk assessment showed their potential in providing useful information for software managers. However, they still suffer from limitations preventing them being more widely used. The next chapter discusses the most important limitations and proposes some relatively simple solutions to improve existing models.

5 Limitations of existing BN models

This chapter discusses research challenges, which arise from the limitations of existing models, addressed by this thesis. It also discusses relatively simple solutions addressing some of these limitations (the more substantive solutions are the subject of the remaining chapters). This analysis shows that implementing these enhancements does not require much model developer's effort but can significantly improve model usability, especially by implementing dynamic discretisation. The new contributions of this chapter are the review of limitations of existing models and the discussion on how these models can be improved in various ways. Section 5.4 is based on [72].

5.1 *Integration of models*

Currently we have different models for specific tasks:

- MODIST Project-level Model – for managing a project overall,
- MODIST Phase-level Model – for predicting the number of defects in a single phase of development or several phases linked together,
- Revised Defect Prediction Model – focusing on predicting the number of defects for all or part of a project.

In general, all models incorporate defect prediction as a key factor in project success. But there are other important issues which are described in this chapter.

For complete project management we need an integrated model. Such an integrated model will generally be based on the MODIST models' philosophy, which is to enable various types of analyses with the main focus on:

- trade-off analysis between key project factors,
- defect prediction.

Generally, existing models partially allow such analysis but in a limited way because the MODIST Project-level Model does not contain the variable *number of defects*, and *defect rate* is only supported in a limited way (as an indicator to *quality delivered*). On the other hand, the two defect prediction models allow detailed defect prediction but both lack any means of incorporating effort in a quantitative way. An integrated model, with all the key project variables expressed on a numeric scale, will allow new types of analysis to be performed. Developing such an integrated model is the main challenge of this work (Chapters 6-7).

5.2 Incorporating new empirical data

Consider the node *new functionality delivered* in the MODIST Project-level Model. It is defined as shown in 11.

$$\begin{aligned} \text{new functionality delivered} = \\ \text{Normal}(30 * \text{total effective effort}^{0.8}, 100 * \text{total effective effort}^{1.7}) \end{aligned} \quad (11)$$

The parameters of this expression were evaluated using some empirical data. Suppose that now we would like to incorporate the new empirical data on the productivity achieved in a particular company. This expression for the *new functionality delivered* is not very informative for model users. Analysed BN models usually have important variables defined using such non-informative expressions (Table 5-1). They capture exponential relationships between the predictor and the dependent variable but the interpretation and the source of the numeric values used in these expressions is not explained. Therefore adjusting these expressions is a difficult task.

Another difficulty occurs when the variables needing adjustment are not observable. For example, how do we enter new data to update the impact of *new functionality* on *inherent potential defects* when the latter cannot be directly observed?

Table 5-1 Difficult expressions in existing models

| Model | Node | Expression |
|--|---|--|
| MODIST Project-level Model | total effort adjusted by Brooks factor | $\text{project duration} * (1 + \text{average \# people full time}^{0.8})$ |
| MODIST Project-level Model | new functionality delivered | $\text{Normal}(30 * \text{total effective effort}^{0.8}, 100 * \text{total effective effort}^{1.7})$ |
| MODIST Project-level Model | quality effort FD differential dummy | $\text{Normal}((30 * \text{total effective effort}^{0.8} - \text{new functionality delivered}) / (30 * \text{total effective effort}^{0.8} / 6), 0.1)$ |
| MODIST Phase-based Defect Prediction Model | inherent potential defects from poor specification | $\text{TNormal}(0.4 * \text{new functionality}^{1.3}, \max(0.001, 20 * \text{new functionality}), 0, 20000)$ |
| MODIST Phase-based Defect Prediction Model | inherent potential defects (indep. of poor specification) | $\text{TNormal}(0.7 * \text{new functionality}^{1.2}, \max(0.001, 20 * \text{new functionality}), 0, 20000)$ |
| Revised Defect Prediction Model | inherent potential defects | $\text{TNormal}(\min(10000, 30 * \text{effective KLOC}^{1.1}), 100 * \text{effective KLOC}, 0, 10000)$ |

There are some parts of the analysed BN models where the new empirical data can easily be introduced. The most significant such place is the node *defects per KLOC post release* in the MODIST Project-level Model. Here it is possible to provide information about what users really mean by the specific states of *quality delivered*. In a

sense: how do they describe the defect rate? With such an updated NPT for *defects per KLOC post release* it is possible to perform more accurate estimates for the number of defects for the whole project.

Let us assume, for example, that a company delivers specialized software with a higher quality requirement than for other types of software. This means that software with ‘average’ quality has a lower defect rate than in other types of software. The original and adjusted defect rates for *defects per KLOC post release* are summarized in Table 5-2. To improve clarity, we only include the mean values in these partitioned Normal expressions.

Table 5-2 Original and adjusted defect rates for different states of *quality delivered*

| State of quality delivered | Original | Adjusted |
|----------------------------|----------|----------|
| ‘Abysmal’ | 35 | 10 |
| ‘Very poor’ | 25 | 4 |
| ‘Poor’ | 12 | 2 |
| ‘Average’ | 4 | 1.2 |
| ‘Good’ | 2 | 0.7 |
| ‘Very good’ | 1.5 | 0.25 |
| ‘Perfect’ | 0.2 | 0.1 |

At first glance it may appear that it is possible to obtain precise predictions for defect rates using this model. But it is only this simple in theory. In reality we would not be able to obtain precise predictions – rather we would generate relatively wide intervals. This is because in this model a 7-point ranked scale of *quality delivered* is translated to a 7-interval numeric scale of *defects per KLOC post release*. We do not believe that realistic estimates can be performed with such a model structure. In fact this model wasn’t supposed to provide accurate predictions for the defects rate – the other two defect prediction models were created for this purpose. However, as shown in Table 5-1 the complexity of the expressions which define the influence of the project size on the potential number of defects make incorporating new empirical data into these models very difficult.

Other places where incorporating new empirical data may be useful are the prior distributions for the unconditional nodes (i.e. without the parents). Updating the distributions in these nodes is a relatively simple task as it only requires changing probabilities in the manually defined NPTs, or parameters in expressions in the case of nodes defined as expressions (usually using Normal distribution). However, such changes have only limited impact on model usability because most of the unconditional

nodes are the nodes describing the development process or the product, and these are nodes where observations are supposed to be provided by users anyway. When a user enters an observation into such a node the model predictions are unaffected by the prior distribution.

Yet another place where adjustments may be necessary are the nodes reflecting aggregations of the process factors often defined with weighted expressions (Section 4.5). Adjusting these expressions involves:

- changing the weights assigned to the parent (detailed) process variables,
- changing the variance in the TNormal distribution.

Table 5-3 summarizes the types of adjustments of existing BN models which should be enabled to the end users when they wish to incorporate new empirical data.

Table 5-3 **Types of adjustments of existing models**

| Type of adjustment | Difficulty of performing an adjustment |
|---|--|
| expressions in numeric nodes | high |
| partitioned expressions for numeric nodes depending on ranked nodes | medium, not always effective |
| prior distributions | low |
| aggregations of ranked nodes | low |

Software companies often routinely collect various types of empirical data. Productivity and defect rates are examples of metrics which can be relatively easily extracted from a past projects' database or, in the absence of such a database, might even be provided (albeit less accurately) by a project management expert. It is widely known that these rates heavily depend on the environment in which the software development takes place. Because of the difficulties in adjusting the expressions for various numeric nodes in existing models we chose to build a new model (discussed in Chapters 6 and 7) that would explicitly capture productivity and defect rates, and which users can adjust directly.

5.3 ***Using constants***

Many expressions in BN models, such as the MODIST Project-level model, capture empirical relations involving constants. For example, based on empirical studies the relationship between KLOC and FPs for C code can be described as a Normal distribution whose mean is $0.15 \cdot \text{FPs}$. However, if a particular company found that the relationship between KLOC and FPs involved a different constant from 0.15, then to

calibrate the model users would need to directly change the NPT expression, which is undesirable. The traditional way BN modellers get round this problem is to introduce a new variable (*constant parameter*) as shown in Figure 5-1. This approach avoids the problem of end-users having to edit NPTs (since they can just enter the constant as an observation). However, it results in an unnecessarily complex model.

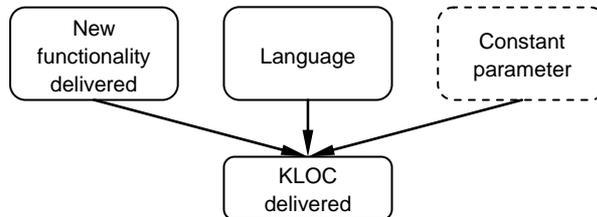


Figure 5-1 Simple model parameterization with additional node

It turns out that there is a way round this, namely by introducing genuine constants as part of the BN model. Table 5-4 illustrates how constants replace hard coded values (indicated in bold) in expressions for *KLOC delivered*. The same typographical convention applies in subsequent tables in this section. Figure 5-2 illustrates how constants can be defined in AgenaRisk.

Table 5-4 Current and adjusted partitioned expressions for *KLOC delivered*

| State of language | Current expression* | Adjusted expression |
|-------------------|---|---|
| 'Assembler' | Normal(0.32 * <i>fd</i> , <i>fd</i>) | Normal(<i>KLOC_Assembler</i> * <i>fd</i> , <i>fd</i>) |
| 'C' | Normal(0.15 * <i>fd</i> , <i>fd</i>) | Normal(<i>KLOC_C</i> * <i>fd</i> , <i>fd</i>) |
| 'C++' | Normal(0.08 * <i>fd</i> , <i>fd</i>) | Normal(<i>KLOC_Cpp</i> * <i>fd</i> , <i>fd</i>) |
| 'Java' | Normal(0.07 * <i>fd</i> , <i>fd</i>) | Normal(<i>KLOC_Java</i> * <i>fd</i> , <i>fd</i>) |
| 'Ada' | Normal(0.05 * <i>fd</i> , <i>fd</i>) | Normal(<i>KLOC_Ada</i> * <i>fd</i> , <i>fd</i>) |

* *fd* indicates *new functionality delivered* expressed in FPs

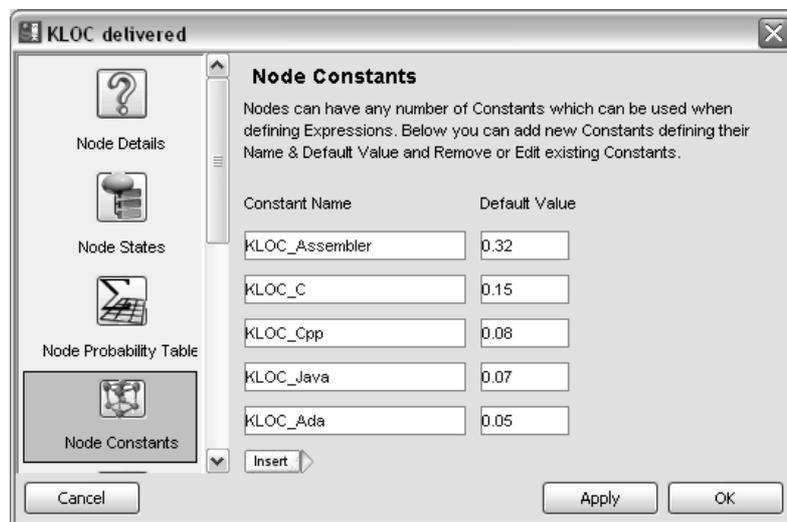


Figure 5-2 Defining constants for *KLOC delivered* in AgenaRisk

In the MODIST Project-level Model constants can be applied in various other places:

1. Definition of defects per KLOC post release depending on the level of quality delivered

In the original model defects per KLOC post release were defined by a set of partitioned expressions with a different expression for each state of the parent node: quality delivered. These partitioned expressions should be treated as a transformation of the qualitative node quality delivered to the numeric node defects per KLOC post release. For example, the model assumes that if quality delivered is ‘good’ then the defect rate is around ‘2’ defects per KLOC with a variance of ‘3’. We propose replacing the parameters in the expression for defects per KLOC post release.

Table 5-5 Partitioned expressions using constants for defects per KLOC post release in Project-level Model

| State of quality delivered | Current expression for defects per KLOC post release | Adjusted expression for defects per KLOC post release |
|----------------------------|--|--|
| ‘Abysmal’ | TNormal(35, 30, 0, 100) | TNormal(<i>mean_for_abysmal</i> , <i>deviation_for_abysmal</i> ² , 0, 100) |
| ‘Very poor’ | TNormal(25, 20, 0, 100) | TNormal(<i>mean_for_very_poor</i> , <i>deviation_for_very_poor</i> ² , 0, 100) |
| ‘Poor’ | TNormal(12, 12, 0, 100) | TNormal(<i>mean_for_poor</i> , <i>deviation_for_poor</i> ² , 0, 100) |
| ‘Average’ | TNormal(4, 5, 0, 100) | TNormal(<i>mean_for_average</i> , <i>deviation_for_average</i> ² , 0, 100) |
| ‘Good’ | TNormal(2, 3, 0, 100) | TNormal(<i>mean_for_good</i> , <i>deviation_for_good</i> ² , 0, 100) |
| ‘Very good’ | TNormal(1.5, 1.5, 0, 100) | TNormal(<i>mean_for_very_good</i> , <i>deviation_for_very_good</i> ² , 0, 100) |
| ‘Perfect’ | TNormal(0.2, 0.5, 0, 100) | TNormal(<i>mean_for_perfect</i> , <i>deviation_for_perfect</i> ² , 0, 100) |

2. Impact of process and people quality on total effective effort

In the original Project-level Model *total effective effort* is the *total effort adjusted by Brooks factor*, which also captures the level of overall *process and people quality*. The impact of *process and people quality* is hard coded into the model. It can be parameterized using constants as illustrated in Table 5-6. Here we introduced the constants to parameterize the multipliers which reflect the impact of *process and people quality* on *total effective effort*.

Table 5-6 Current and adjusted partitioned expressions *total effective effort*

| State of process and people quality | Current expression* | Adjusted expression |
|--|---|--|
| 'very low' | Normal(<i>tefa</i> * 0.12 , <i>tefa</i> / 10) | Normal(<i>tefa</i> * <i>ppq_very_low_on_effort</i> , <i>tefa</i> / 10) |
| 'low' | Normal(<i>tefa</i> * 0.5 , <i>tefa</i> / 10) | Normal(<i>tefa</i> * <i>ppq_low_on_effort</i> , <i>tefa</i> / 10) |
| 'medium' | Normal(<i>tefa</i> , <i>tefa</i> / 10) | Normal(<i>tefa</i> , <i>tefa</i> / 10) |
| 'high' | Normal(<i>tefa</i> * 2 , <i>tefa</i> / 10) | Normal(<i>tefa</i> * <i>ppq_high_on_effort</i> , <i>tefa</i> / 10) |
| 'very high' | Normal(<i>tefa</i> * 5 , <i>tefa</i> / 10) | Normal(<i>tefa</i> * <i>ppq_very_high_on_effort</i> , <i>tefa</i> / 10) |
| * <i>tefa</i> indicates <i>total effort adjusted by Brooks factor</i> expressed in person-months | | |

3. Priors for nodes without parents

We selected two nodes reflecting effort for which we decided to parameterize the prior distributions: *project duration* and *average # people full time*. Instead of hard coded priors we created constants which are used in the expressions (Table 5-7). This example shows how the same constant that represents the mean value for the TNormal distribution can also be used to determine the variance.

Table 5-7 Current and adjusted expressions for *project duration* and *average number of people full time*

| Model variable | Current expression | Adjusted expression |
|----------------------------|-----------------------------------|---|
| project duration | TNormal(24, 600 , 0, 120) | TNormal(<i>mean_duration</i> , <i>mean_duration</i> ² , 0, 120) |
| average # people full time | TNormal(12, 700 , 0, 250) | TNormal(<i>mean_num_people</i> , <i>mean_num_people</i> ² , 0, 120) |

We have also analyzed which parts of other existing models could be parameterized. Proposed changes are summarized in Table 5-8. We applied 4 types of parameterization:

1. Substituting the *mean* value in a TNormal distribution by a constant.
2. Substituting a fixed weight in a weighted mean expression by a constant.
3. Substituting part of an expression for the *mean* in a Normal distribution in a set of partitioned expressions.
4. Substituting transformed parameters in a Beta distribution from fixed α , β , lower bound and upper bound to constants *mean_KLOC*, *deviation*, *lower_bound*, *upper_bound*. The main benefit of this transformation is that users are able to provide more informative *mean_KLOC* and *deviation* instead of α and β parameters. We created a much more complicated equation than the original (but hidden from user) by entering the following estimates of parameters (Equations 12-15).

$$\alpha = \bar{x} \left(\frac{\bar{x}(1-\bar{x})}{\sigma^2} - 1 \right) \quad (12)$$

$$\beta = (1-\bar{x}) \left(\frac{\bar{x}(1-\bar{x})}{v} - 1 \right) \quad (13)$$

$$\bar{x} = \frac{\text{mean_KLOC} - \text{lower_bound}}{\text{upper_bound} - \text{lower_bound}} \quad (14)$$

$$v = \frac{\text{deviation}^2}{(\text{upper_bound} - \text{lower_bound})^2} \quad (15)$$

Table 5-8 Implementing constants in other existing models

| Model, variable and type of parameterization | Current definition | Adjusted definition |
|--|---|--|
| MODIST Phase-based defect prediction model: <i>new functionality implemented this phase</i> (1) | TNormal(300 , 100000, 0, 30000) | TNormal(<i>mean_functionality</i> , 100000, 0, 30000) |
| MODIST Phase-based defect prediction model: <i>KLOC (new)</i> (3) | as in Table 5-4 | as in Table 5-4 |
| MODIST Phase-based defect prediction model: <i>development process overall effectiveness</i> (2) | TNormal(wmean(1, <i>development process effort</i> , 2, <i>development process quality</i>), 0.0001, 0, 1) | TNormal(wmean(1, <i>development process effort</i> , weight_development_process_quality , <i>development process quality</i>), 0.0001, 0, 1) |
| Revised defect prediction model: <i>KLOC (new)</i> (4) | Beta(1.5, 21, 0, 1000) | Beta(<i>(mean_KLOC - lower_bound) / (upper_bound - lower_bound) * ((mean_KLOC - lower_bound) / (upper_bound - lower_bound) * (1 - (mean_KLOC - lower_bound) / (upper_bound - lower_bound))) / (deviation² / (upper_bound - lower_bound)² - 1), (1 - (mean_KLOC - lower_bound) / (upper_bound - lower_bound)) * ((mean_KLOC - lower_bound) / (upper_bound - lower_bound) * (1 - (mean_KLOC - lower_bound) / (upper_bound - lower_bound))) / (deviation² / (upper_bound - lower_bound)² - 1), lower_bound, upper_bound)</i> |

| Model, variable and type of parameterization | Current definition | Adjusted definition |
|---|---|---|
| Revised defect prediction model: <i>KLOC existing code base</i> (4) | Beta(1.5, 100, 0, 1000) | $\text{Beta}(\frac{(\text{mean_KLOC} - \text{lower_bound}) / (\text{upper_bound} - \text{lower_bound}) * ((\text{mean_KLOC} - \text{lower_bound}) / (\text{upper_bound} - \text{lower_bound}) * (1 - (\text{mean_KLOC} - \text{lower_bound}) / (\text{upper_bound} - \text{lower_bound}))) / (\text{deviation}^2 / (\text{upper_bound} - \text{lower_bound})^2) - 1, (1 - (\text{mean_KLOC} - \text{lower_bound}) / (\text{upper_bound} - \text{lower_bound}) * ((\text{mean_KLOC} - \text{lower_bound}) / (\text{upper_bound} - \text{lower_bound}) * (1 - (\text{mean_KLOC} - \text{lower_bound}) / (\text{upper_bound} - \text{lower_bound}))) / (\text{deviation}^2 / (\text{upper_bound} - \text{lower_bound})^2) - 1), \text{lower_bound}, \text{upper_bound})$ |
| Revised defect prediction model: <i>complexity of new functionality</i> (1) | TNormal(0.5, 0.2, 0, 1) | TNormal(mean_complexity, 0.2, 0, 1) |
| Revised defect prediction model: <i>testing process overall effectiveness</i> (2) | TNormal(wmean(1, testing effort, 3, testing process quality), 0.0001, 0, 1) | TNormal(wmean(1, testing effort, weight_testing_process_quality, testing process quality), 0.0001, 0, 1) |

5.4 Dynamic discretisation

Both the MODIST models and the Revised Defect Prediction Model contain a mixture of nodes that are qualitative (measured on a ranked scale) such as *overall management quality*, and nodes that are numeric, such as *defects found* and *KLOC (new)*. Because BNs generally require numeric nodes to be discretised, even if they represent continuous variables, there is an inevitable problem of inaccuracy because a set of fixed intervals has to be defined in advance. To improve accuracy in predictions we have to split the whole range of possible values for a particular node into a larger number of intervals. The more intervals we have, the longer the calculation time – since this includes generating the NPT from an expression in many cases. It is not simply a question of getting the right balance because in many cases we need to assume an infinite scale for which, of course, there can never be a satisfactory discretisation.

One proposed solution is to minimize the number of intervals by discretising more heavily in areas where the probability is expected to be higher and using wider intervals elsewhere. This approach fails in situations where we do not know in advance which

values are more likely to occur. This situation is inevitable if we seek to use the models for projects outside their original scope.

To give a feel for the problems caused by static discretisation, let us enter the following observations into the MODIST defect prediction model: *total defects in* = 80, *defects fixed* = 40. Note that the discretisation for *total defects in* is such that an observation of 80 cannot be distinguished from an observation of 61 since both values fall in the interval 61-80. Similarly an observation of 40 for *defects fixed* cannot be distinguished from 59 because both lie in the interval 40-59. Hence, although *total defects in* and *defects fixed* were entered as point values, actually they are still treated in the model as relatively wide intervals. Thus, the predicted distribution for *residual defects post* (Figure 5-3) has 3 intervals with non-zero probability, and both mean and median values are approximately 22, whereas we would expect a value of 40 (subtraction: 80-40). Such errors accumulate in this statically discretised model since it contains a chain of numeric nodes linked together – there are 8 numeric nodes between *KLOC (new)* and *residual defects post*.

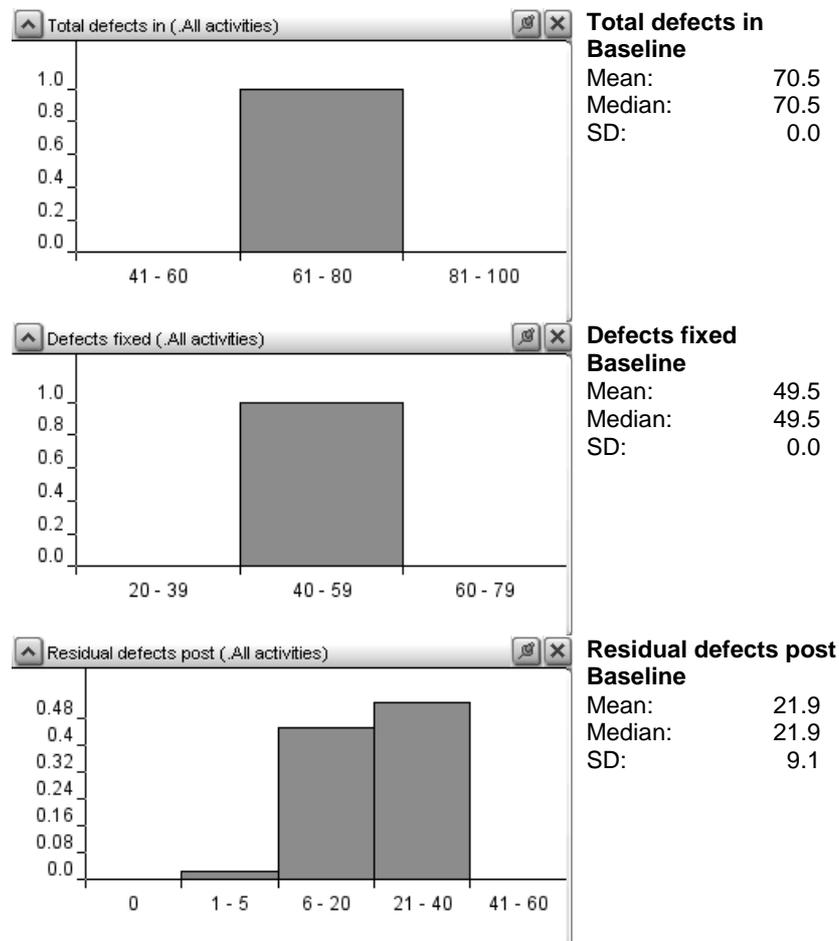


Figure 5-3 Predicted *residual defects post* with static discretisation

Table 5-9 illustrates the node states in this model for two nodes describing the size of the new software: *new functionality* and *KLOC (new)*. There are several intervals where the end value is around 50% or more higher than the start value. If we enter as an observation, the model cannot differentiate between a start value, an end value or any value in between. They are all treated as the same observation – the middle of the interval.

There were two main reasons for defining such node states:

- the availability of empirical data that the model was later validated against
- the calculation time which was acceptable for the number of states.

The node *KLOC (new)* contains intervals with high differences between starting and ending values. But those high differences are for values below 15 KLOC and over 200 KLOC – it was assumed that the *KLOC (new)* in a single phase would never lie outside these boundaries. Hence, we can expect that predictions for software size between 15 and 200 KLOC will be more accurate than those outside this range.

Table 5-9 Intervals for nodes *new functionality* and *KLOC (new)*

| New Functionality | | | | KLOC (new) | | | |
|-------------------|-------|----------------|--|------------|-------|----------------|--|
| Start | End | Interval Width | Percentage Difference Between Starting and Ending Values | Start | End | Interval Width | Percentage Difference Between Starting and Ending Values |
| 0 | 24 | 25 | - | 0 | 0,5 | 0,5 | - |
| 25 | 49 | 25 | 100,0% | 0,5 | 1 | 0,5 | 100,0% |
| 50 | 74 | 25 | 50,0% | 1 | 2 | 1 | 100,0% |
| 75 | 99 | 25 | 33,3% | 2 | 5 | 3 | 150,0% |
| 100 | 124 | 25 | 25,0% | 5 | 10 | 5 | 100,0% |
| 125 | 149 | 25 | 20,0% | 10 | 15 | 5 | 50,0% |
| 150 | 199 | 50 | 33,3% | 15 | 20 | 5 | 33,3% |
| 200 | 298 | 99 | 49,5% | 20 | 25 | 5 | 25,0% |
| 299 | 399 | 101 | 33,8% | 25 | 30 | 5 | 20,0% |
| 400 | 499 | 100 | 25,0% | 30 | 40 | 10 | 33,3% |
| 500 | 749 | 250 | 50,0% | 40 | 50 | 10 | 25,0% |
| 750 | 999 | 250 | 33,3% | 50 | 60 | 10 | 20,0% |
| 1000 | 1499 | 500 | 50,0% | 60 | 80 | 20 | 33,3% |
| 1500 | 1999 | 500 | 33,3% | 80 | 100 | 20 | 25,0% |
| 2000 | 2999 | 1000 | 50,0% | 100 | 125 | 25 | 25,0% |
| 3000 | 4999 | 2000 | 66,7% | 125 | 150 | 25 | 20,0% |
| 5000 | 7999 | 3000 | 60,0% | 150 | 175 | 25 | 16,7% |
| 8000 | 12000 | 4001 | 50,0% | 175 | 200 | 25 | 14,3% |
| 12001 | 15999 | 3999 | 33,3% | 200 | 300 | 100 | 50,0% |
| 16000 | 19999 | 4000 | 25,0% | 300 | 500 | 200 | 66,7% |
| 20000 | 30000 | 10001 | 50,0% | 500 | 10000 | 9500 | 1900,0% |

For the *new functionality* node we cannot find any range of intervals with relatively low differences between the lower and upper bounds of an interval. This means that we will have relatively inaccurate predictions for most software sizes expressed in function points.

This model contains several variables for predicting different types of defects. Most of them have similar states in terms of both the number of intervals and their ranges. Table 5-10 illustrates intervals for one of them: *defects found*.

Table 5-10 Node states for *defects found* in MODIST Phase-based Defect Prediction Model

| Defects found | | | | Defects found (cont.) | | | |
|---------------|------|----------------|--|-----------------------|-------|----------------|--|
| Start | End | Interval Width | Percentage Difference Between Starting and Ending Values | Start | End | Interval Width | Percentage Difference Between Starting and Ending Values |
| 1 | 4 | 4 | 400,0% | 1500 | 2000 | 501 | 33,4% |
| 5 | 19 | 15 | 300,0% | 2001 | 3000 | 1000 | 50,0% |
| 20 | 39 | 20 | 100,0% | 3001 | 4000 | 1000 | 33,3% |
| 40 | 59 | 20 | 50,0% | 4001 | 5000 | 1000 | 25,0% |
| 60 | 79 | 20 | 33,3% | 5001 | 6000 | 1000 | 20,0% |
| 80 | 99 | 20 | 25,0% | 6001 | 7000 | 1000 | 16,7% |
| 100 | 124 | 25 | 25,0% | 7001 | 8000 | 1000 | 14,3% |
| 125 | 149 | 25 | 20,0% | 8001 | 9000 | 1000 | 12,5% |
| 150 | 174 | 25 | 16,7% | 9001 | 10000 | 1000 | 11,1% |
| 175 | 199 | 25 | 14,3% | 10001 | 11000 | 1000 | 10,0% |
| 200 | 249 | 50 | 25,0% | 11001 | 12000 | 1000 | 9,1% |
| 250 | 299 | 50 | 20,0% | 12001 | 13000 | 1000 | 8,3% |
| 300 | 349 | 50 | 16,7% | 13001 | 14000 | 1000 | 7,7% |
| 350 | 399 | 50 | 14,3% | 14001 | 15000 | 1000 | 7,1% |
| 400 | 449 | 50 | 12,5% | 15001 | 16000 | 1000 | 6,7% |
| 450 | 499 | 50 | 11,1% | 16001 | 17000 | 1000 | 6,2% |
| 500 | 749 | 250 | 50,0% | 17001 | 18000 | 1000 | 5,9% |
| 750 | 999 | 250 | 33,3% | 18001 | 19000 | 1000 | 5,6% |
| 1000 | 1499 | 500 | 50,0% | 19001 | 20000 | 1000 | 5,3% |

Once we agree on a compromise between precision and execution time it may be difficult to change it in the future when the model needs to be updated (e.g. because new empirical data needs to be incorporated).

It should thus be clear that the static discretisation used in the existing models is a fundamental weakness both in terms of accuracy and adaptability. Fortunately, this problem has been addressed by very recent work on ‘dynamic discretisation’ (DD) algorithms [178, 180]. The general outline of the algorithm in [180] is as follows:

1. Calculate the current marginal probability distribution for a node given its current discretisation.
2. Split the discrete state with the highest entropy error into two equally sized states.
3. Repeat steps 1 and 2 until converged to an acceptable level of accuracy.
4. Repeat steps 1, 2 and 3 for all nodes in the BN.

The algorithm has now been implemented in the AgenaRisk toolset [3]. Using this toolset we can simply set a numeric node as a simulation node without having to worry about pre-defining intervals. It is sufficient to define a single interval $[x, y]$ for any variable that is bounded below by x and above by y , while for infinite bounds we only need introduce one extra interval.

In AgenaRisk we can specify the following global simulation parameters (for the whole model, not for individual nodes):

- *maximum number of iterations* – this value defines the maximum number of iterations that will be performed during the calculation; it directly influences the number of intervals that will be created by the algorithm and thus calculation time,
- *simulation convergence* – the difference between the entropy error value between subsequent iterations that we would like to achieve; the lower the simulation convergence, the more accurate the results but at the cost of computation time [2].

MODIST Phase-based Defect Prediction Model with DD

Table 5-11 illustrates the differences between node types for numeric nodes in the original (with static discretisation) and the revised (with DD) models. We do not present the number of states for numeric nodes in the revised model because they are not fixed. They rather depend on simulation parameters which are set by users.

Table 5-11 Numeric node types in original and revised models

| Node | Original model | | | Model with DD | |
|---|------------------|------------|------------------|------------------|------------|
| | Type of Interval | Simulation | Number of states | Type of Interval | Simulation |
| Prob avoiding spec defects | Continuous | No | 7 | Continuous | Yes |
| KLOC (new) | Continuous | No | 21 | Continuous | Yes |
| Total number of inputs and outputs | Integer | No | 5 | Integer | Yes |
| Number of distinct GUI screens | Integer | No | 5 | Integer | Yes |
| New functionality implemented this phase | Integer | No | 21 | Continuous | Yes |
| Inherent potential defects from poor spec | Integer | No | 25 | Integer | Yes |
| Inherent pot defects (indep. of spec) | Integer | No | 25 | Integer | Yes |
| Pot defects given spec and documentation adequacy | Integer | No | 26 | Integer | Yes |
| Total pot defects | Integer | No | 26 | Integer | Yes |
| New defects in | Integer | No | 24 | Integer | Yes |
| Total defects in | Integer | No | 38 | Integer | Yes |
| Defects found | Integer | No | 38 | Integer | Yes |
| Defects fixed | Integer | No | 39 | Integer | Yes |
| Residual defects pre | Integer | No | 38 | Integer | Yes |
| Residual defects post | Integer | No | 38 | Integer | Yes |
| Prob of avoiding defect in dev | Continuous | No | 5 | Continuous | Yes |
| Prob of finding defect | Continuous | No | 5 | Continuous | Yes |
| Prob of fixing defect | Continuous | No | 5 | Continuous | Yes |

Comparison of results

All calculations have been performed on a Pentium M 1.8 GHz Processor and 2 GB RAM under MS Windows XP Professional using AgenaRisk ver. 4.0.8b15. We ran calculations for the revised model using two values of the parameter *maximum number of iterations*: 10 and 25. We compared the results achieved using the model with DD and the original model.

We observed very significant changes in predicted values for the revised and original model. Those differences varied among nodes and scenarios. Most of the predicted means and medians were significantly lower in the revised model than in the original (the range of those differences was from -3% to -80%). This result fixed a consistent bias that we found empirically when we ran the models outside the scope of the MODIST project. Specifically, what was happening was that previously, outside the original scope, we were finding some probability mass in the end intervals. For example, an end interval like [10,000-infinity] might have a small probability mass, which without dynamic discretisation, will bias the central tendency statistics like the

mean upwards. Only in a few cases did we observe an increase in predicted values. In all of them the differences were small – the highest was around 40%, but most of them did not reach 10%.

We also observed a decrease in the standard deviation for the predicted distributions (from -8% to -80%). Partially, this is explained by the model no longer suffering from the ‘end interval’ problem that also skewed the measures of central tendency. However, another reason is that dynamic discretisation fixes the problem whereby nodes that are defined by simple arithmetic functions had unnecessary variability introduced. For example, nodes like *total potential defects*, *total defect in*, *residual defects post* no longer suffer from inaccuracies due entirely to discretisation errors affecting addition/subtraction.

The dynamic discretisation algorithm creates node states with narrow intervals in the area of highest probabilities and wide intervals where the probabilities are low (Figure 5-4). This ensures greater accuracy for predicted values. The number of intervals created for simulation nodes depends mainly on the parameter *maximum number of iterations*. We can observe that in the areas of higher probability more intervals have been created. Node states are fixed for the nodes not marked as simulation nodes. They do not change according to the predicted values for those nodes.

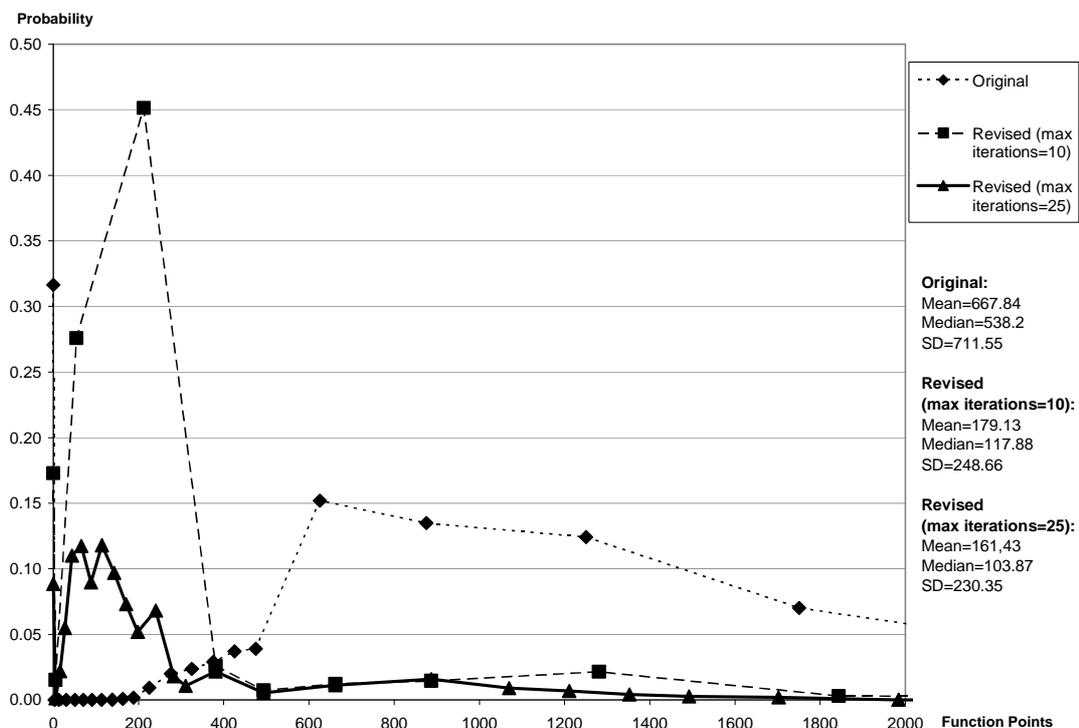


Figure 5-4 Predicted probability distributions for *residual defects post* for original and revised models in selected scenario

Predicted values for the node *residual defects post* decreased significantly using the model with simulation nodes compared to the original. This occurred for both tested values of *maximum number of iterations*. Predicted values for this node in both cases in the revised model were very similar (Figure 5-4). Our results show this was also true in other scenarios and for other nodes.

The difference in calculation times between different settings of *maximum number of iterations* is shown in Table 5-12. When this number is set to 10 the increase in calculation time (about 14%) will be hardly noticeable for users; but it results in significantly more accurate predictions. At 25 the increase in calculation time is significant and may not be merited given the marginal increase in accuracy from the setting 10.

Table 5-12 Comparison of calculation times for selected scenarios in original and revised model

| Model | Time (in minutes) | | Percentage difference in calculation times (compared to original model) | |
|---|----------------------|----------|---|----------|
| | Average | Shortest | Average | Shortest |
| Original | 0:08.5 | 0:06.7 | - | - |
| With DD (Maximum number of iterations = 10) | 0:09.7 | 0:07.5 | 14.1% | 11.9% |
| With DD (Maximum number of iterations = 25) | 1:02.1 | 0:45.8 | 730.6% | 683.6% |

The results of our research applying dynamic discretisation to the MODIST Defect Prediction Model has led us to the following conclusions:

1. Providing that we set a suitable value for the parameter *maximum number of iterations* the dynamic discretisation algorithm ensures greater accuracy of predicted values for simulation nodes than for nodes with fixed states.
2. Changing numeric node types to simulation nodes caused significant decrease in the predicted *number of defects* and the standard deviations in several nodes. This result fixed a consistent (pessimistic) bias we had found empirically in projects outside the scope of MODIST.
3. Applying the dynamic discretisation algorithm does not force model builders to define node states at the time of creation of the model. This is a very useful feature especially in those cases when we do not know in advance in which ranges we should expect higher probabilities.

4. We can mix simulation and traditional nodes in a single model. We can define fixed node states for some of the nodes while setting others as simulation.
5. The cost of increased accuracy and model building simplicity that comes with dynamic discretisation is increased calculation time but these increases are insignificant for values which still provide significant increases in accuracy.

5.5 Custom units of measurement

At present MODIST models use function points as the main measure for software size. The problem with this is that there are few companies who actually use functions points. This does not mean that companies using other size metrics cannot use these models. MODIST models use a set of indicators of project size. We can enter observations into those indicators rather than in the *new functionality* node directly. Figure 5-5 illustrates the scale of new functionality implemented subnet in the Phase-Based Model (the Project-level Model has fewer indicators).

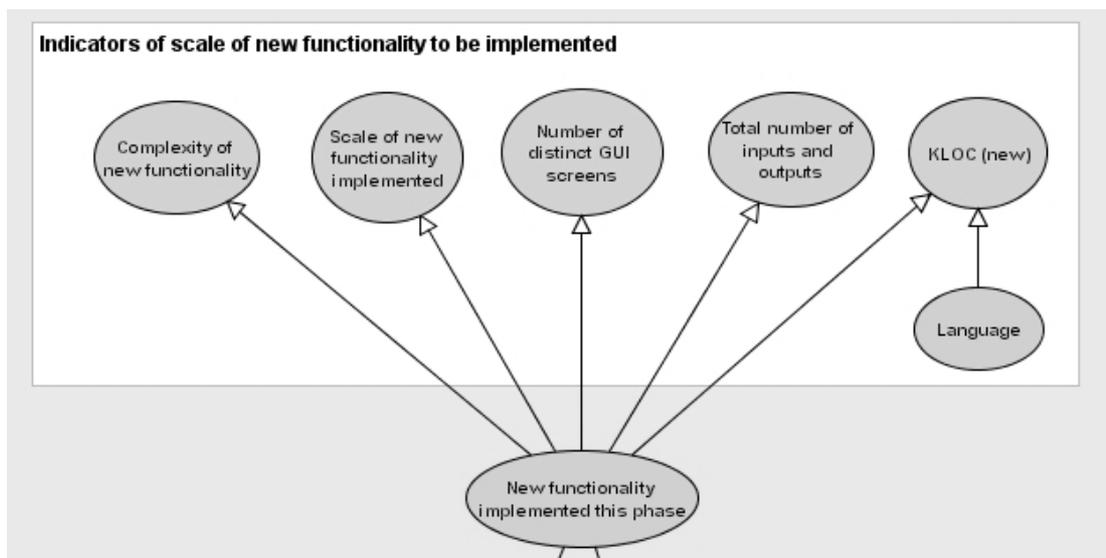


Figure 5-5 Scale of new functionality subnet in Phase-based Defect Prediction Model

For example, after entering an observation for software size expressed in KLOC and the programming language used, the model estimates a distribution for number of function points. Because this is a distribution, rather than a point value, it causes the following problem: When users observe a different value than that predicted for the number of *defects found*, the model updates by, essentially looking for the most likely explanations. The rationale behind the model was to revise the distributions for various process factors (indicating lower or higher process effectiveness). Although the model

does this, it also looks at *new functionality* as being a an explanation and because it value is *uncertain*, the model revises its probability distribution. So, for example, if fewer defects are observed then expected, one of the explanations the model finds for this is that *new functionality* (in addition to the process factors) must have been lower than expected. But we know that *new functionality* was fixed – its variability is just a result of not being able to enter it directly into the model.

Therefore, another challenge is to support various units of measurements for variables expressed on a numeric scale: effort, functionality, quality (defect rate). The user should be able to choose from one of a predefined set of units of measurement for each of these variables. However, in an ideal situation it should also be possible for users to choose their own custom units, even if they are not incorporated in the model during model development. Extending the existing model to incorporate more units of measurement and custom units is developed as a new model, called the Productivity Model in Chapters 6-7.

5.6 Defect prediction

Types of defects

So far all defects were treated equally and only captured quantitatively as the total number of defects. However, software companies and end-users might be interested in classifying defects by different criteria. The challenge of this work is to develop a model capable of predicting the proportions of defects categorised by their severity (minor, major, extreme). This model can use various predictors – both quantitative (e.g. project size) and qualitative (e.g. type of software, development process effectiveness). Such a model should be able to be linked with other defect prediction models in order to provide estimates on the number of defects by each category depending on defect severity.

Learning model

There is yet another completely separate challenge from the issues discussed earlier in the field of defect prediction. In all the above challenges we assumed that model parameters are estimated outside the model during a calibration process. Here, the challenge is to develop a defect prediction model that learns its parameters from observations entered into the model. The requirements for such a model are:

- It should be dynamic – it should capture successive testing and fixing iterations linked together.
- It should predict the number of defects likely to be found and fixed in future testing and fixing iterations.
- It should learn its parameters only on observations entered for the current project, not using data from other projects.
- It should learn its parameters based on the number of defects found and fixed (and possibly some process factors) observed in the past iterations.
- It should enable analysis of various scenarios with different levels of process factors in future testing and fixing iterations to see the impact of these process factors on predicted number of defects found and fixed.

5.7 Summary

Although existing BNs were successful in both research and industry they suffer some limitations discussed in this chapter. Some well defined research challenges have arisen from these limitations. Some of these challenges are addressed using relatively simple solutions. However, the most important challenges cannot be solved by extending existing models as they require building new models. These new BNs are discussed in the next five chapters.

6 The Productivity Model

This chapter introduces a new model for software project risk assessment at the overall project level. This new model, called the Productivity Model, adopts the basic philosophy of the existing models but overcomes their key limitations, of which two are the most important: capturing project trade-offs properly and enabling the use of local empirical data. This chapter also covers the process of calibrating this model using results from a questionnaire survey and the steps needed in preparing it. Also discussed are the issues which appeared while developing this model and validating it as well as possible future enhancements. This model is the main contribution of the whole thesis. This chapter is partially based on a previous work [206, 66].

6.1 Overview of the Productivity Model

The main goal for the Productivity Model was to enable improved software project risk assessment compared to the previous models. It captures the basic philosophy of past models but structurally is very different from them.

Figure 6-1 illustrates the main subnets in the Productivity Model together with the main relationships between them.

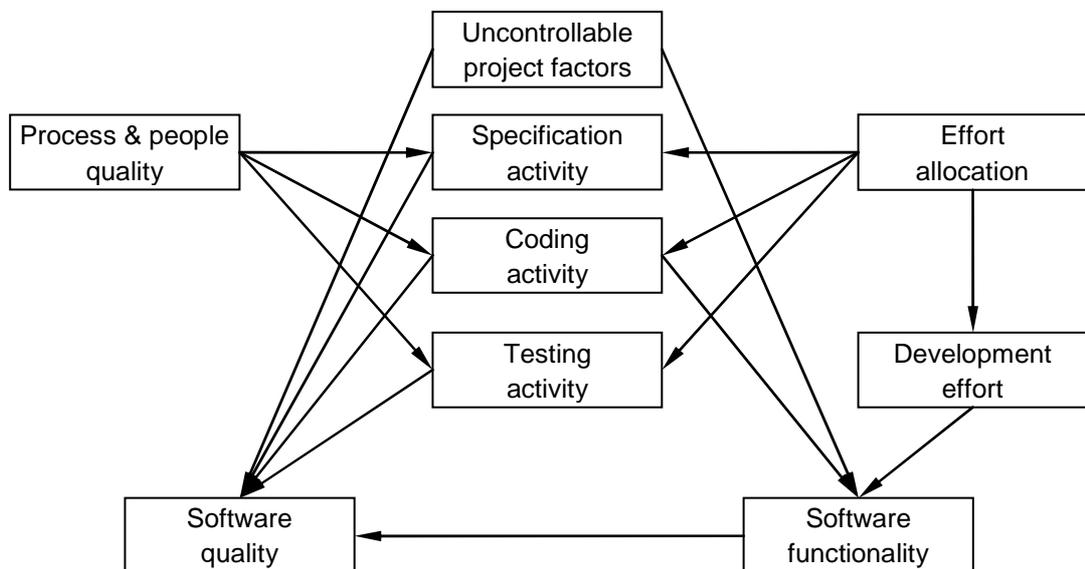


Figure 6-1 Main subnets of the Productivity Model

The key part of the model is a trade-off part. Relationships between these variables are influenced by the impact of other sets of variables reflecting:

- uncontrollable project factors,
- process and people quality,
- effort allocation,
- development activities (specification, coding and testing).

6.2 *Model features*

The Productivity Model provides the following unique features which were not available in previous models:

1. Allows us to enter custom *prior productivity rate* and *prior defect rate*. Companies typically gather some data from their past projects. Productivity and defect rates are among the easiest to be extracted from such databases. Even if a company does not collect effort data [214], they are often easy to estimate *post hoc*. In cases when providing such custom *productivity rate* and *defect rate* is not possible, these rates can be estimated by our PDR model discussed in Chapter 8.
2. Enables us to perform trade-off analysis with variables expressed on a numeric scale. In existing BN models some of them were expressed on a ranked scale.
3. Enables us to use different units of measurement up to the extent where the users perform the analysis using their custom units.
4. The impact of qualitative factors can be easily changed by changing weights in node expressions. We provide a questionnaire which can help determine users' opinions on the relationships between various model variables.
5. Allows target values for numeric variables to be defined as intervals, not just as point values. For example, this model can answer the question: how can we achieve a defect rate between 0.05 and 0.08 defects/FP for a project of a specific size and with other possible constraints.

Other important features of the Productivity Model, which at least partially were also available in the past models, include:

1. The model can be easily extended by adding other qualitative factors.
2. Numeric variables in this model are dynamically discretised resulting in the kind of benefits already discussed in Chapter 5.

6.3 Summary of model variables

We identified variables to be used in the model by analysing significant factors influencing productivity, effort and quality in existing models. We also used some of the factors which other researchers reported as significant (discussed in Chapter 3). Additionally, we discussed the influence of these factors with other experts in software engineering, asking them in a questionnaire survey (Appendix B.2) how they may influence various aspects of the software engineering process and its products. Table 6-1 summarizes the main variables of the Productivity Model. There are some additional hidden nodes not listed in this table and introduced to reduce model calculation time. A detailed definition of the whole model, including the hidden nodes, is contained in Appendix A.

Table 6-1 Summary of main variables in the Productivity Model

| Code / Type* | Name | Description |
|---------------------|-----------------------------------|--|
| q_input_doc R7 | quality of input documentation | quality of documentation reused from past projects, not the documentation produced in the current project during analysis and documentation process |
| pos_cust_inv R7 | positive customer involvement | the extent to which the customers are positively involved in the development (leads to both improvement of the software quality and project group productivity because the customers may relax the development group from some activities, like part of testing) |
| neg_cust_inv R7 | negative customer involvement | the extent to which the customers are excessively involved in the development (they want to participate in most of activities – thus the development team cannot use their resources productively) |
| dead_pres R7 | deadline pressure | the extent to which there is a pressure on delivering software faster than you normally would like to deliver such software |
| proj_compl R7 | project complexity | how complex is the task/problem – this is not complexity of source code |
| proj_novel R7 | project novelty | how novel is the task for your team |
| proj_scale R7 | project scale | how big is the project – in bigger projects you may need more percentage of effort on other activities than software development |
| n_prod_effort R7 | change in non-productive effort | how much non-productive effort in current project is higher or lower than non-productive effort in base project(s) |
| spec_effort R7 | change in effort on specification | how much effort on specification in current project is higher or lower than effort on specification in base project(s) |
| cod_effort R7 | change in effort on coding | how much effort on coding in current project is higher or lower than effort on coding in base project(s) |
| test_effort R7 | change in effort on testing | how much effort on testing in current project is higher or lower than effort on testing in base project(s) |

| Code / Type* | Name | Description |
|-----------------------------|---|---|
| change_total_effort CI S | change in total effort | how much total effort in current project is higher or lower than total effort in base project(s): revised effort to prior effort |
| staff_exper R7 | staff experience | level of staff experience |
| staff_motiv R7 | staff motivation | level of staff motivation |
| staff_educ R7 | staff education | level of staff education |
| team_org R7 | team organization | level of team organisation |
| app_meth_tool R7 | appropriateness of methods and tools used | how appropriate are methods and tools used during a project |
| dist_comm R7 | level of distributed communication | level of distributed communication between project participants |
| def_proc R7 | defined process followed | degree at which defined development process is followed during a project |
| lead_q R7 | leadership quality | level of leadership quality (higher and lower management level) |
| req_stab R7 | requirements stability | level of frequency of requirements being changed by customers/users |
| req_complet R7 | requirements completeness | degree at which requirements are completely captured from the customers/users |
| req_clear R7 | requirements clearness | level of understanding of requirements by the employees of software company |
| req_q R7 | requirements quality | aggregation of requirements stability, completeness and clearness |
| spec_ppq R7 | specification process and people quality | aggregation of all process and people factors for the specification activity |
| spec_effort_spec_eff R7 | impact of specification effort on specification effectiveness | how effort allocated to specification activity influences specification effectiveness; it depends on effort on specification and on coding – the more effort you spend on coding the more you need to spend on specification to produce documentation of the same quality |
| doc_q R7 | documentation quality | how good is the documentation produced during specification activity – aggregation of req_q, spec_ppq and spec_effort_spec_eff |
| cod_ppq R7 | coding process and people quality | aggregation of all process and people factors for the coding activity |
| cod_eff R7 | coding effectiveness | how effective the coding activity is – aggregation of cod_ppq, cod_effort, and doc_q |
| test_ppq R7 | testing process and people quality | aggregation of all process and people factors for the testing activity |
| test_effort_test_eff R7 | impact of testing effort on testing effectiveness | how effort allocated to testing activity influences testing effectiveness; it depends on effort on testing and on coding – the more effort you spend on coding the more you need to spend on testing to achieve the same effectiveness of testing |
| test_eff R7 | testing effectiveness | how effective the test activity is – aggregation of test_ppq, test_effort_test_eff and doc_q |
| uncontrollable_on_D R7 | uncontrollable on defect rate | impact of aggregated uncontrollable factors on defect rate |
| controllable_on_D CI S | controllable on defect rate | impact of aggregated controllable factors (specification, coding and testing effectiveness) on defect rate |

| Code / Type* | Name | Description |
|---|---|--|
| impact_on_D CI S | impact on defect rate | impact of development activities' effectiveness and of uncontrollable project factors on defect rate |
| D_uom L | defect rate – unit of measurement | unit of measurement for defect rate; must have an observation entered by user in each scenario |
| prior_D CI S | prior defect rate | defect rate for project/projects treated as a base to which a current project refers to |
| D_revised CI S | revised defect rate | defect rate for current project after adjusting by all relevant factors |
| uncontrollable_on_P R7 | uncontrollable on productivity rate | impact of aggregated uncontrollable factors on productivity rate |
| impact_on_P CI S | impact on productivity | impact of coding effectiveness and uncontrollable project factors on productivity in the current project |
| P_uom L | productivity rate – unit of measurement | unit of measurement for productivity rate; must have an observation entered by user in each scenario |
| prior_P CI S | prior productivity rate | productivity rate for project/projects treated as a base to which a current project refers to |
| P_revised CI S | revised productivity rate | productivity rate for current project after adjusting by all relevant factors |
| effort_uom L | effort – unit of measurement | unit of measurement for effort; must have an observation entered by user in each scenario |
| prior_effort CI S | prior effort | effort for project/projects treated as a base to which a current project refers to |
| effort_revised CI S | revised effort | effort for current project after adjusting by all relevant factors |
| num_units CI S | delivered number of units | size of the current project (expressed in unit of measurement depending on unit for productivity rate) |
| num_defects II S | number of defects | number of defects remaining in software after release |
| *Types of variables: L – labelled R – ranked; the number indicates the number of states CI – continuous interval II – integer interval S – simulation node (for CI and II) | | |

6.4 Model structure

The Productivity Model is intended mainly for analysis of trade-offs between:

- the size of delivered software (expressed as *number of delivered units*),
- the quality of delivered software (expressed as *revised defect rate* and *number of defects*),
- the effort required for developing the software (expressed as *revised effort*).

The structure of the Productivity Model (Figure 6-2) consists of subnets for:

1. Prior defect and productivity rates and prior effort

These are the values, taken from a typical past project, for the *prior defect rate*, the *prior productivity rate*, and the *prior effort*. It is the users' task to provide them as observations from the past project database. If the users are unable to provide them, the

defect and productivity rates are estimated as probability density functions, depending on the environmental factors influencing them, using the PDR Model (Section 8).

2. Uncontrollable project factors

These are the factors which are not under control of the software development company. They depend on the particular project and they include: *project complexity*, *project novelty*, *project scale*, *quality of input documentation*, *positive customer involvement*, *negative customer involvement*, and *deadline pressure*.

3. Development activities:

a. Specification and documentation

This subnet contains variables related to *requirements quality* (measured as a weighted mean of requirements stability, completeness and clearness), *specification process and people quality* and *change of effort on specification* which altogether influence the specification and documentation process effectiveness (aggregated as *documentation quality*).

b. Coding

This subnet contains variables related to *coding process and people quality*, *change of effort on coding* and the influence on *coding effectiveness* caused by the *documentation quality*.

c. Testing

This subnet contains variables related to *testing process and people quality*, *change of effort on testing* and the influence on testing effectiveness caused by the *documentation quality* and *coding effectiveness*.

4. Revised defect and productivity rates

These are the rates adjusted by the four groups of factors described above (uncontrollable project factors, *documentation quality*, *coding effectiveness* and *testing effectiveness*). The adjusted rates influence the relationships between variables in the trade-off component.

5. Trade-off component

Product functionality (*delivered number of units*) is calculated as the product of *revised effort* and *revised productivity rate*. Similarly, *number of defects* is calculated as the product of *delivered number of units* and *revised defect rate*.

Uncontrollable project factors as well as controllable (describing development activities) are all measured on a 7-point ranked scale with the following states: ‘extra

lower', 'much lower', 'lower', 'the same', 'higher', 'much higher', 'extra higher'. These states indicate that they are not measured on an absolute scale but reflect comparisons to appropriate variables in another project or other projects. This 'other' project is a completed project that is judged to be most similar to the current one – a 'template project'. States of these uncontrollable and controllable factors in the current project need to be compared by users to those in the 'template project'.

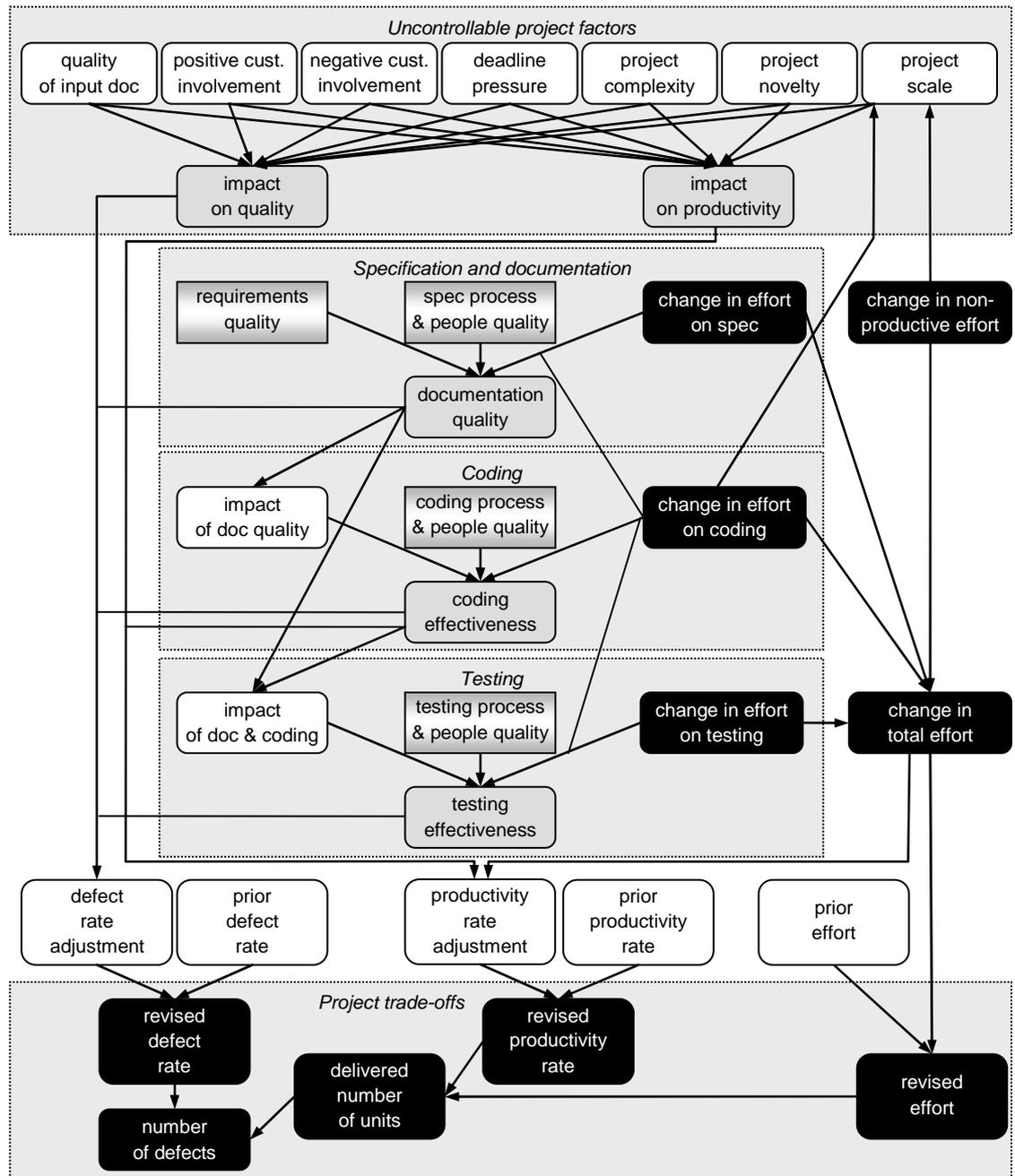


Figure 6-2 Structure of the Productivity Model

The key feature that enables the model to properly reflect project trade-offs is the way it models effort. It contains several variables reflecting effort values:

- **Effort on a specific activity (specification, coding, testing and non-productive) –** This reflects how much the effort on a specific activity changed in the current project compared to the template project. These variables influence the effectiveness of each activity. For example, *coding effectiveness* increases together with the increase in effort spent on coding. This leads to an increase in product functionality (*delivered number of units*) and product quality (*revised defect rate*). However, increasing the effort spent on testing does not automatically mean that the defect rate will decrease as it also depends on the effort spent on coding. A greater increase in effort spent on coding compared with effort spent on testing will result in an increase in product size but a decrease in quality (because the bigger product requires more testing to maintain its quality). Similarly, with specification effectiveness and project scale –bigger projects usually requires more effort on specification and documentation and typically use more non-productive effort (on administration, infrastructure etc.).
- **Prior effort –** The total development effort for the whole project based on those projects that are most similar to the template project.
- **Change in total effort –** This reflects how the total effort in the current project is different from the template project. It is calculated as the sum of effort on each development activity and then transformed into the range from ‘0’ to ‘1’.
- **Revised effort –** The value of total effort for the current project. It is calculated using the formula in Equation 16.

$$effort_revised = prior_effort * 10^{4 * change_total_effort - 2} \quad (16)$$

This expression assumes that the effort in the current project can vary from 0.01 of the *prior effort* to 100 times the *prior effort*. If this range is not wide enough for some practical needs it can be extended by replacing the value ‘10’ in the formula with a higher value.

Figure 6-3 illustrates the structure of the ‘Process and people quality subnet’ (PPQ). Our model estimates *process and people quality* separately for each development activity: specification, coding and testing. This feature enables the model to reflect the fact that often teams responsible for specific development activities are of

different quality, e.g. there are good programmers but bad testers. However, the structure of subnets for *process and people quality* in each activity is the same.

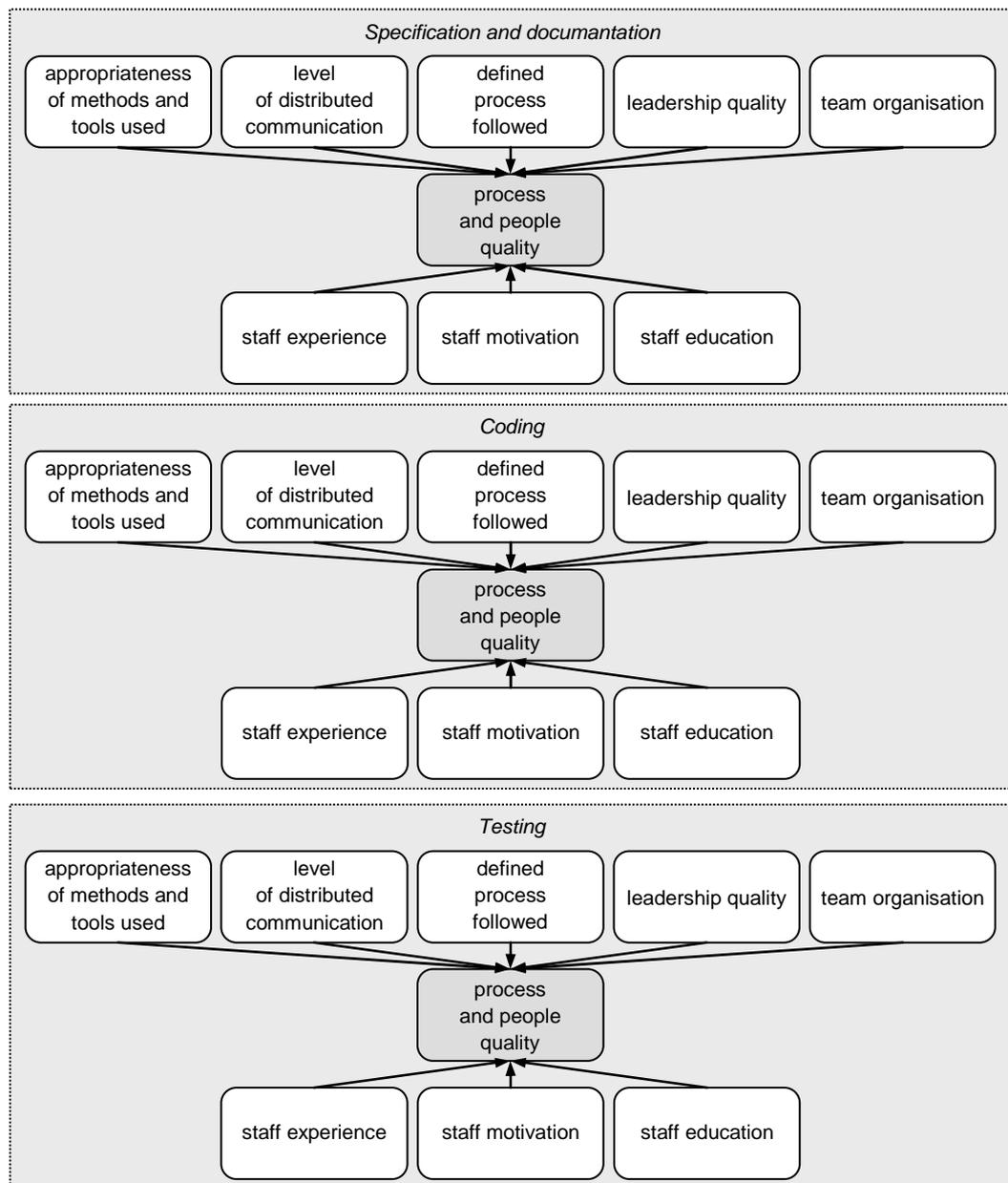


Figure 6-3 Structure of subnet for process and people quality

This subnet also enables entering observations to process and people factors which describe the whole project, not just particular activities. This can be a useful feature when a particular factor (such as *team organisation*) has the same state in all development activities. The model still propagates such observations to the appropriate factors in all development activities (Figure 6-4) and then to *process and people quality* in the specific activities (Figure 6-3). This is necessary because the main part of the

Productivity Model needs three separate inputs describing *process and people quality* for different activities even though they might be the same.

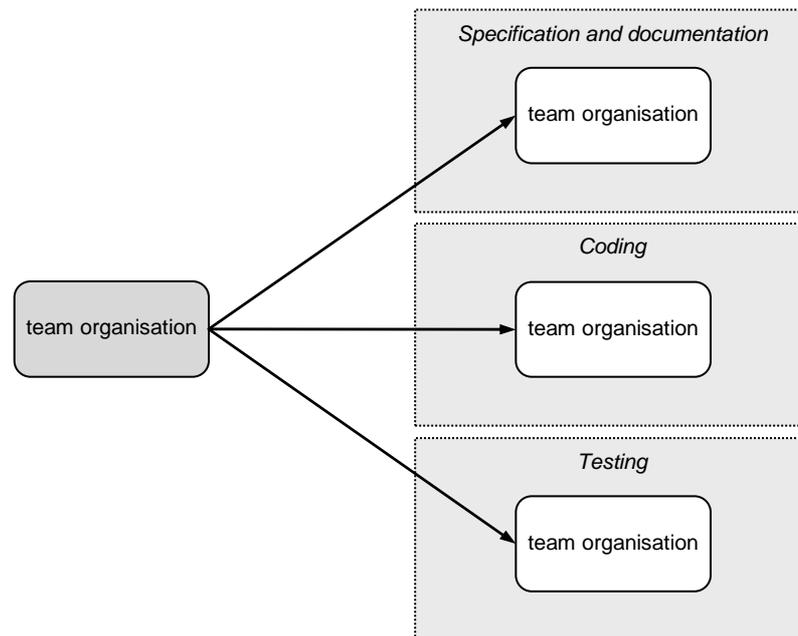


Figure 6-4 Propagating observation for process factor the same in all development activities

As discussed in Section 4.6 aggregated values of qualitative factors expressed on a ranked scale can be modelled either using causal relationships (from detailed variables to aggregated) or using the indicator approach (links from aggregated to detailed variables). In the Productivity Model we decided to use causal relationships because this structure makes it is easier to calibrate the model using the questionnaire survey data (details in Section 6.6). It may be argued that there are relationships between some of the qualitative variables which might suggest that we should use the indicator approach. However, we assume that whenever users assign observations to these qualitative variables they do it for all of them. In this case the relationships between these variables would not be reflected anyway.

Uncontrollable project factors also act as ‘common causes’. In the Productivity Model they influence *revised defect rate* and *revised productivity rate*. This structure increases model calculation time. However, our tests show that model calculation times are at an acceptable level – depending on the scenario it takes up to 4 minutes to calculate the model on a laptop with Intel Pentium M 1.8 GHz and 2 GB of RAM.

6.5 Prior probabilities in ranked variables

Generally, companies tend to improve their development processes over time. But there are situations when the quality of process factors decrease. One of the most important single factors influencing the process quality is the *leadership quality*. A leader must have an established position in the company, be familiar with the products, processes and people in his company. It usually takes some time to achieve such a high leadership level.

Suppose now, that the key leader leaves the company for some reason. Such an event may have a catastrophic impact on the current project or upcoming projects. In the long term this could be compensated for as new people's experience increases. But in the short term this cannot be easily compensated – you cannot easily find a replacement for a key person in the project. As a result in the short term we generally observe higher fluctuations in the level of the qualitative factors both downwards (more often) and upwards (less often).

In our model, process factors are expressed as changes compared with the template project(s). We assume that our model should work properly when comparing the current project both to a much older project and to the newest (if they are similar). Therefore, we assume that changes in the level of the qualitative factors may either be negative or positive. From this point of view, without any other information, our belief that they will increase is exactly the same as our belief that they will decrease. As a result we decided to assign symmetrical probability distributions for these factors. The point of symmetry is the middle value of these factors: 'the same' which is also the most probable value. With the higher changes of the factor (downwards or upwards) – the lower probability is assigned to these levels.

6.6 Model calibration

We calibrated the Productivity Model using the results of a questionnaire survey. The aim of this survey was to determine software engineering experts' opinions about the impact of relevant factors on software quality and development team productivity. The full content of the questionnaire is presented in B.2 and raw results obtained from respondents in B.3.

We received responses from 9 experts out of 15 who were asked. The answers provided were often contradictory. For example, according to one of the respondents the

quality of input documentation has 5 times higher impact on software quality than the *project complexity*. On the other hand, another respondent believed that *project complexity* has 2 times higher impact on software quality than the *quality of input documentation*. When analyzing the results we found that such differences were very common.

We decided to assign a weight to each respondent's questionnaire depending on:

- respondent's experience (as provided by each respondent),
- known approach of each respondent to filling the questionnaire (motivation and time spent on thinking about providing responses).

Table 6-2 illustrates summarized results of this survey by presenting the mean, median and weighted mean of the values provided by respondents. Predictors' impact on dependent variables was measured from '0' (no impact) to '10' (the highest impact). The bottom four rows illustrate how much productivity and software quality is expected to decrease or increase should a combination of all possible factors be least or most favourable. For example, when all factors are least favourable, project group productivity is expected to decrease to 0.21 of its typical value; when all factors are most favourable, project group productivity is expected to increase to 5.63 times higher of its typical value.

Figure 6-5 illustrates respondents' opinions on the expected change in productivity and software quality depending on different combinations of controllable and uncontrollable factors. The scale for these changes is from '0' to '10'. Value '0' indicates the greatest possible decrease in productivity and software quality, value '5' means no change and value '10' indicates the highest possible increase.

We needed to incorporate the results of this questionnaire survey into the Productivity model. We entered the weighted mean values from Table 6-2 as the weights for a particular dependent variable. However, to incorporate results from the four bottom rows in Table 6-2 and from Figure 6-5 we needed to find the expressions which best suited the gathered data. When analysing the perceived impact of controllable and uncontrollable factors on productivity and software quality we found that these relationships are most accurately reflected by the *weighted min* function.

Table 6-2 Summary of the questionnaire survey results

| Influence on | Factor | Response value provided | | |
|---|---|-------------------------|---------------|--------|
| | | Mean | Weighted mean | Median |
| software quality (controllable factors) | effectiveness of analysis and documentation process | 8.1 | 7.2 | 8.0 |
| | effectiveness of coding process | 7.1 | 6.5 | 8.0 |
| | effectiveness of testing process | 9.1 | 8.3 | 10.0 |
| software quality (uncontrollable factors) | project complexity | 7.3 | 7.4 | 7.0 |
| | project novelty | 6.9 | 6.8 | 7.0 |
| | project scale | 5.8 | 5.3 | 7.0 |
| | deadline pressure | 8.6 | 8.5 | 9.0 |
| | quality of input documentation | 6.0 | 6.3 | 5.0 |
| | positive customer involvement | 7.3 | 7.2 | 7.0 |
| | negative customer involvement | 5.2 | 5.5 | 4.0 |
| project group productivity (uncontrollable factors) | project complexity | 7.1 | 7.1 | 8.0 |
| | project novelty | 7.9 | 8.0 | 7.0 |
| | project scale | 5.7 | 5.9 | 7.0 |
| | deadline pressure | 7.3 | 7.0 | 8.0 |
| | quality of input documentation | 5.6 | 6.1 | 5.0 |
| | positive customer involvement | 6.2 | 5.4 | 7.0 |
| | negative customer involvement | 5.2 | 4.8 | 4.0 |
| overall process and people quality | staff experience | 8.4 | 8.3 | 9.0 |
| | staff motivation | 8.8 | 8.7 | 9.0 |
| | staff education | 6.6 | 6.7 | 6.0 |
| | team organisation | 7.8 | 7.8 | 8.0 |
| | appropriateness of methods and tools used | 7.8 | 7.4 | 8.0 |
| | level of distributed communications | 6.9 | 6.8 | 8.0 |
| | well defined process followed | 6.2 | 5.8 | 6.0 |
| | leadership quality | 7.3 | 7.2 | 7.0 |
| effectiveness of analysis and documentation process | requirements quality | 8.3 | 8.5 | 9.0 |
| | process and people quality | 6.9 | 6.9 | 8.0 |
| | effort | 7.7 | 7.8 | 8.0 |
| effectiveness of coding process | documentation quality | 7.0 | 6.6 | 8.0 |
| | process and people quality | 7.7 | 7.9 | 8.0 |
| | effort | 6.9 | 6.8 | 7.0 |
| effectiveness of testing process | documentation quality | 7.9 | 8.1 | 8.0 |
| | process and people quality | 7.6 | 8.1 | 8.0 |
| | effort | 7.6 | 8.3 | 8.0 |
| software quality | all factors overall – worst possible | 0.19 | 0.14 | 0.10 |
| | all factors overall – best possible | 5.94 | 5.95 | 5.00 |
| project group productivity | all factors overall – worst possible | 0.23 | 0.21 | 0.21 |
| | all factors overall – best possible | 5.56 | 5.63 | 4.00 |

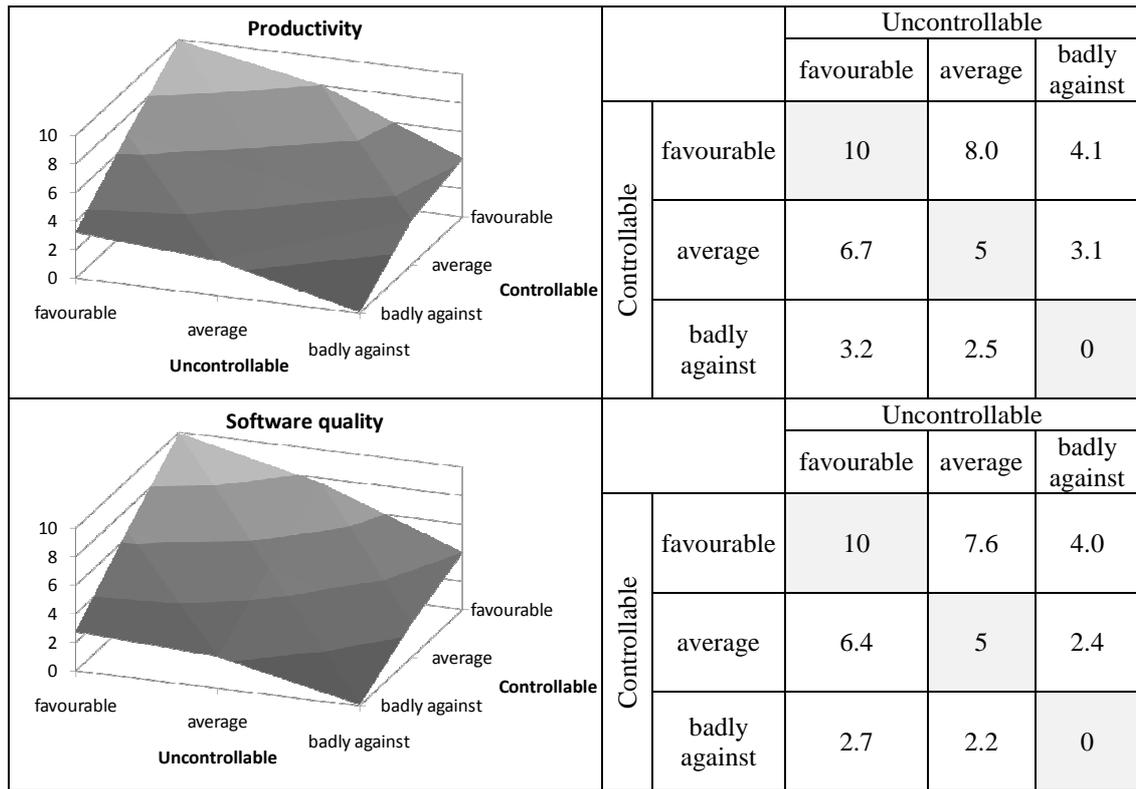


Figure 6-5 Impact of controllable and uncontrollable factors on productivity and software quality

We found the weights which most accurately reflected the impact of combinations of controllable and uncontrollable factors on productivity and software quality, by minimising the mean magnitude of relative error (MMRE) between the actual values provided by respondents and the estimates produced by the *weighted min* function. The MMRE achieved for these weights were 0.068 for productivity and 0.049 for software quality. Estimated weights are summarized in Table 6-3. The values of these weights suggest that respondents believe that controllable factors have greater impact on both productivity and software quality than the uncontrollable factors.

Table 6-3 Weights for controllable and uncontrollable factors

| Predictors | Dependents | Productivity | Software quality |
|------------------------|----------------------|--------------|------------------|
| | Controllable factors | | 2.422 |
| Uncontrollable factors | | 1.581 | 1.667 |

The overall combined impact of both controllable and uncontrollable factors on the productivity and software quality is presented in Table 6-2 (bottom 4 rows). Again,

we had to find an expression that reflected this data. Expressions for *revised productivity rate* and *revised defect rate* are shown in Equations 17-19.

$$d_{revised} = prior_d * 2.6^{-4 * impact_on_defects + 2} \quad (17)$$

$$p_dummy = prior_p * 2.4^{-4 * impact_on_productivity - 2} \quad (18)$$

$$p_{revised} = p_dummy * 10^{-4 * change_total_effort + 2} \quad (19)$$

With these expressions the model assumes that *revised defect rate* can change from 0.15 to 6.76 of the *prior defect rate* and that *revised productivity rate* ranges from 0.17 to 5.76 of the *prior productivity rate*. The model calculates the *revised productivity rate* in two steps because it depends on *change in total effort* and *change in other factors*.

This model can be used in software companies after basic tailoring. This step involves adding new detailed factors or removing those which are not relevant. Such changes require setting new weights in the nodes using weighted expressions. Regardless of whether the list of detailed factors has been changed or not, another task here is to recalibrate the model using the questionnaire provided or a similar one. Although we already calibrated the model we still suggest recalibrating the model to ensure that it more accurately reflects the terminology and development process used in a particular company.

6.7 Issues arising from model development

Effort on coding – coding effectiveness

Initially the effort allocation nodes reflected the difference in proportion of effort allocated on a particular task between the current project and the template project. For example, if in the past 20% of total effort was allocated to specification and in the current project 30% is allocated to specification, then this means that the difference in the proportion of effort allocated to specification was 10%. This way of capturing effort allocation meant that all the effort allocation nodes must sum to '0'. We could use this as a constraint node in the trade-off analysis. For example, if we knew that we now spend proportionally more effort on quality assurance (specification and testing) we automatically spend proportionally less effort on coding.

During testing we found that the model made inconsistent and unexpected predictions with these type of nodes. This was due to the incorrect assumption about the impact of these effort allocation nodes on effectiveness of each development activity.

Let us explain this with a hypothetical example of effort allocation as illustrated in Table 6-4. Initially the nodes describing effort allocation used the values from the column ‘difference in proportion’. We can observe that in the current project total effort increased by just over 50%. The absolute amount of effort allocated on any activity was at least on the previous level. However, because effort on specification and testing doubled while effort on coding only increased by 5%, the proportion of effort spent on coding actually decreased. In this case the initial model predicted that the coding process was less effective, which is not what it should predict. Effort allocated on coding was still higher than before and coding effectiveness should not depend on how effort is allocated on other activities.

Table 6-4 Example of effort allocation

| Activity | Effort in template project | Effort in current project | Proportion of total effort (past) | Proportion of total effort (current) | Difference in proportion (current – past) | Ratio (current / past) |
|----------------|----------------------------|---------------------------|-----------------------------------|--------------------------------------|---|------------------------|
| Specification | 100 | 200 | 0.2 | 0.26 | 0.06 | 2 |
| Coding | 200 | 210 | 0.4 | 0.28 | -0.12 | 1.05 |
| Testing | 150 | 300 | 0.3 | 0.39 | 0.09 | 2 |
| Non-productive | 50 | 50 | 0.1 | 0.07 | -0.03 | 1 |
| Total | 500 | 760 | 1 | 1 | 0 | 1.52 |

Modelling impact on adjusted rates

Initially the nodes *revised defect rate* and *revised productivity rate* were modelled as partitioned expressions. In this case the distribution of the revised rates was defined as a set of expressions – one expression for each state of *impact on defect rate* and *impact on productivity*. For example, if *impact on defect rate* was ‘extra lower’ (‘EL’) then *revised defect rate* was expressed as shown on Equation 20, when *impact on defect rate* was ‘much lower’ (‘ML’) then *revised defect rate* was expressed as shown on Equation 21. The multipliers adjusting the *prior defect rate* (μ parameter) were the following for different *impact on defect rate*: ‘0.1’, ‘0.2’, ‘0.5’, ‘1’, ‘2’, ‘5’, ‘10’.

$$D_{revised}(adj='EL') = \text{Normal}(0.1 * prior_D, (0.5 * prior_D)^2) \quad (20)$$

$$D_{revised}(adj='ML') = \text{Normal}(0.2 * prior_D, (0.5 * prior_D)^2) \quad (21)$$

This definition meant that when no observations were entered in any parent of *impact on defect rate* the mean and median of this adjustment node was not exactly 1. At that time (before obtaining the results from the questionnaire survey) the model did not contain *weighted min* expressions, but instead used *weighted means* to model the

impact of various factors on *revised defect rate* and *revised productivity rate*. The model assumed that this adjustment was actually always higher than 1 which meant that the *revised defect rate* was always higher than the *prior defect rate*. We expected the model to predict a *revised defect rate* very similar to the *prior defect rate*, possibly only with a higher variation as some of the factors were unknown (left without observations).

We converted the *revised defect rate* to a single expression of the form: ' $prior_D * a^{impact\ on\ defect\ rate}$ ', with a indicating the strength of the influence of the *impact on defect rate*. We observed that these deviations became significantly lower but they were not completely removed. This problem also affected the *revised productivity rate* and the *revised effort*. We solved the problem by performing the calculations in 5 steps:

1. Enter observations to all relevant ranked nodes (describing process and project factors),
2. Calculate the model.
3. Read the mean (or median if it is more appropriate) value of all *impact* nodes and enter them as observations.
4. Calculate the model again.
5. Check and analyze the results.

This solution can be automated using the AgenaRisk API. However, it prevents us from performing any analysis involving back-propagation. For example, suppose we set some target values for *revised defect rate* and *delivered number of units*, and we would like the model to predict the total *revised effort*. Additionally we would like to get an information on effort allocation and the levels of the process factors in order to achieve these targets. Such an analysis can still be performed in the traditional way while keeping in mind that the predictions may be slightly affected by the imperfect capturing of the impact of adjustments on *revised effort*, *revised productivity rate* and *revised defect rate*.

Adjusted rates as a single expression

Changing the modelling of adjusted rates from partitioned expressions to a single exponential-like expression also resulted in the following.

- Model runs faster – there is a single expression to be used for each of the revised variables instead of 7.
- The difficulty caused by the possible change of scale in adjustment nodes is removed. If someone wants to change the scale for the adjustment rate e.g. from

a 7-point ranked scale to a 9-point ranked scale, they would have to manually change the multipliers in each of the partitioned expressions in the revised rates according to the new list of states. Now, with just a single expression in the revised variables, no additional change is required, apart from the change of states itself.

Process and people quality as separate subnet

Complex BN models are often divided into multiple subnets to reduce model calculation time, the model's memory requirements, and to improve the visualisation of such complex models. These subnets (called 'risk objects' in AgenaRisk) are calculated sequentially so that results from one of them can be passed as inputs to the next one and so on.

The process and people quality subnet is modelled as a separate risk object. It is calculated first – before the main part of the Productivity Model. The results from the output nodes of this subnet (*process and people quality* in all development activities) are then passed to the main part of the Productivity Model and then this main part is calculated.

The disadvantage of this structure is that the PPQ subnet cannot use observations entered in the main part of the model. Therefore, some decision problems cannot be solved. For example, if we enter some observations for effort, functionality, target for defect rate and some (but not all) process factors, then the model cannot answer the question: what level of *team organisation* do we need in order to meet these specific targets using our available resources? However, we believe that in such cases it might be enough for users to obtain estimates for aggregated *process and people quality* for specific development activities from the main part of the model.

6.8 Future Enhancements to the Productivity Model

Compatibility with causal framework for risk

Currently the Productivity Model is not compatible with the causal framework for risk described earlier in Section 4.3. To simplify the model creation, the uncontrollable project factors do not influence any factors in the development activities. Rather, they are directly linked to revised productivity and defect rates. This means that the Productivity Model is not a fully causal model.

However, the project risk factors are still grouped according to the categories to which they belong . The defined main categories are currently:

- uncontrollable project factors,
- process and people factors,
- effort allocation,
- prior rates (productivity and defect),
- trade-off variables.

It is possible to extend this model to make it compatible with the causal framework for risk and to make the model clearer for future users. In May 2008 we started a research project [30] which aims to ensure this compatibility. The main aims of the project are:

- building a methodology supported by software tools that allows models that are compatible with the causal framework for risk to be built more easily,
- extending the tools' the presentation layer of the model structure and the results.

Incorporating code reuse

The current version of the Productivity Model assumes that no code is reused from past projects. To make the model more realistic, code reuse may be incorporated into the model. This is relatively simple in terms of estimating functionality delivered from a given effort. However, modelling the impact of code reuse on *revised defect rate* may not be so easy because we need to know the quality of the reused code. It will probably be of better quality than the currently developed new code because it has already been tested. However, to model this properly more details are required about the reused code, like:

- information on whether the reused code is a complete module (or set of modules) or rather smaller pieces of code from various modules,
- how good was the development process when the reused code was originally written and tested,
- what is the typical proportion of reused code in the total code delivered.

We have already prepared data about the last of these issues during the questionnaire based survey which was mainly intended to gather the data to calibrate the Productivity Model. Table 6-5 summarizes the relevant data for the proportion of code reuse.

Table 6-5 Summary of code reuse statistics obtained from questionnaire survey

| Value | Response value statistic | | |
|-----------------------|--------------------------|---------------|--------|
| | Mean | Weighted mean | Median |
| typical | 0.50 | 0.50 | 0.50 |
| lowest (lower bound) | 0.14 | 0.14 | 0.00 |
| highest (upper bound) | 0.72 | 0.75 | 0.80 |

Integrating detailed defect prediction with trade-off analysis

The Productivity Model is already an integrated model in the sense that it can predict both effort and quality (*revised defect rate, number of defects*). However, the prediction for *number of defects* is less detailed than in the earlier defect prediction models discussed in Chapter 4. Both of the earlier models predict not only the total number of residual defects, but also detailed data on:

- potential number of defects depending on project size,
- potential number of defects adjusted by specification and documentation adequacy,
- number of defects inserted as a result of imperfect coding process,
- number of defects found,
- number of defects fixed,
- number of defects in the reused code (Revised Defect Prediction Model only).

A possible extension to the Productivity Model is to incorporate such detailed defect prediction with the existing trade-off analysis. The model would then capture the impact of development activities on detailed defect data and effort estimation.

User satisfaction

The Productivity Model captures software quality in an objective way – as the number of defects, which can be objectively counted by testers or managers. It does not refer in any way to subjective quality – how future users will perceive software quality. Such a subjective assessment will also be based on the *number of defects*, but user satisfaction also captures the following:

- adequacy of the software to meet user needs – is the software doing what users expect from it,
- appearance of the interface – is the interface nice, user-friendly, easily navigable,
- performance – is the software performing its actions as fast as it should,

- level of training and quality of documentation.

The customer (company) who ordered the system typically also extend this list by adding the costs paid for the system and the project duration.

6.9 Summary

The new Productivity Model discussed in this chapter addresses the key limitations of existing BNs: problems in trade-off analysis and incorporating new empirical data. This model adopts the basic philosophy of existing models but is structurally different from them. We calibrated this model using the data gathered in a questionnaire survey. The next chapter discusses the model's potential in providing useful information for software managers.

7 Validating Productivity Model

This chapter focuses on validating the Productivity Model. Each section discusses a different independent hypothetical validation scenario. In these scenarios we check the outcome produced by the model and discuss how this outcome can be explained on the basis of our expert knowledge and existing state of knowledge in the domain. We also validate whether the model correctly incorporates known empirical data and can provide useful information for decision-support. This chapter is partially based on the previous work [206, 66].

7.1 *Constraints to external validation*

Although the validation performed was extensive it is impossible to perform a validation (for this or any similar model) against data from real software projects because the relevant data is not available. For example, the ISBSG dataset does not include qualitative process factors as required by the model. The dataset published in [68, 69] contains such process data but does not contain either effort allocation or descriptive project factors data. The extensive survey of other publicly available software project datasets [200] shows that the data required by our model is not published.

Besides, the Productivity Model needs to be calibrated for use in a particular company as described in the previous chapter. This calibration can only be conducted properly from within a single company. This ensures that users within that company understand the model and that their expert knowledge is properly represented in the model. This further reinforces the need to use data from a single company if validation against a dataset is to be performed.

As a result we perform an internal validation against existing knowledge where we can discuss the directions of changes in estimates as a result of some input. We validate the model's precision by comparing the predicted results with the data gathered in the questionnaire survey.

Our model provides predictions in a form of probability distributions for each variable. When we compare the different scenarios analysed in this section we frequently refer to median values of these distributions. Distributions for numeric nodes

in most of the analyzed scenarios are heavily skewed to the right (to positive infinity) and the mean value of such distributions would not appropriately describe such nodes.

7.2 Non-default productivity and defect rates

The MODIST Project-level Model incorporates trade-off relationships without the ability for users to specify their own prior productivity and defect rates. The new Productivity Model enables users to enter their own productivity and defect rates, which they can estimate outside the model.

For example, suppose we have a development team of 10 people full time for 24 months. If we do not provide any other information, the MODIST model predicts that from these resources we could deliver around 1500 FPs. If our process and people are at the highest possible level, then we should be able to produce software of 6000 FPs. That gives productivity rates of 6.25 and 25 FPs per person-month respectively.

However, to fully understand the limitations of the MODIST model, suppose further that, because of the special nature of our recent projects, we achieve typically a much higher productivity rate of 30 function points per person-month. Because this information is so different from the built-in priors of the MODIST model the predictions will be severely underestimated.

In contrast, the Productivity Model enables us to enter custom prior productivity and defect rates directly as observations. We enter an observation for *prior productivity rate* of 30 FPs per person-month. With these assumptions the model estimates the probability distributions for all other variables for which no observation was entered. Here the new model predicts *revised productivity rate* to be around 29 FPs per person-month. The decrease from prior to revised productivity rate is caused by other uncertain factors (more in Section 7.7). With such a productivity rate, from 240 person-months of effort we should be able to deliver around 6830 FPs. With the highest possible increase in *process and people quality* expected *delivered number of units* will shift to around 10160 FPs.

Let us further assume that this company delivers a specific type of software and the development process leads to a situation where they achieve ‘non-standard’ quality in terms of number of defects per unit of size (defect rate). Again, the MODIST model does not enable us to use such information. With the Productivity Model it is possible to enter a custom defect rate as an observation and analyze how this affects predictions for

other variables. For example, assume that in previous similar projects they achieved a defect rate of 0.04 defects per FP. We enter this value in the model as *prior defect rate*. The model predicts around 300 defects post-release. Now we increase process and people quality in all development activities to the highest level. This results in *revised defect rate* being 0.028 and an estimated 280 defects post-release.

The capability of being able to enter custom prior productivity and defect rates is one of the most important features of the new model. This feature is used in most scenarios analyzed in this chapter.

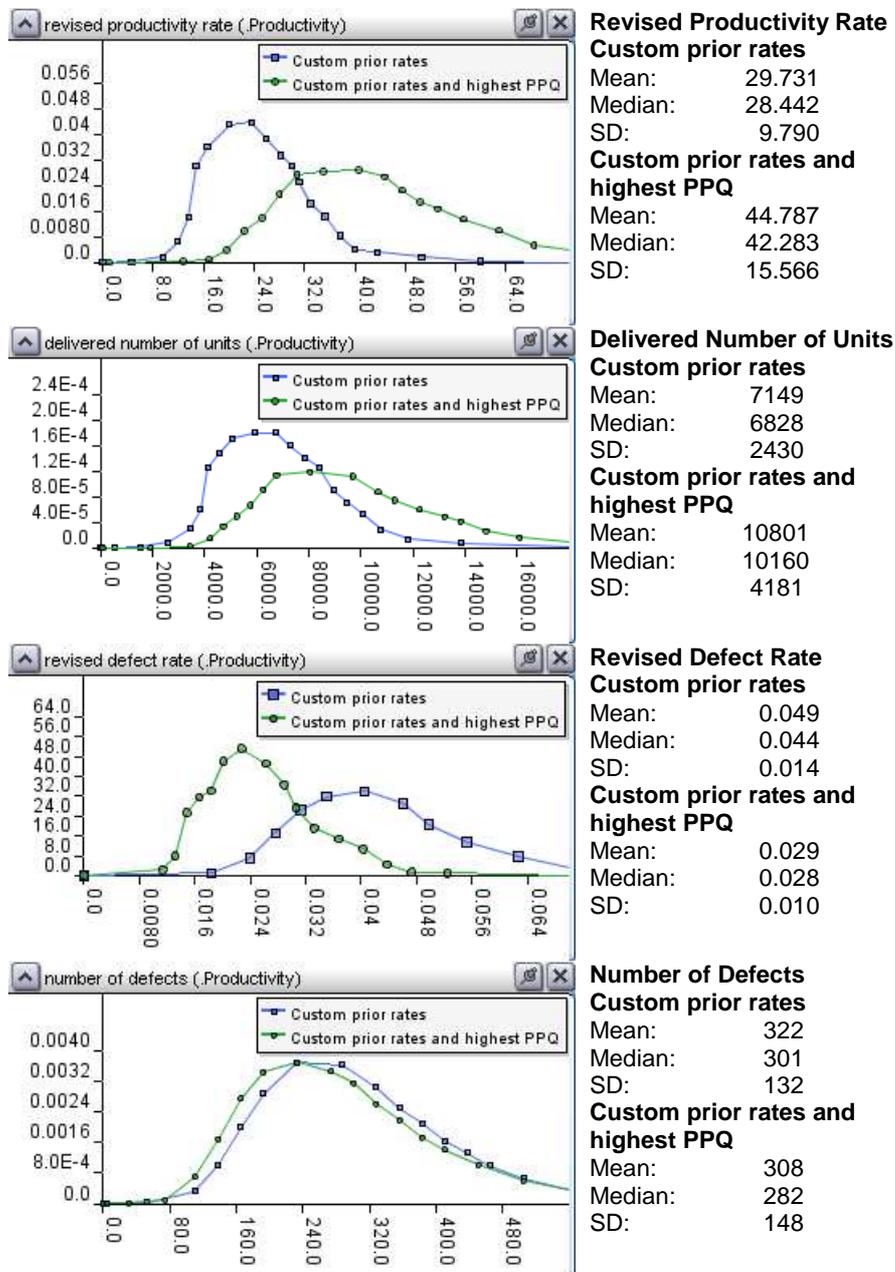


Figure 7-1 Predictions with custom prior rates

7.3 Meeting the target for product quality

In the Productivity Model we can perform multi-step ‘what-if’ analyses where we add new observations to the model in subsequent scenarios as they appear. The model contains the key trade-off variables expressed on a numerical scale as opposed to the previous models where some of them were on a ranked scale. That gives the ability to get predictions expressed as numbers (probability density functions to be exact) for all of these variables.

Let us suppose that a software company has to deliver software of 500 function points but constrained to 2500 person-hours of effort. Suppose that in similar projects in the past the defect rate was typically 0.15 defects per function point, productivity rate was 0.2 function points per person-hour and the effort was 2000 person-hours. For a fair comparison in these examples we assume that all other factors included in the model are the same in all scenarios. With this information entered in the model, it predicts the number of defects delivered will be around 79 defects (Figure 7-2). Suppose managers decide that this number of defects is unacceptably high and they wish to know how they can improve it to, say, just 40 defects. The model predicts that to achieve this quality target a better development process and more productive people are required. Also, allocating more effort on specification and testing at the cost of effort on coding is required. The lower effort on coding is balanced by the better coding process and the ability of more productive people to deliver the assumed product size.

Now let us further assume that the company is not able to improve the process and people for this project. Thus we need to perform trade-off analysis between key project variables. We might remove a constraint for the product size. This would result in predicting lower product size containing lower total number of defects. But sacrificing product size would rarely be a good solution. We rather analyze how much more effort is required to achieve the target for the number of defects. The model now predicts that this company should spend around 7215 person-hours on this project to achieve the lower number of residual defects after release. The majority of the increased effort should be allocated to specification and testing activities.

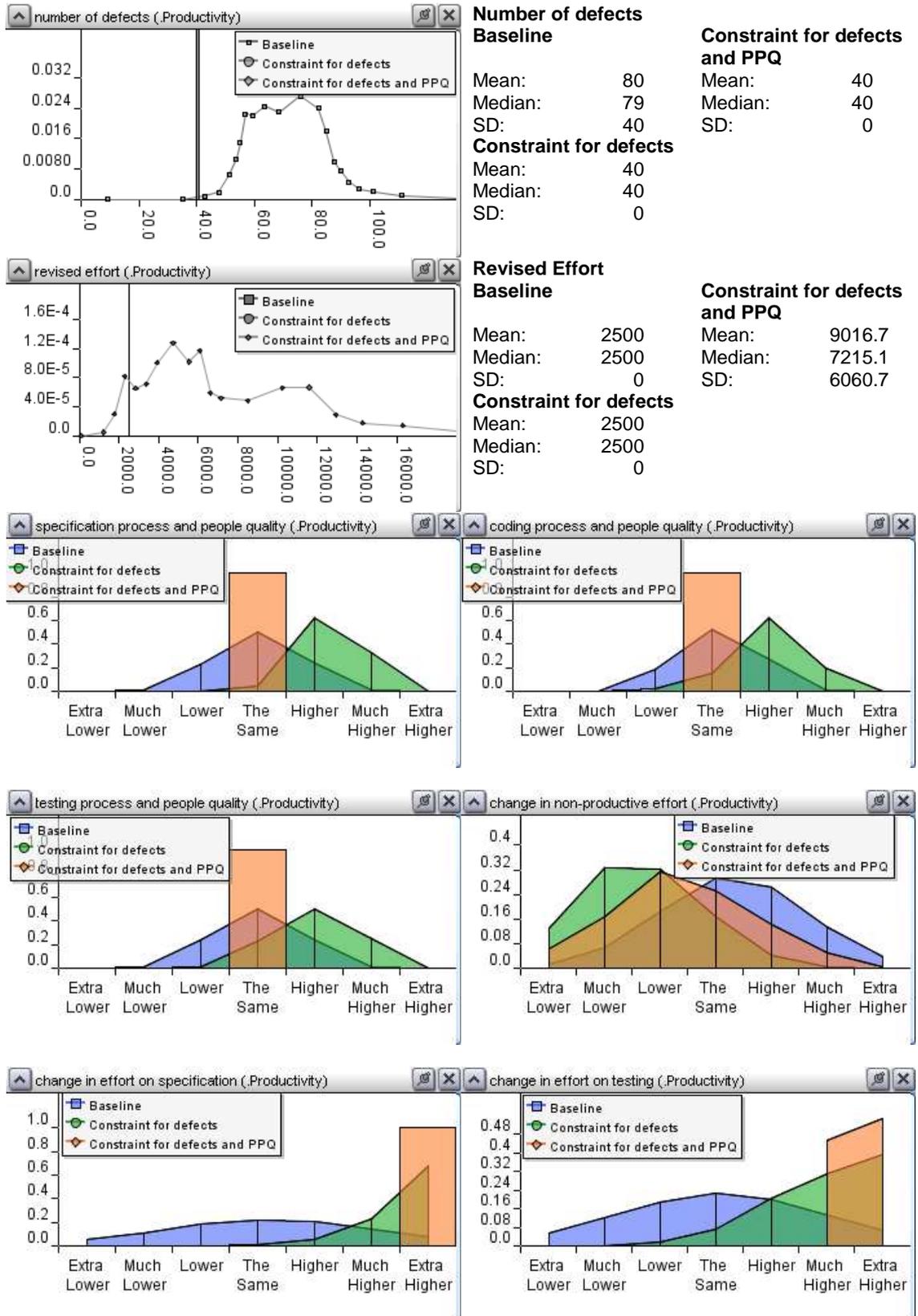


Figure 7-2 Meeting the target for product quality

7.4 Impact of uncontrollable project factors

In our Productivity Model we can analyze the impact of different uncontrollable project variables on the team productivity and the product quality. Suppose that the size of the software to be developed is 1000 FPs. We leave the prior effort, productivity and defect rates at their default states. We further assume that *project complexity* increased by the highest possible value – which means by as high as it has been observed before in the company in the past most complex projects. In this case the model predicts that we are likely to achieve a lower *revised productivity rate* (0.161 FP/person-hour) and that the developed software will be of lower quality (higher *revised defect rate* with median=0.062 defects/FP) compared to the scenario which assumes no change in *project complexity* (Figure 7-3 – ‘Complex project’ scenario).

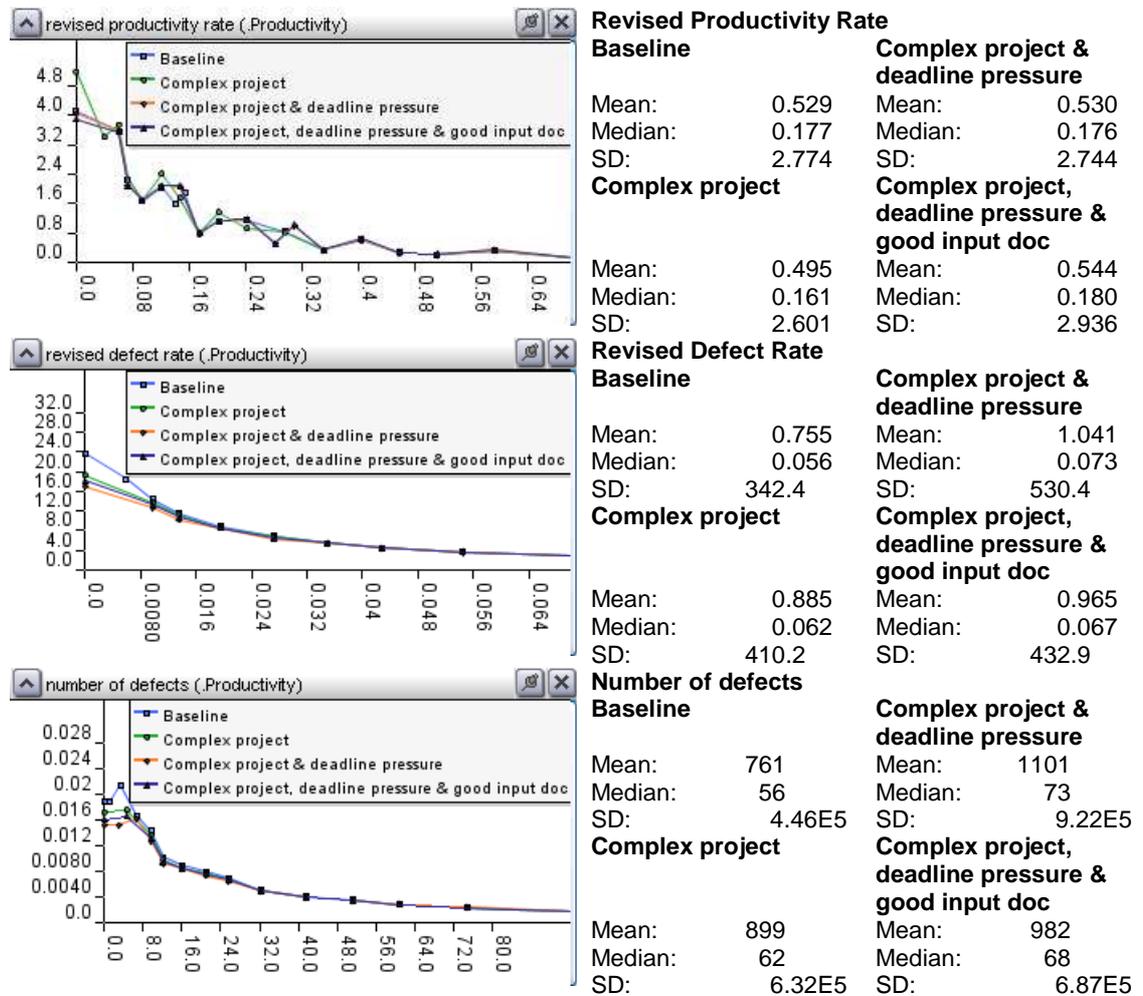


Figure 7-3 Predictions with changed uncontrollable factors

7.5 **Changed effort allocation**

In the Productivity Model we can define the allocation of effort spent on different development activities: specification, coding, testing and the effort which is non-productive.

For example, suppose our aim is to predict the functionality and the quality of the developed software for some given effort, e.g. 1000 person-hours, which is the same amount of effort as we spent in similar projects in the past. With the default probability distributions for prior productivity and defect rate the model predicts that the company should deliver about 104 FPs, the number of residual defects should be around 6 and the defect rate should be of around 0.054 defects per function point. (Figure 7-5 – ‘Baseline’ scenario).

Let us now assume that the same total amount of effort will be allocated in a different way. More effort will be spent on coding which will automatically lead to spending less effort on specification and testing. As we should expect the model predicts a higher *delivered number of units* (123 FPs) but with a higher *number of defects* (7) and a higher *revised defect rate* (0.060 defects per FP) as illustrated on Figure 7-5 (‘More effort on coding’ scenario). This means that the model properly captures the impact of effort allocation on productivity and defect rates which in turn influence the delivered functionality and number of defects.

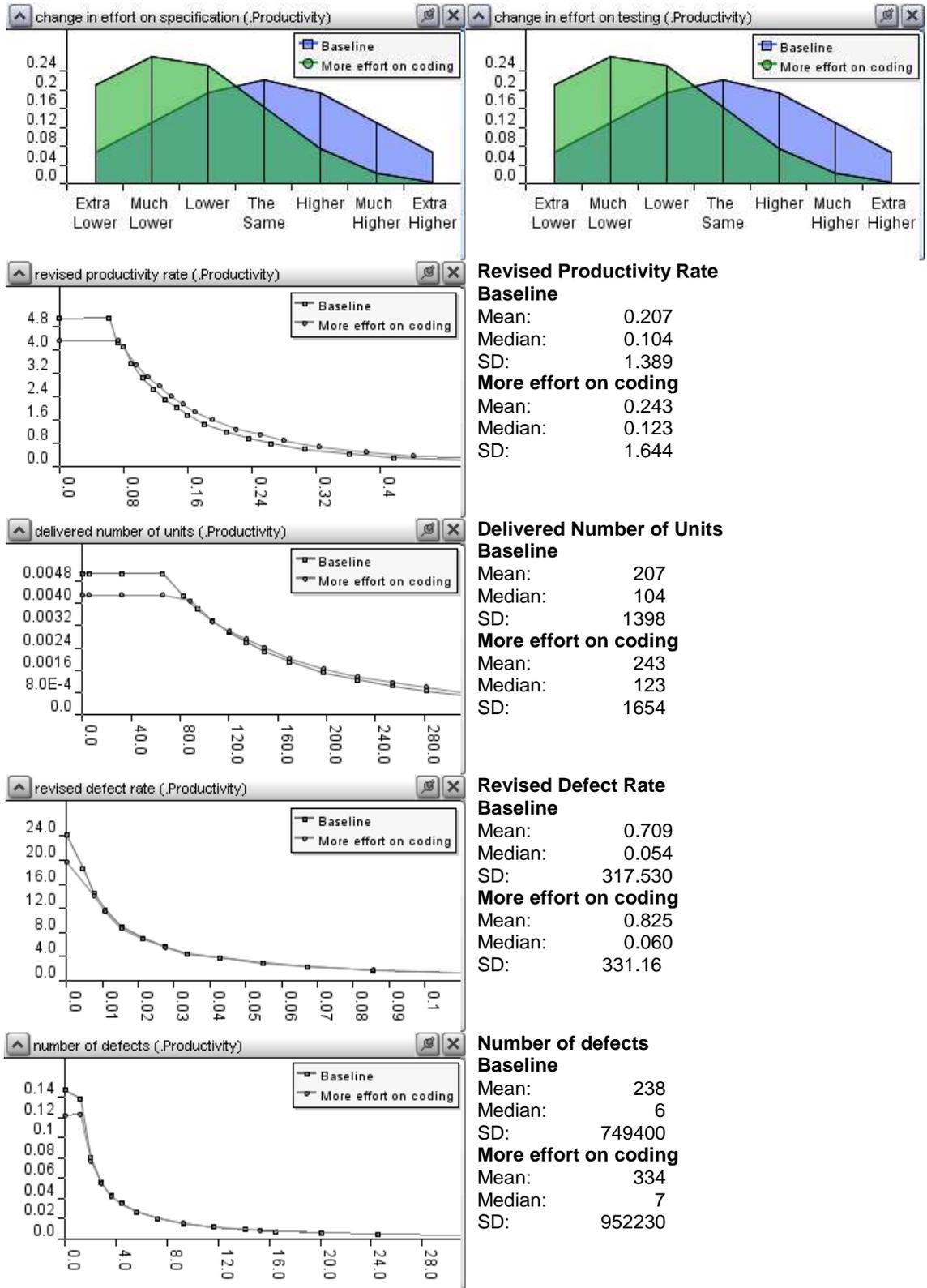


Figure 7-5 Predictions with default and changed effort allocation

7.6 **Explaining counterintuitive outcome**

When analyzing if the Productivity Model captures known relationships between key project variables (trade-off analysis) we set the other variables to fixed states. However, we can relax this condition to analyze some counterintuitive outcomes. Let us assume that a company can spend 500 person-hours delivering new software. By entering this information into the model we get a prediction for *delivered number of units* of around 48 FPs and a *defect rate* of 0.055 defects per FP. Normally increases in both the process and people quality, and the development effort should lead to an increase in at least one of: functionality or software quality.

But let us assume that, in contrast to what we would normally expect, we observe lower delivered functionality (35 FPs) and poorer quality (0.08 defects per FP). The Productivity Model can give us the explanation for this because it contains other variables which influence the trade-off relationships. The most likely explanations in this case, as given by the model, are an increase in *project complexity*, *project novelty* and *project scale* (the latter indicating the need to spend more effort on non-productive activities such as infrastructure), and a decrease in *positive customer involvement*, *quality of input documentation* and *change in effort on coding* as illustrated on Figure 7-6. This shows that the model can give explanations of some ‘strange’ situations occurring in real projects.

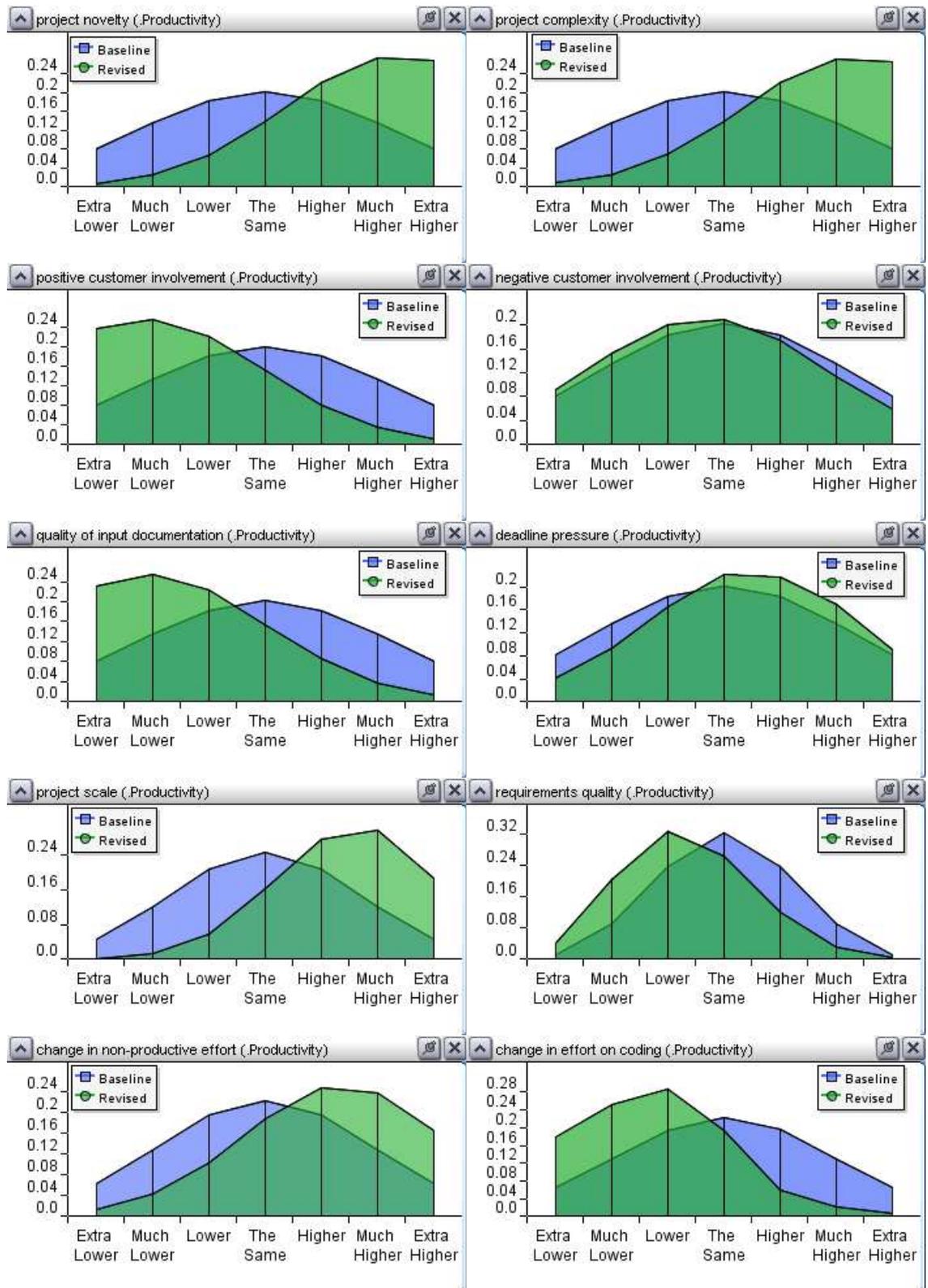


Figure 7-6 Predictions for variables influencing trade-off relationships

7.7 Influence of uncertainty in qualitative factors

In some of the cases analyzed above we assumed that qualitative factors were left at their default settings whereas in others we assumed that these factors were at ‘the same’ level. The model gives different results in these cases. The default probability distribution assumes that all of the states for such variables are possible. The most probable is ‘the same’ and the more a particular state is different from ‘the same’ the less probable it is. Setting an observation to state ‘the same’ is different from leaving it without any observation because it assumes that this ‘the same’ state is 100% probable and all other states are impossible in the given scenario. This spread in default probability distributions causes different predictions. With the default probabilities for qualitative factors the model predicts a lower *revised productivity rate* and higher *revised defect rate* than with qualitative factors set to ‘the same’ state. Figure 7-7 illustrates the discussed differences in prediction results.

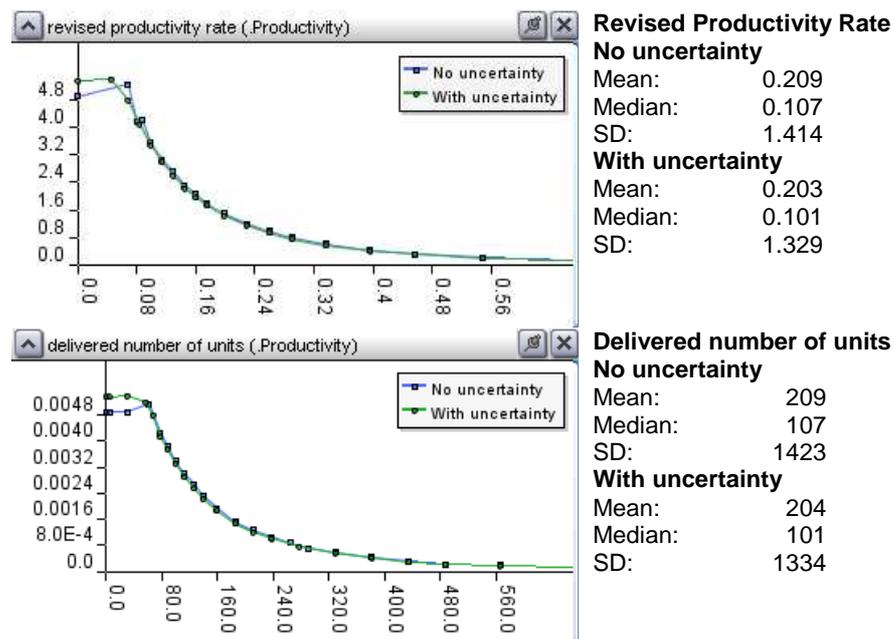


Figure 7-7 Predictions for default and most probable scenarios

7.8 Influence of controllable and uncontrollable factors

In this case we analyze the impact of various factors on *revised productivity rate* and on *revised defect rate* in the most and least favourable scenarios. The most favourable scenario is the one which leads to the highest possible increase in *revised productivity rate* and the greatest possible decrease in *revised defect rate*. The least

favourable scenario leads to the greatest possible decrease in *revised productivity rate* and highest possible increase in *revised defect rate*. In this analysis we verify whether the spread for these two dependent variables reflects respondents' opinions provided in the questionnaire survey during model calibration.

Table 7-1 show various combinations of controllable and uncontrollable factors in the most and least favourable scenarios for productivity and defect rates. There are different combinations for productivity and defect rate. Some factors are positively correlated with both productivity and defect rate. Some are positively correlated with productivity rate but negatively correlated with defect rate.

Table 7-1 Combinations of factors in most and least favourable scenarios for productivity and defect rate

| Predictor \ Dependent & scenario | Productivity rate | | Defect rate | |
|--|-------------------|-----------------|------------------|-----------------|
| | Least favourable | Most favourable | Least favourable | Most favourable |
| Deadline pressure | extra lower | extra higher | extra higher | extra lower |
| Negative customer involvement | extra higher | extra lower | extra lower | extra higher |
| Positive customer involvement | extra lower | extra higher | extra lower | extra higher |
| Project complexity | extra higher | extra lower | extra higher | extra lower |
| Project novelty | extra higher | extra lower | extra higher | extra lower |
| Project scale | extra higher | extra lower | extra higher | extra lower |
| Quality of input documentation | extra lower | extra higher | extra lower | extra higher |
| Requirements quality | extra lower | extra higher | extra lower | extra higher |
| Change in non-productive effort | extra higher | extra lower | extra higher | extra lower |
| Change in effort on specification | extra lower | extra higher | extra lower | extra higher |
| Specification process and people quality | extra lower | extra higher | extra lower | extra higher |
| Change in effort on coding | extra lower | extra higher | extra higher | extra lower |
| Coding process and people quality | extra lower | extra higher | extra lower | extra higher |
| Change in effort on testing | – | – | extra lower | extra higher |
| Testing process and people quality | – | – | extra lower | extra higher |

Figure 7-8 illustrates predictions provided for *revised productivity rate* and *revised defect rate* in the least favourable and most favourable scenarios. We can observe that the *revised productivity rate* is 13.7 times higher in the most favourable scenario compared to the least favourable scenario. The *revised defect rate* is 17.7 times higher in the least favourable scenario compared to the most favourable scenario. This spread correctly reflects the respondents' opinions obtained during the questionnaire survey (results summarized in Table 6-2).

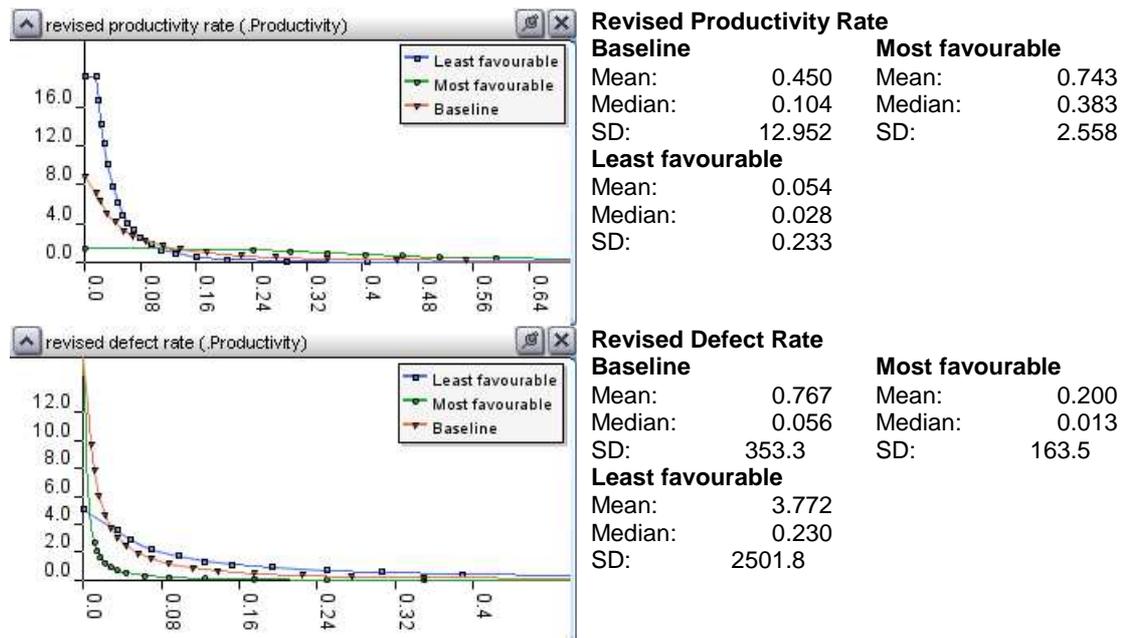


Figure 7-8 Predictions for most and least favourable scenarios of controllable and uncontrollable factors

In the above example we assumed that in the ‘least favourable’ scenario both controllable and uncontrollable factors were least favourable while in the ‘most favourable’ scenario both controllable and uncontrollable factors were most favourable. Now we analyze the predicted productivity and defect rates in two other combinations of controllable and uncontrollable factors. The first of these scenarios assumes that controllable factors are most favourable and uncontrollable factors are least favourable. The second scenario assumes that controllable factors are least favourable and uncontrollable factors are most favourable. Figure 7-9 illustrates the predictions in these scenarios. We can observe that there are no big differences in these two scenarios. However, in the first scenario *revised productivity rate* is higher and *revised defect rate* is lower than in the second scenario. Furthermore, in both of these scenarios *revised productivity rate* is lower and *revised defect rate* is higher than in the ‘baseline’ scenario (without any observations entered to the controllable and uncontrollable factors).

The analyses performed in this case study leads to the following conclusions which confirm that the model properly captures the results of the questionnaire survey used to calibrate the model:

- if uncontrollable (or controllable) factors are least favourable it is not possible to mitigate them by changing the most favourable controllable (or uncontrollable) factors;
- the impact of controllable factors is stronger than uncontrollable factors.

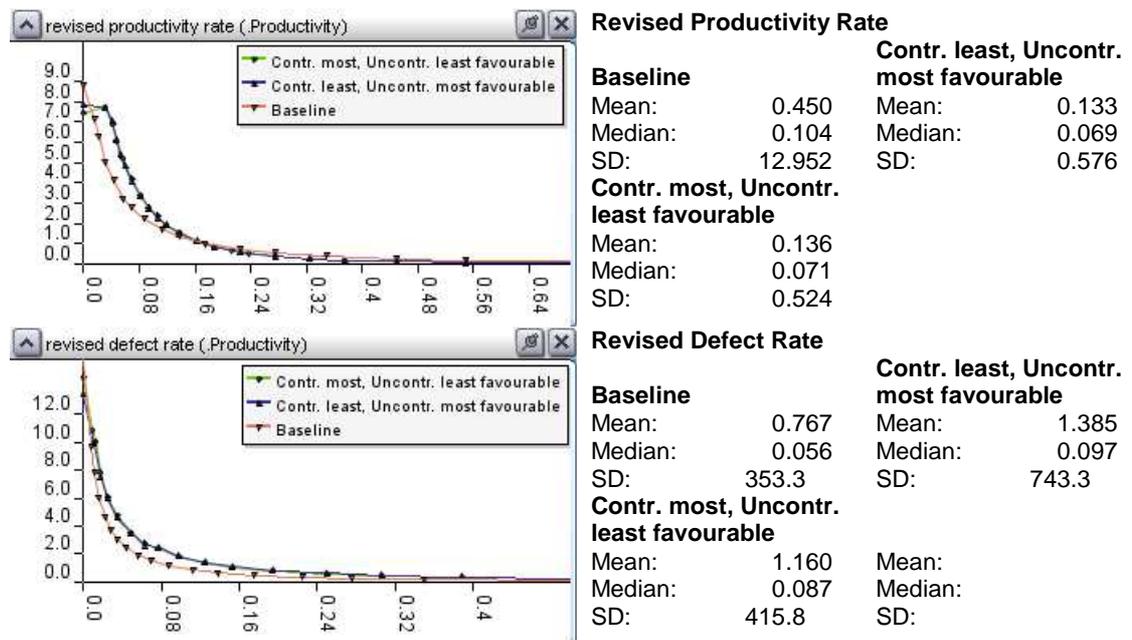


Figure 7-9 Predictions for different combinations of controllable and uncontrollable factors

7.9 Using custom units of measurement

The Productivity Model can use different units of measurement for entering observations and obtaining predictions. Let us suppose that users want the model to predict the effort required to deliver a specific amount of functionality and meet a specific constraint about delivered quality. Furthermore, suppose they want this functionality to be expressed in a non-standard unit of measurement: ‘number of requirements’. The MODIST model cannot be used in this case because functionality can only be directly expressed in function points or indirectly expressed in KLOC. But the Productivity Model enables users to choose appropriate units of measurement for effort, defect rate and productivity rate. The model supports the following predefined lists of units of measurement:

- for effort: ‘person-month’, ‘person-week’, ‘person-day’, ‘person-hour’;
- for defect rate: ‘defects/FP’, ‘defects/KLOC’;
- for productivity rate: ‘FPs/person-hour’, ‘FPs/person-day’, ‘FPs/person-week’, ‘FPs/person-month’, ‘KLOC/person-hour’, ‘KLOC/person-day’, ‘KLOC/person-week’, ‘KLOC/person-month’, ‘Classes/person-hour’, ‘Classes/person-day’, ‘Classes/person-week’, ‘Classes/person-month’.

For each of these variables users can also choose ‘other’ units of measurement, for example they can choose to represent effort in ‘number of requirements per person-

week' or 'number of requirements per programming-group-week' if such non-standard units are used in their company.

If users wish to use units of measurement of their choice ('other') they need to enter prior values as observations for variables expressed in those custom units of measurement. Entering these priors is necessary because choosing 'other' unit of measurement for any numeric node (which has the ability to be switched to other units of measurement) means that estimates of these numeric nodes are meaningless. For example, if the user selects 'other' as a unit of measurement for effort, *prior effort* will be reflected by the distribution: Normal(0, 1E20), which says that it is very uncertain what the actual *prior effort* is. That is because the model cannot anticipate the meaning of 'other' units of measurement in relation to existing units supported by the model.

Going back to our example, we assume that we would like to get predictions for functionality measured in 'number of requirements' and effort in 'person-weeks'. This causes the productivity rate to be measured in 'number of requirements per person-weeks' and defect rate in 'defects per requirement'. Because the model does not directly support these units we apply observations to appropriate variables: 'person-weeks' to effort and 'other' to productivity and defect rates. We assume that in most similar past projects we spent 20 'person-weeks' of effort, achieved a productivity rate of 5 'requirements per person-week' and a defect rate of 0.1 'defects per requirement'. The final assumption is to deliver 70 requirements.

Figure 7-10 illustrates the predictions for this case ('Baseline scenario'). The model predicts that given these constraints about 13.6 person-weeks of effort needs to be spent on this project. Let us further assume that in addition there is a quality target for *revised defect rate* which is 0.05 defects per requirement. The model then predicts that more effort needs to be spent on the project (median of 21.9 person-weeks).

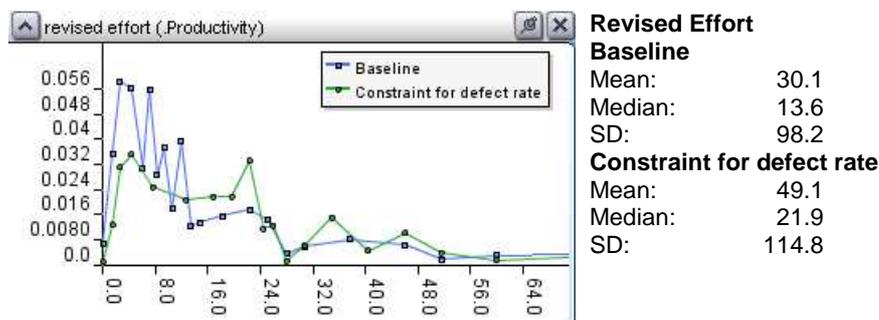


Figure 7-10 Predictions with custom units of measurement

These results confirm that the Productivity Model supports several units of measurement for numeric variables. Predictions, including trade-off analysis can also be performed for variables expressed in custom units.

7.10 Setting interval target for numeric node

In all previous cases we assumed that entering an observation to a numeric node meant entering a point value. However, it is also possible to enter observations which are intervals. There are 2 ways to achieve this:

1. Adjusting a node to a fixed discretisation.
2. Adding a new Boolean node holding the interval observation data.

We illustrate both of them with the same hypothetical scenario. Let us assume that a company is about to deliver software of 1000 FPs. In the past they developed similar software with 4000 person-hours, achieving a productivity rate of 0.1 FPs per person-hour and a defect rate of 0.08 defects per FP. For simplicity we assume the same level of all qualitative factors. The question to be answered is: “how do we achieve a defect rate within a range from 0.05 to 0.08 defects per FP?”

The first approach assumes that the numeric node (*revised defect rate*) will have a fixed discretisation. We need to define an interval with boundaries set equal to the interval that we want to set as an observation. Since the defect rate ranges from 0 to positive infinity we need to define 3 intervals:

- 0-0.05,
- 0.05-0.08,
- 0.08-infinity.

Then we need to apply hard evidence to the second of these intervals. After entering observations to other nodes (depending on the constraints for the scenario) we run the model. Figure 7-11 illustrates predictions provided by our model. The model predicts the need for a very significant increase in development effort compared to the similar past project used as the base for these calculations.

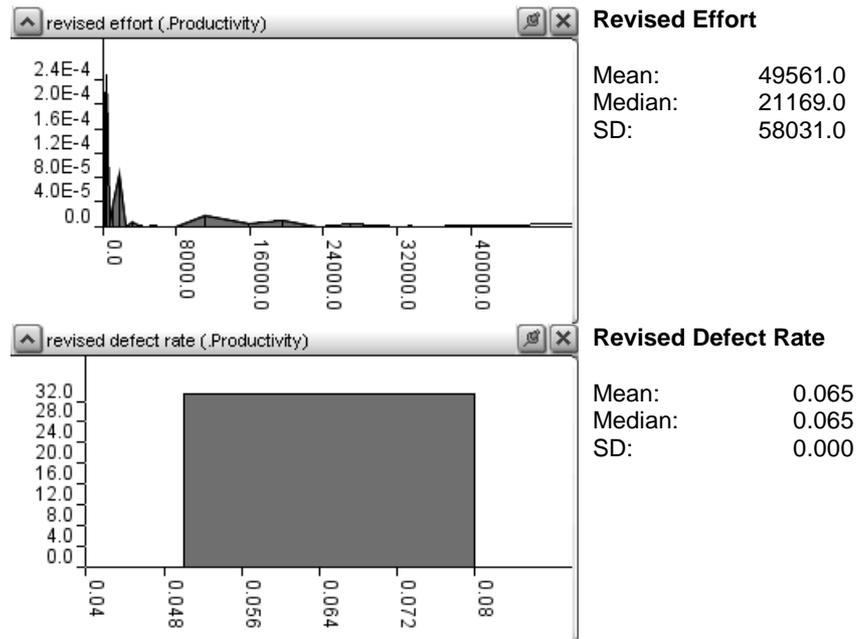


Figure 7-11 Predictions for scenario with interval target for *revised defect rate* (approach 1)

The second proposed solution requires adding three new Boolean nodes, but no changes are required to the numeric node for which we wish to enter an interval observation. We add Boolean nodes defined as shown in Equations 22-24.

$$d_1 = \text{if } (d_{\text{revised}} \geq 0.5, \text{"Yes"}, \text{"No"}) \quad (22)$$

$$d_2 = \text{if } (d_{\text{revised}} \leq 0.8, \text{"Yes"}, \text{"No"}) \quad (23)$$

$$d_{\text{constraint}} = \text{if } ((d_1 = \text{"Yes"}) \ \&\& \ (d_2 = \text{"Yes"}), \text{"Yes"}, \text{"No"}) \quad (24)$$

The node $d_{\text{constraint}}$ says that if the *revised defect rate* is within the assumed boundaries then it has a value ‘Yes’, otherwise it has a value ‘No’. Then we set the observed state of $d_{\text{constraint}}$ to ‘Yes’ and after entering observations into other nodes we can run the model. This custom interval can also be parameterized by using constants. In this case the interval boundaries could be defined in the risk table view and they could be different for each scenario analyzed. A detailed discussion on extending the models using constants is contained in Section 5.3.

Figure 7-12 illustrates the model predictions with the second approach applied. Constraining the defect rate by adding three Boolean nodes gives different predictions for *revised effort* compared to the first approach. The model still predicts that we need significantly more effort than in the past project. However, in this solution this increase is much lower than using the first solution. This difference is caused by the fact that here the model does not estimate a uniform distribution for *revised defect rate*. Rather it

predicts that values closer to the upper bound of the target interval (0.08) are more probable than values closer to the lower bound of this interval (0.05).

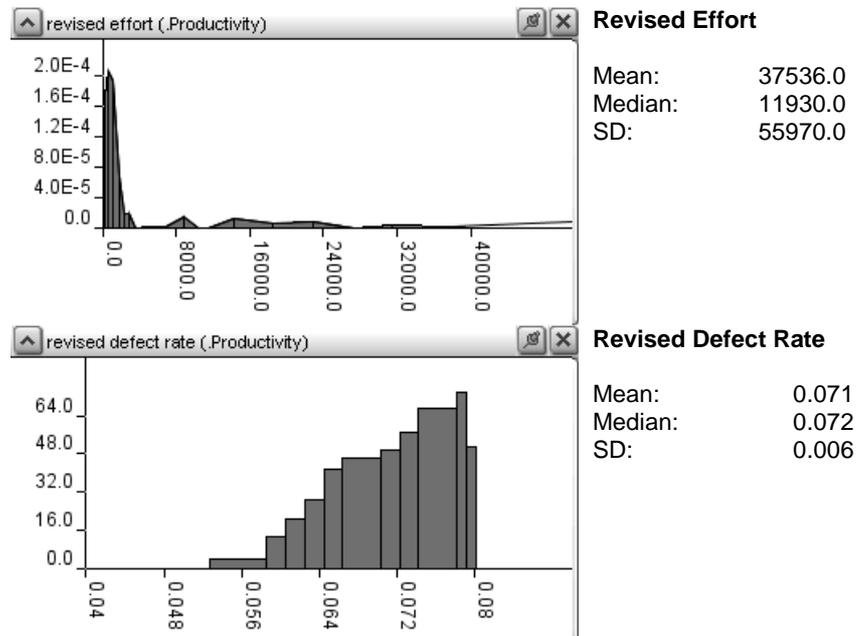


Figure 7-12 Predictions for scenario with interval target for *revised defect rate* (approach 2)

If the target interval should be uniformly distributed, the first solution should be used (with fixed discretisation). When the target interval should be non-uniformly distributed, i.e. reflect the prior distribution within this interval, the second solution should be used (by adding 3 Boolean nodes).

Generally these solutions to entering interval observations can be applied to any numeric node in our model. Especially useful might be:

- *revised effort*,
- *delivered number of units*,
- *number of defects*,
- *revised productivity rate*,
- *revised defects rate*.

7.11 Sensitivity analysis

In this sensitivity analysis we examine the impact of particular factors on dependent variables. The aim of this analysis is to determine whether the model properly captures the results from the questionnaire survey and how some unexpected results can be explained. For each predictor used in this analysis we examine its impact

on a dependent variable by assigning an observation to each of its states and running the model.

Figures 9-14 to 9-18 illustrate the results of the sensitivity analysis presented as tornado graphs (the longer bar for a particular predictor indicates its greater impact on a particular dependent variable). Generally, these results confirm that the Productivity Model correctly captures the questionnaire survey data. However, in the following cases we observed unexpectedly high or low predictor's impact:

- *documentation quality on coding effectiveness* (Figure 7-13),
- *project scale on revised productivity rate* (Figure 7-14),
- *change in effort on specification on revised productivity rate* (Figure 7-15),
- *change in effort on testing on revised productivity rate* (Figure 7-15),
- *testing process and people quality on revised productivity rate* (Figure 7-15),
- *change in effort on coding on revised defect rate* (Figure 7-15),
- *change in effort on testing on revised defect rate* (Figure 7-15),
- *testing effectiveness on revised productivity rate* (Figure 7-17),
- *documentation quality on revised productivity rate* (Figure 7-17).

The unexpected impact of these variables is caused by two factors:

1. In each scenario we enter an observation into only one predictor while keeping the other variables with their default probability distributions.
2. The relationships between effort allocation nodes and their impact on development activity effectiveness (as discussed in Section 6.4 on page 109).

In some cases entering an observation into one of the predictors causes a change in the posterior probability distribution for other variables which also have an influence on the dependent variable. For example, the lower than expected impact of *documentation quality on coding effectiveness* is caused by the fact that *change in effort on coding* influences *documentation quality*. In the scenario assuming 'extra lower' *documentation quality* the model explains that such low *documentation quality* is partially caused by the fact that significantly less effort is allocated to specification and significantly more on coding (indicating a larger project). So, decreased *documentation quality* is balanced by increased *change in effort on coding*. As a result the model predicts that *coding effectiveness* changes very little according to changes in *documentation quality*.

On the other hand, with the ‘extra higher’ *documentation quality* the model believes that it is partially a result of decreased *change in effort on coding* (meaning a smaller project). In this case increased *documentation quality* is balanced by decreased *change in effort on coding*. Thus, the impact of *documentation quality* on coding effectiveness is again very low.

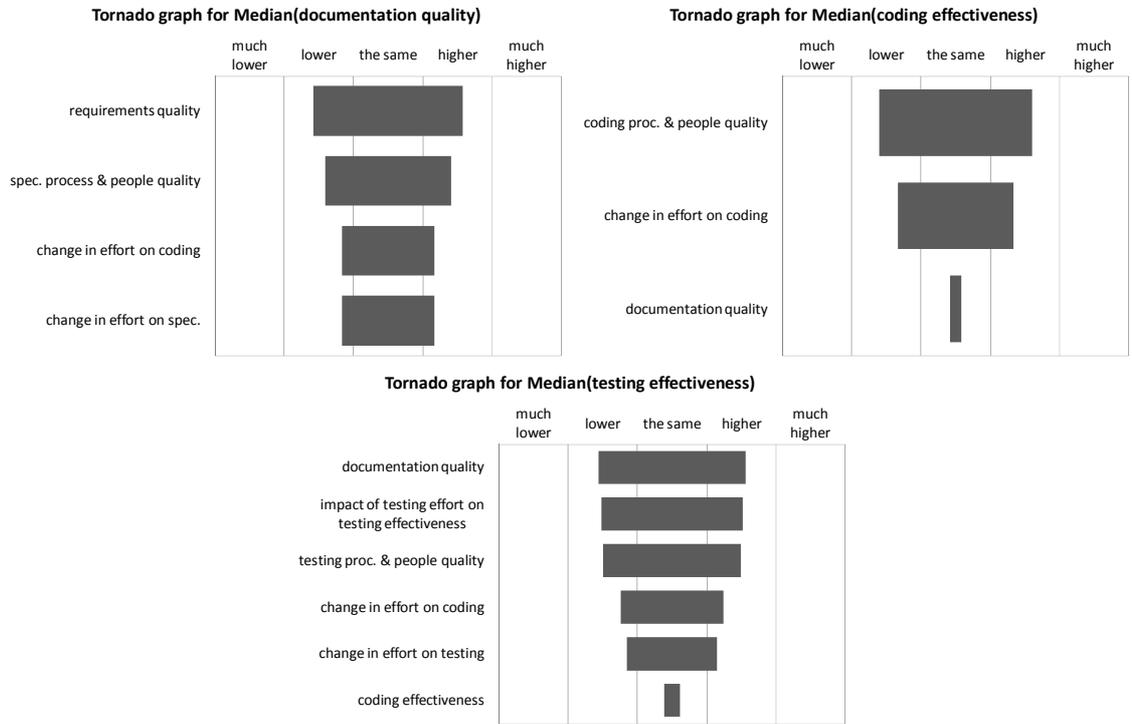


Figure 7-13 Impact of factors on process effectiveness in different development activities

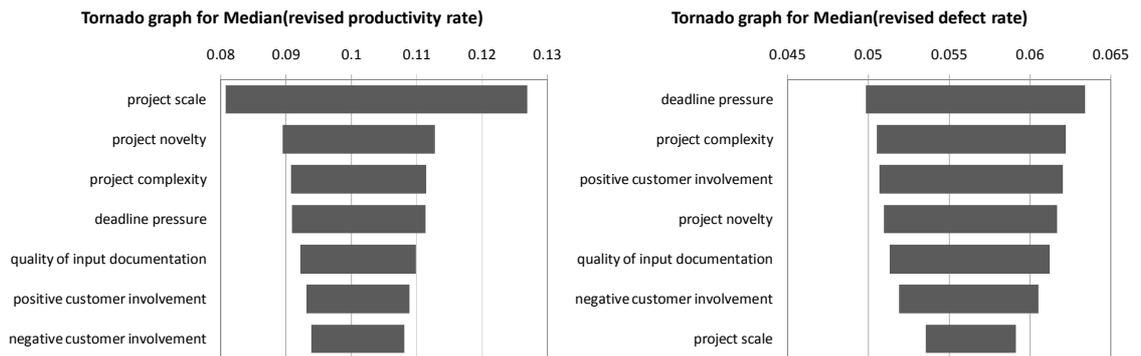


Figure 7-14 Impact of uncontrollable project factors on productivity and defect rates

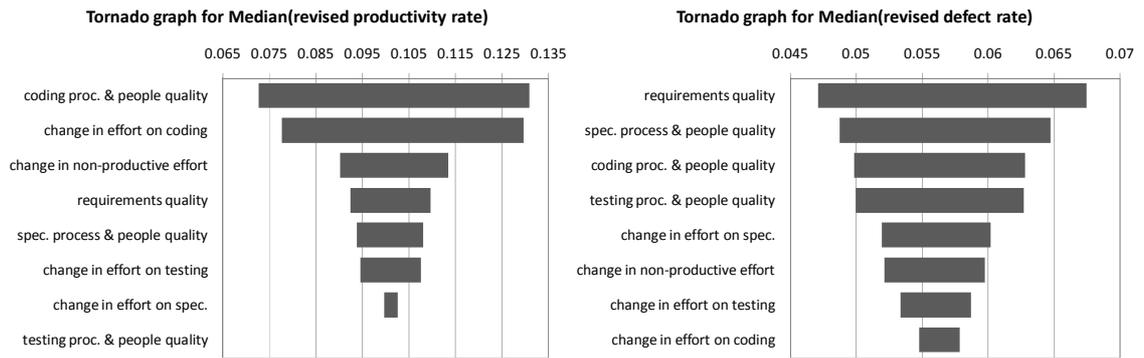


Figure 7-15 Impact of process factors on productivity and defect rates

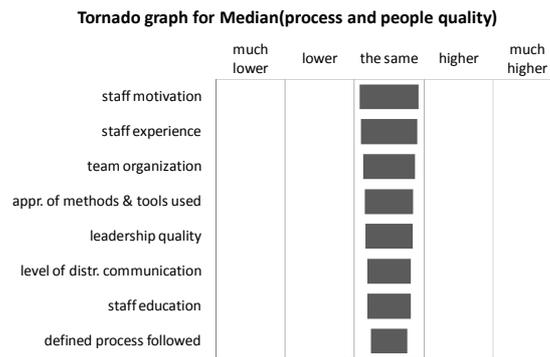


Figure 7-16 Impact of individual process and people factors on overall *process and people quality*

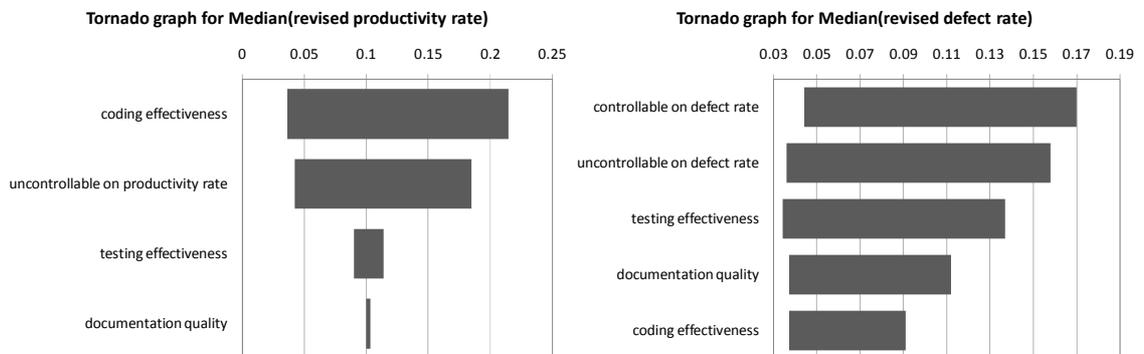


Figure 7-17 Impact of aggregated controllable and uncontrollable factors on productivity and defect rates

7.12 Summary

The validation performed on the Productivity Model showed its great potential in providing useful information for project risk assessment. The model provides reasonable explanations in various scenarios. In most of the scenarios discussed we assumed that users are able to provide productivity and defect rates as well as effort from the most similar past projects. This model can still be used even when such data is

not available thanks to the estimates for productivity and defect rates provided by the PDR model discussed in the next chapter.

8 Modelling prior defect and productivity rates

The Productivity Model assumes that managers can provide prior productivity and defect rates based on some past projects. But often such data is not easily available. Therefore, this chapter focuses on extending the Productivity Model with a sub-model able to estimate such prior rates. This study discusses the steps and results of statistical analysis aimed to determine project descriptors influencing the productivity and defect rates. The model for estimating the productivity and defect rates (PDR model) uses the results of this statistical analysis adjusted by other data sources and expert judgement. The new contributions of this chapter are the statistical analysis of productivity and defect rates' factors and a new PDR model. This chapter is based on [199, 203, 66]. Results of the statistical analysis come from the STATISTICA 8 tool [234].

8.1 The dataset

In this analysis we have used the ISBSG dataset [118] which contains data about 3024 projects described by 99 variables. The newer edition of the dataset [121] contains more data (both more observations and more variables) but we did not have an access to this edition. We filtered the dataset to leave only those projects for which *data quality* was classified as 'A' or 'B' – meaning respectively 'very good' and 'good' quality as suggested in [120]. In the analysis with productivity rate as the dependent variable we further filtered the dataset to ensure that productivity rate is calculated in the same way for all projects considered and used only these projects where:

- *count approach* was 'IFPUG' – to measure project size in FPs and
- *resource level* was '1' – only development team effort was included in effort data.

As a result of such filtering we got two datasets containing:

- 2095 projects to analyze productivity rate,
- 510 projects to analyze defect rate (fewer here since in many projects no defect information was available).

None of the dependent variables were provided in the dataset directly. We added them and defined as follows (Equations 25-26).

$$productivity\ rate = \frac{functional\ size}{summary\ work\ effort} \quad (25)$$

$$defect\ rate = \frac{total\ defects\ delivered}{functional\ size} \quad (26)$$

8.2 Statistical analysis of the dataset

The aim of this statistical analysis was to identify predictors which could be used for estimating productivity and defect rates. We performed this analysis in 5 steps summarized below. Section C.1 contains additional tables and figures illustrating the results of this statistical analysis.

1. Analyzing dependent variables and applying transformations

First, we analyzed the distributions for the dependent variables. Without any transformation the distributions of the dependent variables were extremely non-symmetrical and dispersed. We applied the Ln (natural logarithm) transformations to the data to bring their distributions closer to normal distributions as suggested in [4]. In some cases the defect rate was '0' which cannot be transformed using Ln. Therefore, we performed the Ln transformation in 3 steps:

1. When total number of defects in a project was '0' we replaced this value by a small positive number: '0.1'.
2. We calculated adjusted defect rate using the adjusted total number of defects.
3. We transformed the adjusted defect rate with the Ln function.

Such an approach has also been applied in [5].

Table 1 illustrates summary statistics for original and transformed dependent variables.

Table 8-1 Summary of descriptive statistics for dependant variables

| Variable \ Statistic | Mean | Median | Std. dev. | Lower Quartile | Upper Quartile | Skewness | Kurtosis |
|----------------------|--------|--------|-----------|----------------|----------------|----------|----------|
| Productivity Rate | 0.254 | 0.114 | 1.114 | 0.057 | 0.226 | 26.639 | 903.37 |
| Defect Rate | 0.061 | 0.009 | 0.251 | 0.000 | 0.033 | 11.471 | 166.60 |
| Ln Productivity Rate | -2.175 | -2.170 | 1.112 | -2.873 | -1.488 | 0.207 | 1.26 |
| Ln Defect Rate | -4.819 | -4.571 | 2.116 | -6.273 | -3.424 | -0.088 | -0.32 |

2. Preparing list of potential predictors

Based on our experience we nominated a set of possible predictors (Table 8-2) which were listed in the ISBSG dataset to be used later in the model. We took descriptions for most of these potential predictors from [122].

Table 8-2 Descriptions of potential predictors used to build the model

| Name | Type | Description |
|------------------------------|------------|---|
| Average Team Size | Continuous | The average number of people that worked on the project, (calculated from the team sizes per phase). |
| Project Elapsed Time | Continuous | Total elapsed time for the project in calendar months. |
| Functional Size | Continuous | The unadjusted function point count (before any adjustment by a Value Adjustment Factor if used). |
| Summary Work Effort | Continuous | Provides the total effort in hours recorded against the project. |
| Value Adjustment Factor | Continuous | The adjustment to the function points, applied by the project submitter, that takes into account various technical and quality characteristics e.g.: data communications, end user efficiency etc. This data is not reported for some projects, (i.e. it equals 1). |
| Application Type | Nominal | This identifies the type of application being addressed by the project. (e.g.: information system, transaction/production system, process control.) |
| Architecture | Nominal | A derived attribute for the project to indicate if the application is Stand alone, Multi-tier, Client server, or Multi-tier with web public interface. |
| Business Area Type | Nominal | This identifies the type of business area being addressed by the project where this is different to the organisation type. (e.g.: Manufacturing, Personnel, Finance). |
| CASE Tool Used | Nominal | Whether the project used any CASE tool. |
| Client-Server | Nominal | Indicator of whether the application or product requires more than one computer to operate different components or parts of it. |
| Development Platform | Nominal | Defines the primary development platform, (as determined by the operating system used). Each project is classified as: PC, Mid Range, Main Frame or Multi platform. |
| Development Type | Nominal | This field describes whether the development was a new development, enhancement or re-development. |
| How Methodology Acquired | Nominal | Describes whether the development methodology was purchased or developed in-house, or a combination of these. |
| Intended Market | Nominal | This field describes the relationship between the project's customer, end users and development team. |
| Language Type | Nominal | Defines the language type used for the project: e.g. 3GL, 4GL, Application Generator etc. |
| Organisation Type | Nominal | This identifies the type of organisation that submitted the project. (e.g.: Banking, Manufacturing, Retail). |
| Package Customisation | Nominal | This indicates whether the project was a package customisation. |
| Primary Programming Language | Nominal | The primary language used for the development: JAVA, C++, PL/I, Natural, Cobol etc. |
| Used Methodology | Nominal | States whether a development methodology was used by the development team to build the software. |
| User Base – Business Units | Ordinal | Number of business units (or project business stakeholders) serviced by the software application. |
| User Base – Concurrent Users | Ordinal | Number of users using the system concurrently. |
| User Base – Locations | Ordinal | Number of physical locations being serviced/supported by the installed software application. |

3. Analyzing categories of potential nominal predictors

In this step we analyzed the frequencies for nominal predictors. For some categories there were very low numbers of observations (below 10). To improve the predictor performance we grouped such categories with existing ones that had very close meaning. We removed one predictor (*how methodology acquired*) for which the values were not clear enough and inconsistent with values in *used methodology*. Variables *user base – business units*, *user base – locations* and *user base – concurrent users* contained mixed types of values: point numeric and intervals. We categorized them using three intervals: ‘1’, ‘2-5’, ‘>5’. *User base – concurrent users* additionally had one more interval: ‘>100’.

Table 8-3 summarizes predictors kept and removed in this and successive steps of the statistical analysis.

4. Identifying relationships between predictors and dependent variables

In this step we used:

- *Spearman rank correlation coefficient* (SRCC) between dependent variables and numeric potential predictors (We did not apply Pearson product-moment correlation coefficient because it assumes only linear correlation which may not be relevant in our case).
- *Kruskal-Wallis one-way analysis of variance* (KW-ANOVA) between dependent variables and nominal potential predictors.

Besides analysing the values and significance of the above measures, we also analyzed if the identified relationships make sense from a causal perspective. For each nominal variable we analysed box-plots to determine how specific categories of nominal predictors influence dependent variables.

5. Identifying correlations / associations among predictors

Our aim was to build models with a Naïve Bayesian Classifier (NBC) structure. In such types of models predictors should not be correlated or associated (see Section 4.4 for details). As a result of this condition we also have to analyze possible correlations and associations between particular predictors. Therefore, we checked separately for each of the two datasets (productivity and defect rates):

- possible correlations between each pair of numeric predictors – using SRCC,

- possible associations between each numeric and each nominal predictor– using KW-ANOVA,
- possible associations between each pair of nominal predictors – using Phi, Cramer’s V and contingency coefficients.

This stage resulted in removing some predictors from further analysis.

Table 8-3 Predictors kept after each step of statistical analysis

| Predictor \ Dependent / Step | Productivity Rate | | | Defect Rate | | |
|--|-------------------|---|---|-------------|---|---|
| | 3 | 4 | 5 | 3 | 4 | 5 |
| Average Team Size | + | – | | + | – | |
| Project Elapsed Time | + | – | | + | + | – |
| Functional Size | + | – | | + | – | |
| Summary Work Effort | + | – | | + | + | + |
| Value Adjustment Factor | + | + | – | + | – | |
| Application Type | + | – | | + | – | |
| Architecture | + | + | – | + | + | – |
| Business Area Type | + | – | | + | – | |
| CASE Tool Used | + | – | | + | + | + |
| Client-Server | + | – | | + | + | – |
| Development Platform | + | + | + | + | + | + |
| Development Type | + | + | + | + | – | |
| How Methodology Acquired | – | | | – | | |
| Intended Market | + | + | – | + | + | – |
| Language Type | + | + | + | + | – | |
| Organisation Type | + | – | | + | – | |
| Package Customisation | + | – | | + | – | |
| Primary Programming Language | + | – | | + | – | |
| Used Methodology | + | + | + | + | + | + |
| User Base: Business Units | + | – | | + | – | |
| User Base: Concurrent Users | + | + | – | + | – | |
| User Base: Locations | + | + | + | + | + | – |
| ‘+’ indicates that particular predictor was kept in specific step | | | | | | |
| ‘–’ indicates that particular predictor was removed in specific step | | | | | | |

8.3 Bayesian net for prior productivity and defect rates

To produce an NBC structure, the easiest way to model dependent variables was to discretise them. Since we already had them transformed using Ln, we now discretised these transformed variables by rounding to the nearest integer. Thus we got 2 dependent variables (Equations 27 and 28).

$$\ln \text{ productivity rate round} = \text{Round}(\text{Ln}(\text{productivity rate})) \quad (27)$$

$$\ln \text{ defect rate round} = \text{Round}(\text{Ln}(\text{defect rate})) \quad (28)$$

These two adjusted dependent variables are the root nodes in the model – parents for observed leaf nodes (predictors). To improve model usability we transformed dependent variables back to the original scale (Equations 29 and 30).

$$productivity\ rate = e^{\ln\ productivity\ rate\ round} \quad (29)$$

$$defect\ rate = e^{\ln\ defect\ rate\ round} \quad (30)$$

The PDR model uses predictors which we identified as significantly correlated or associated with dependent variables and which were independent on other predictors. They are listed in Table 8-4.

Table 8-4 Summary of predictors in PDR Model

| Predictor | Predictor for* | Scale |
|--|----------------|---|
| CASE Tool Used | D | Boolean |
| Development Platform | P D | Labelled: 'PC', 'Mid-Range', 'Mainframe', 'Multi' |
| Development Type | P | Labelled: 'New', 'Enhancement', 'Re-Development' |
| Functional Size | D | Numeric: 1.65–36316 FPs (10 fixed intervals) |
| Language Type | P | Labelled: '2GL', '3GL', '4GL', 'ApG' |
| Used Methodology | P D | Boolean |
| User Base – Locations | P | Ranked: '1', '2-5', '>5' |
| * P – Productivity Rate; D – Defect Rate | | |

There is one change in the list of predictors obtained earlier from statistical analysis. We added *functional size* as predictor for *defect rate* even though we did not find any correlation between them in the dataset analyzed earlier. However, other sources such as [127 cited after 222] report that the *defect rate* increases as the *functional size* increases. The discretisation for *functional size* is a result of the same type of transformation which we applied to our dependent variables: Round(Ln(*functional size continuous*)) transformed back from logged values to the original scale expressed in FPs as illustrated in Table 8-5.

Table 8-5 Transforming *functional size* to meaningful intervals

| Round(Ln(<i>Functional size continuous</i>)) | Intervals for Round(Ln(<i>Functional size continuous</i>)) | Intervals transformed with Round(Exp()) functions expressed in FPs |
|--|--|--|
| 1 | 0.5 – 1.5 | 2 – 4 |
| 2 | 1.5 – 2.5 | 5 – 11 |
| 3 | 2.5 – 3.5 | 12 – 32 |
| 4 | 3.5 – 4.5 | 33 – 89 |
| 5 | 4.5 – 5.5 | 90 – 244 |
| 6 | 5.5 – 6.5 | 245 – 664 |
| 7 | 6.5 – 7.5 | 665 – 1807 |
| 8 | 7.5 – 8.5 | 1808 – 4914 |
| 9 | 8.5 – 9.5 | 4915 – 13359 |
| 10 | 9.5 – 10.5 | 13360 – 36316 |

Figure 8-1 illustrates the complete structure of the model. There are two separate NBC models – one for each dependent variable. Some of the predictors appear in both

models. We could avoid including them twice but this would require much more difficult assessment of CPTs in these nodes. We would need to define the probabilities given not just one parent but two (productivity and defect rates). The only drawback caused by the selected structure with repeated variables is the need to enter the same observation to the predictors appearing twice.

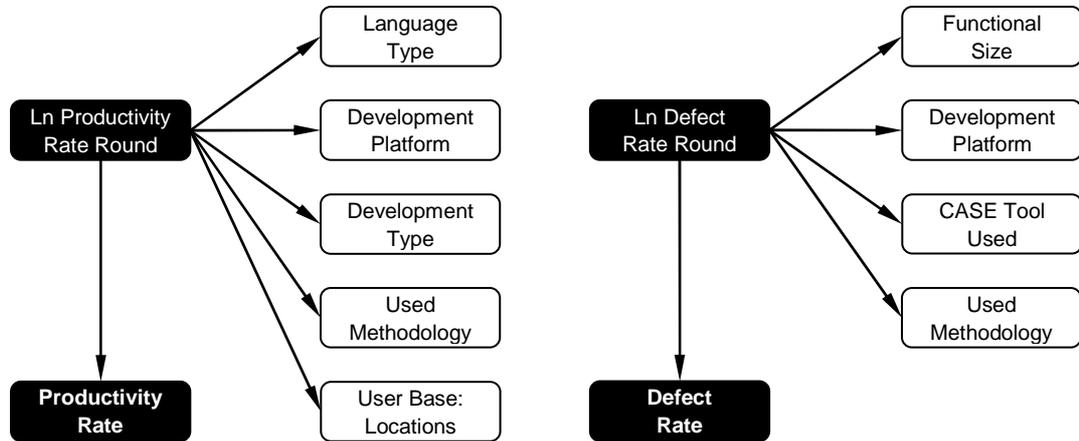


Figure 8-1 Structure of a BN for predicting productivity and defect rates

The prior distribution for *ln productivity rate round* (Equation 31) reflects the empirical distribution from the ISBSG dataset (see Table 8-1). The prior distribution for *ln defect rate round* (Equation 32) is defined to match other sources (discussed in Chapter 3) which report much higher defect rates than in this dataset. Therefore, in our model we increased prior mean for *defect rate* distribution, which now is a mixture of ISBSG data and other reported data.

$$\ln \text{ productivity rate round} \sim \text{LogNormal}(-2.18, 1.11^2) \quad (31)$$

$$\ln \text{ defect rate round} \sim \text{LogNormal}(-3, 2^2) \quad (32)$$

When creating NBC models, conditional probabilities (for predictors) are often estimated from the dataset by unsupervised learning. We believe that the ISBSG dataset does not allow such model learning. There are two main reasons:

1. Missing data – When analyzing crosstabs between the dependent variable and each predictor we found that in many cells there were no observations. We believe that this is not because such combinations of values of predictor and dependent variables do not exist in reality. Rather, we think that it just happened that data donors did not provide data for such combinations and that the majority of them may actually occur (albeit rarely).

2. Prior probabilities for predictors – We are not convinced that prior probabilities for predictors estimated from the dataset by analyzing frequencies really reflect the whole population of the developed projects.

Thus, we analyzed the crosstabs and frequencies and adjusted them by our beliefs and we put such adjusted priors into the model. We needed conditional probabilities of predictor given the dependent variable to be entered into the model. As we found it difficult to generate them directly, we prepared them in four steps:

1. For a given predictor, for each of its categories we defined parameters (μ , σ) for Normal distributions reflecting productivity and defect rates depending on specific predictor $P(Y|X)$.
2. We defined a prior unconditional probability for each state of each predictor $P(X)$.
3. Using the data from steps 1 and 2 we calculated the distributions for dependent variables $P(Y)$.
4. To estimate conditional probabilities for predictor given dependent variable $P(X|Y)$ we used the Bayes's rule shown in Equation 3 on page 46.

Table 8-6 illustrates the direction of numeric and ranked predictors on dependent variables reflected in the model.

Table 8-6 Impact of predictors on dependent variables reflected in the model

| Predictor \ Dependent | Productivity Rate | Defect Rate |
|---|---|---|
| CASE Tool Used | <i>none</i> | 'Yes', 'No' |
| Development Platform | 'Mainframe', 'Mid-Range', 'Multi', 'PC' | 'Mainframe', 'PC', 'Mid-Range', 'Multi' |
| Development Type | 'Re-development', 'Enhancement', 'New' | <i>none</i> |
| Functional Size | <i>none</i> | + |
| Language Type | '2GL', '3GL', '4GL', 'ApG' | <i>none</i> |
| Used Methodology | 'Yes', 'No' | 'Yes', 'No' |
| User Base – Locations | – | <i>none</i> |
| '+' indicates positive impact – as predictor increases dependent also increases '–' indicates negative impact – as predictor increases dependent decreases labelled predictors – order of categories reflects increase of dependent variable: the lowest value of dependent variable is for the first category, the highest value – for the last category | | |

8.4 Model validation

Since we adjusted the distribution for *defect rate* and switched the impact of *functional size* on *defect rate* to the opposite, there was little sense in validating the

model against the ISBSG dataset. Therefore, we decided to validate the model using a set of scenarios where we analyze the impact of particular predictors as well as different combinations of predictors on dependent variables.

Case 1 – No observations

First, we analyze if the model predictions are consistent with initially assumed distributions. We execute the model without any observations. Figure 8-2 illustrates predicted probability density distributions with basic summary statistics for each dependent variable (also logged).

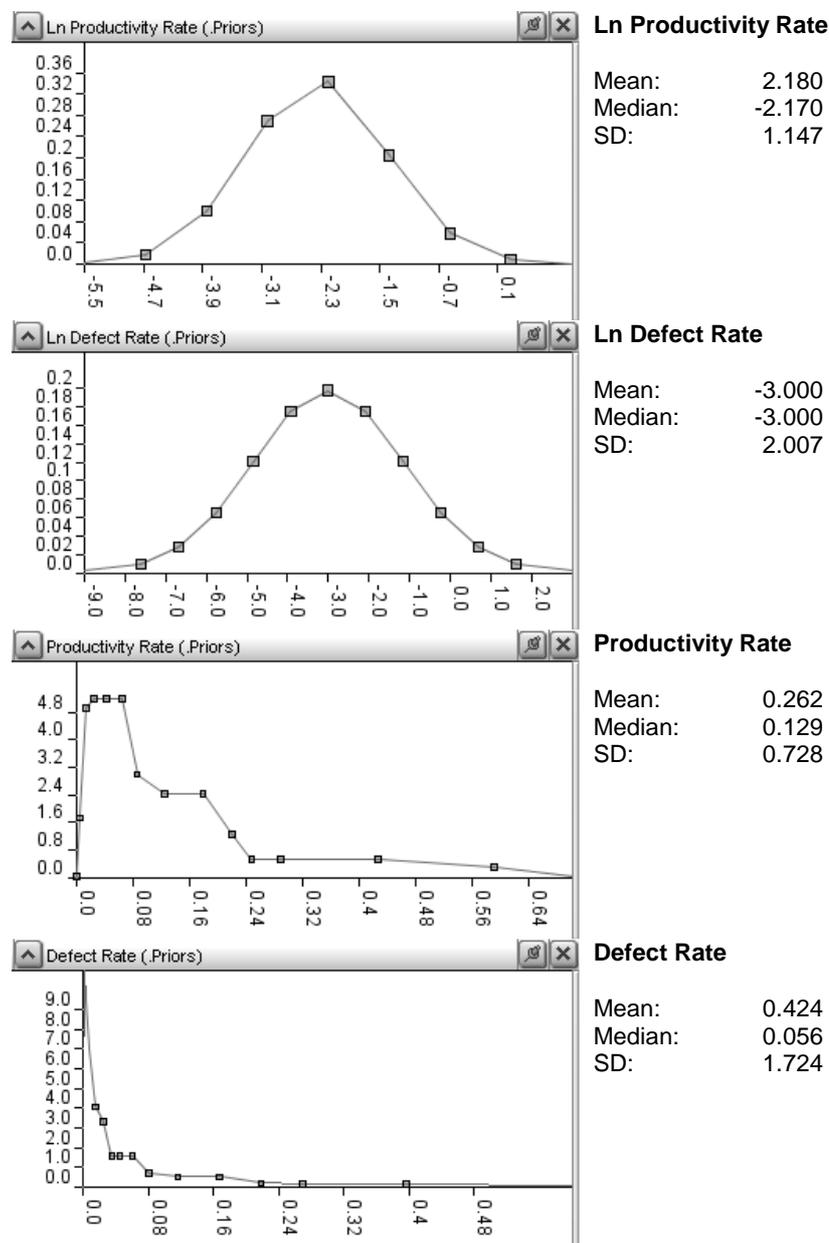


Figure 8-2 Predictions when no observations are entered

We can observe that predicted distributions are consistent with expressions reflecting their prior probabilities as summarized in Equations 31 and 32. Some inaccuracies for median and standard deviation (SD) are very low and are due to the fact that logged dependent variables are statically discretised as required of the model.

Case 2 – Impact of project size

In this case we analyze how the model reflects the impact of *functional size* on *defect rate*. When building this model we assumed (based on empirical data) that *defect rate* increases as *functional size* increases. It means that bigger projects have typically lower quality. Figure 8-3 illustrates predicted *defect rates* for four sample ranges of *functional size*. The first of them reflects the lowest size supported by our model (2-4 FPs), the last – the highest (13360-36316 FPs), and the two others reflect some intermediate intervals for *functional size*. Indeed, we can observe that the model predicts that *defect rate* increases with the increase of *functional size*. When *functional size* has the highest value *defect rate* is 7 times higher (comparing predicted median values) than in scenario when *functional size* is the lowest. Figure 8-4 illustrates predicted median values of *defect rate* for all intervals of *functional sizes*.

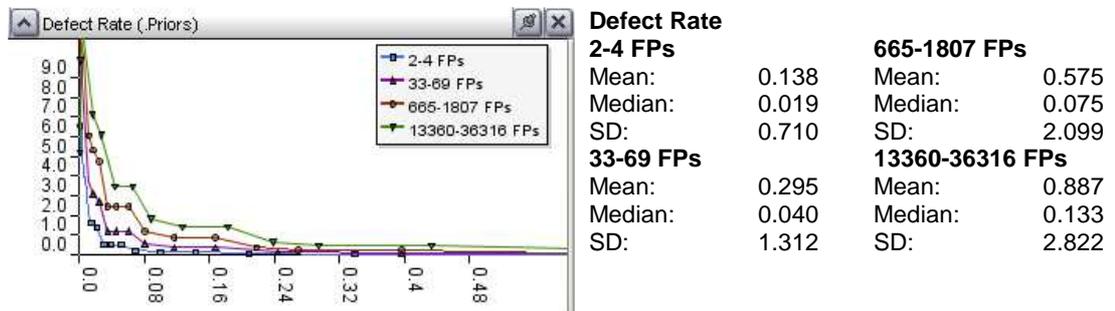


Figure 8-3 Predicted *defect rate* for different *functional sizes*

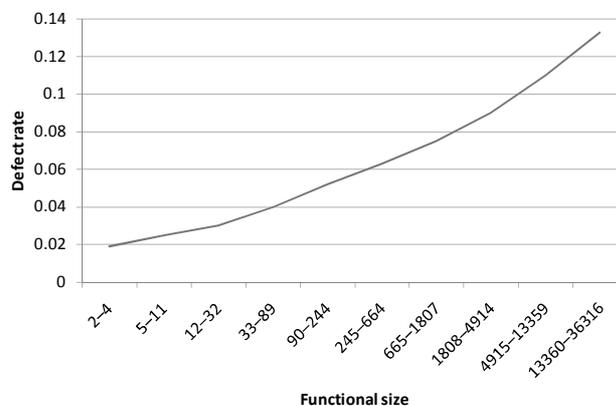


Figure 8-4 Predicted median values for *defect rate* depending on various *functional sizes*

Case 3 – Impact of nominal predictors

Here we analyze the predictions of dependent variables for different states of nominal predictors. We perform this analysis by assigning only one observation at a time – to one predictor only; no other predictors have observations entered.

Figure 8-5 illustrates predictions for *defect rate* with basic summary statistics in two scenarios: when CASE tool is and is not used during a project. The model predicts that when no CASE tool is used we should expect that *defect rate* should be around 5.5 times higher compared to the situation when a CASE tool is used during a project. It can be explained by the fact that using CASE tools is one of the ways to overcome project complexity and size which otherwise lead to increased *defect rate*.

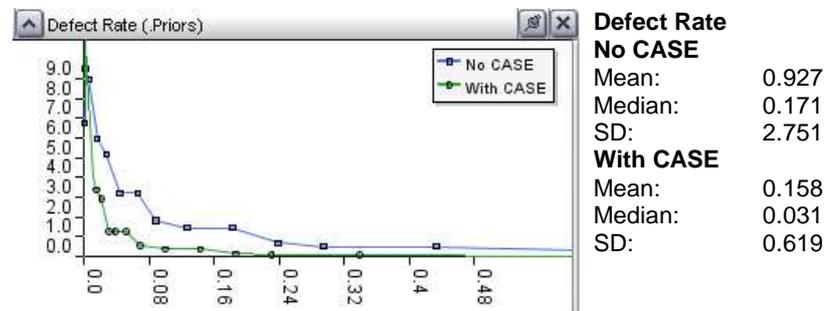


Figure 8-5 Predicted *defect rate* depending on CASE tool usage

Figure 8-6 illustrates model predictions for *productivity rate* and *defect rate* depending on specific *development platform* used in the project. We can observe that predicted *productivity rate* decreases according with the size and complexity of *development platform*. The highest *productivity rate* is expected for ‘PC’ and the lowest for ‘Mainframe’. When multiple platforms are used predicted *productivity rate* should be slightly higher than the average value, but neither the lowest nor the highest.

When analyzing predicted *defect rate* we can observe that it is expected to be significantly higher when multiple platforms are used compared to when a single platform is used. The lowest *defect rate* is expected for ‘Mainframe’ with ‘PC’ as the second.

When analyzing the impact of *development type* on *productivity rate* (Figure 8-7) we can observe that the model predicts highest *productivity rate* for ‘New’ projects, followed by ‘Enhancements’. The lowest *productivity rate* is expected for projects which are ‘Re-developments’. This can be explained by the fact that if it is only new functionality to be delivered no analysis of existing functionality needs to be performed.

When the project is ‘Re-development’ much effort is required on analysing the functionality of existing software which is about to be re-implemented (on new platform, database or operating system, with new GUI etc.).

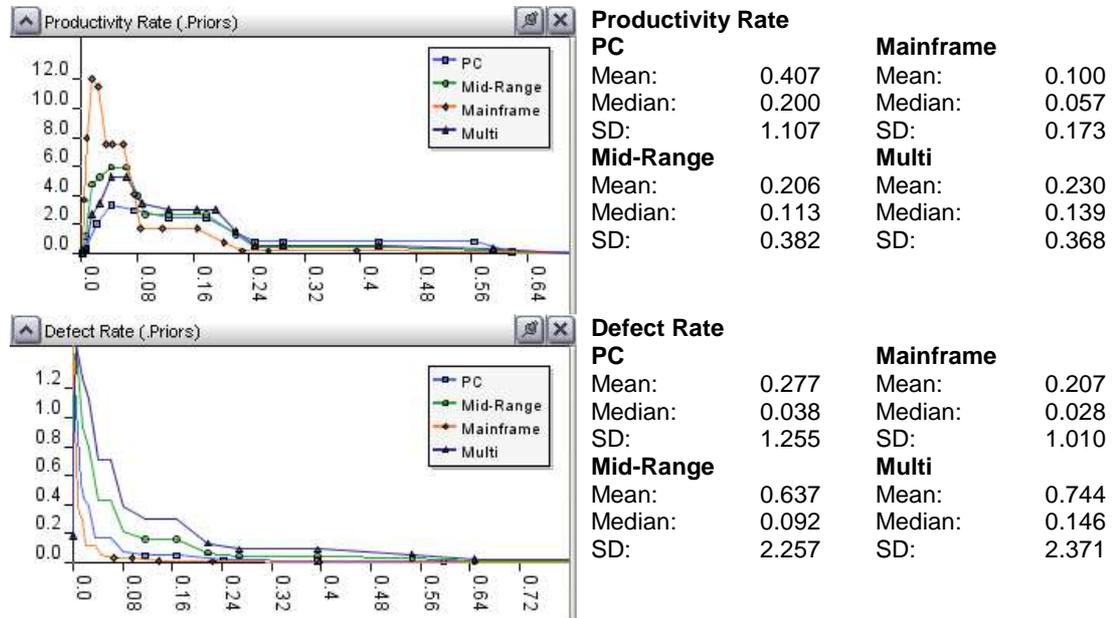


Figure 8-6 Predicted productivity and defect rates depending on *development platform*

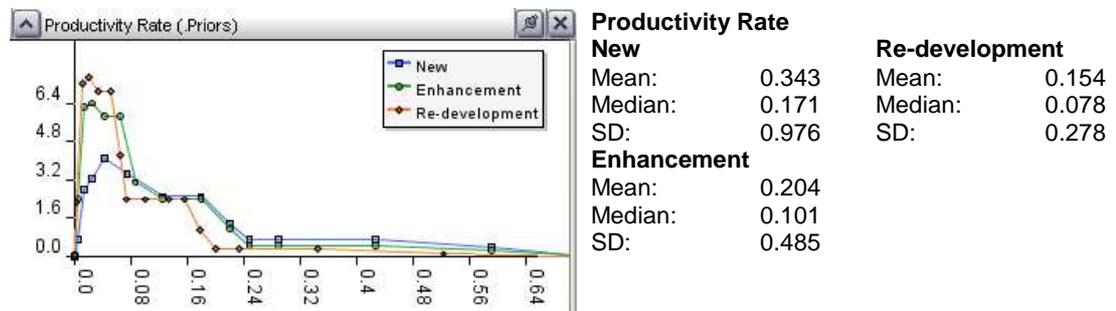


Figure 8-7 Predicted *productivity rate* depending on *development type*

Figure 8-8 illustrates predictions for *productivity rate* depending on *language type*. This relationship appears to be straightforward – the higher level of language type used in a project, the higher productivity achieved for this project. Thus the lowest productivity is typically achieved in projects when ‘2GL’ language is used, and the highest with ‘ApG’ (application generators).

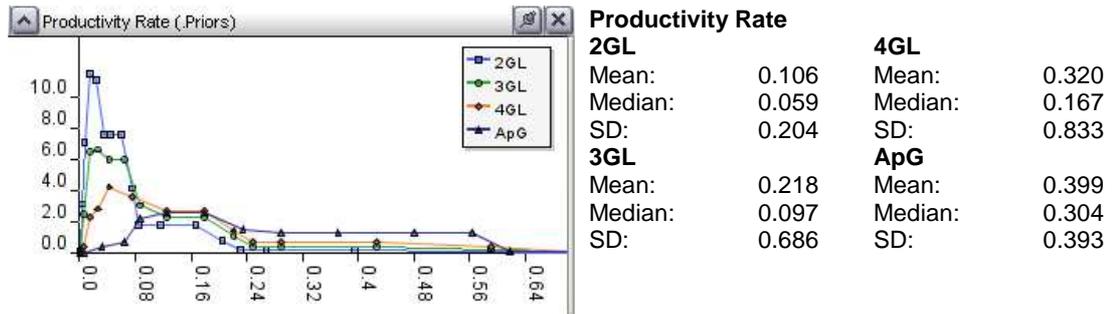


Figure 8-8 Predicted productivity rate depending on language type

Used methodology appears to have a two-fold impact on dependent variables – good on *defect rate* but bad on *productivity rate*. The model predicts (Figure 8-9) that when no methodology is used *defect rate* is expected to be around 3 times higher compared to when methodology is used. This can be explained by the fact that using a development methodology indicates more mature development process and we should expect better quality being delivered by such process. However, the impact on *productivity rate* seems to be the opposite of what we expected. The model predicts higher *productivity rate* without methodology compared to when a development methodology is used. A possible explanation to this surprising result is captured in the PDR model, and also clearly observed in the dataset: using a methodology involves spending additional effort on activities associated with developing documentation for product and process. This effort is ‘wasted’ from the productivity point of view since it is allocated to activities that do not extend functionality.

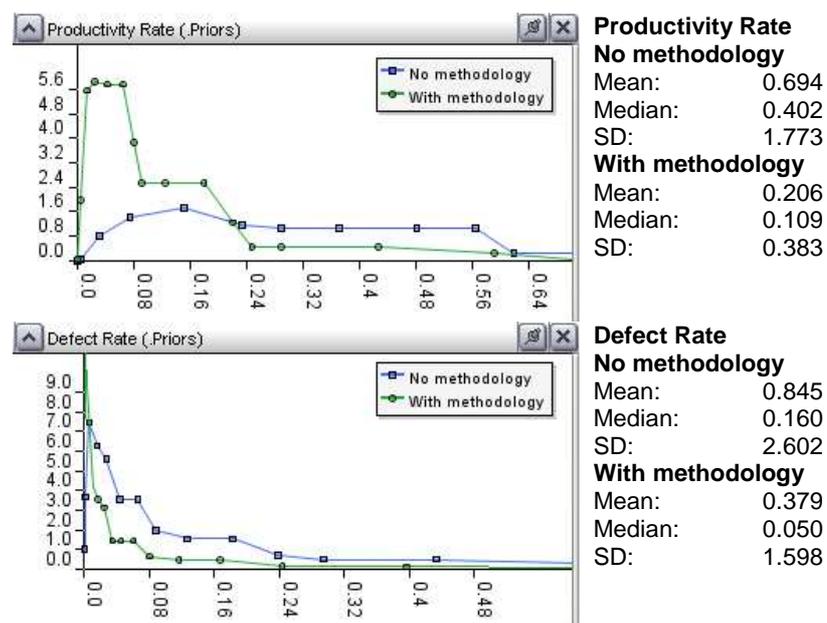


Figure 8-9 Predicted productivity and defect rates depending on used methodology

Figure 8-10 illustrates the impact of *user base – locations* on *productivity rate*. The model predicts that as the number of user locations increases, *productivity rate* is expected to be lower. This is caused by increased communication and infrastructure required to support more user locations.

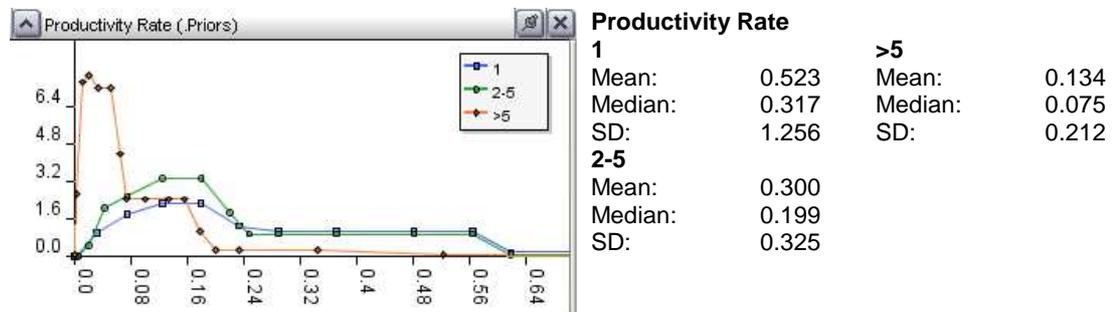


Figure 8-10 Predicted *productivity rate* depending on number of user locations

Case 4 – Most and least favourable scenarios

In this case we analyze what are the combinations of predictors causing the least and most favourable *productivity rate* and *defect rate*. By analyzing the impact of particular predictors on dependent variables we created such combinations of predictors which we present in Table 8-7. The lack of impact of a particular predictor on particular dependent variable is marked with ‘-’.

Table 8-7 Most and least favourable scenarios

| Dependent & scenario Predictor | Productivity rate | | Defect rate | |
|-----------------------------------|-------------------|-----------------|------------------|-----------------|
| | Least favourable | Most favourable | Least favourable | Most favourable |
| CASE tool used | - | - | No | Yes |
| Development platform | Mainframe | PC | Multi | Mainframe |
| Development type | Re-development | New | - | - |
| Functional size | - | - | 13360-36316 | 2-5 |
| Language type | 2GL | ApG | - | - |
| Used methodology | Yes | No | No | Yes |
| User base – locations | >5 | 1 | - | - |

Figure 8-11 illustrates predicted productivity and defect rates in least and most favourable scenarios. In the most favourable scenario the model predicts *productivity rate* 21 times higher than in the least favourable scenario. Predicted *defect rate* is 128 times lower in the most favourable scenario compared to the least favourable. Such wide ranges, especially for *defect rate* may seem too wide. However, we believe that some of these extremes are very unlikely to happen in reality. For example, the least favourable scenario for *defect rate* is when a project is very large and developed using

multiple hardware platforms. For such projects it is almost impossible not to use any CASE tool or development methodology. But actually this least favourable scenario assumes that neither CASE tool nor methodology are used for such projects.

We can clearly observe that ranges for *productivity rate* and *defect rate* are much wider than ranges of these variables in the main part of the Productivity Model. These differences are caused by the fact that we calibrated the main part of the Productivity Model according to questionnaire survey data but the PDR model according to ISBSG dataset (slightly adjusted by expert knowledge). Experts filling the questionnaire already had a view on some limited types of software they developed while the ISBSG dataset covers a wider range of software.

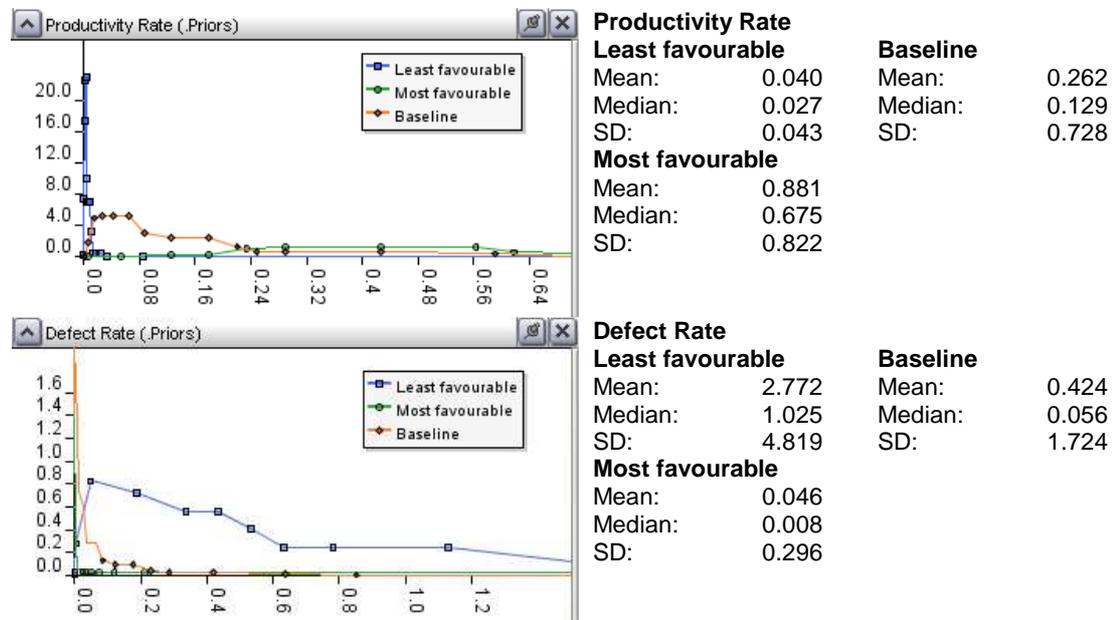


Figure 8-11 Predicted productivity and defect rates in most and least favourable scenarios

Case 5 – Combining PDR model with Productivity Model

In this case we analyze the impact of observations entered in the PDR model on predictions provided by the Productivity Model. We analyze two scenarios for the same functional size (1000 FPs), the same target *defect rate* to be achieved (0.2 D/FP) and wish to get a prediction for *revised effort* from the Productivity Model. In the first scenario we assume that the software project will be developed on a mainframe and with 3GL programming language. The second scenario assumes that a project will be developed on multiple platforms using an application generator. For a fair comparison we further assume that in both scenarios process and people quality and other factors from the Productivity Model have ‘the same’ values.

Figure 8-12 summarizes predictions from the two models for various variables. We can see that observations entered to the PDR model significantly change predictions provided by the Productivity Model. Predicted *revised productivity rate* from the Productivity Model in scenario 2 is over 3.2 times higher than in scenario 1. It means that significantly less *revised effort* is required in scenario 2 than in scenario 1.

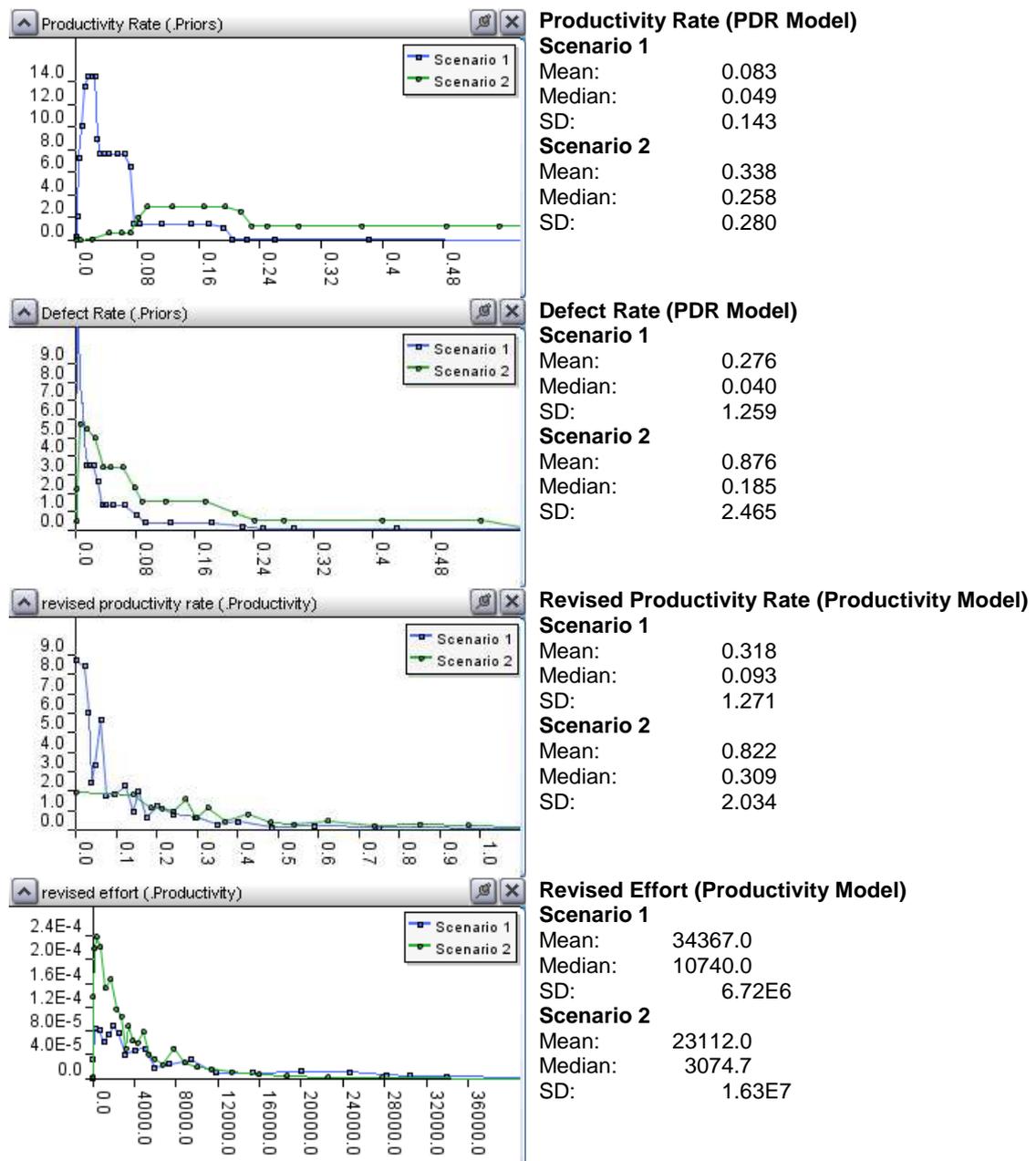


Figure 8-12 Predictions after linking PDR Model with Productivity Model

Sensitivity analysis

We also perform a sensitivity analysis in which we validate if the impact of predictors is reflected in the model in a way expected. Specifically we check if the

direction of impact of each predictor is consistent with the assumptions summarized in Table 8-6.

Figure 8-13 illustrates the results of this sensitivity analysis by visualizing mean and median values of *productivity rate* and *defect rate*. We can observe that the most influential predictor for *productivity rate* is *used methodology* followed by *language type* and *user base - locations*. The least influential predictors for *productivity rate* are *development platform* and *development type*. *CASE tool used* is the most influential predictor for *defect rate*. *Development platform*, *functional size* and *used methodology* have slightly lower impact on *defect rate*.

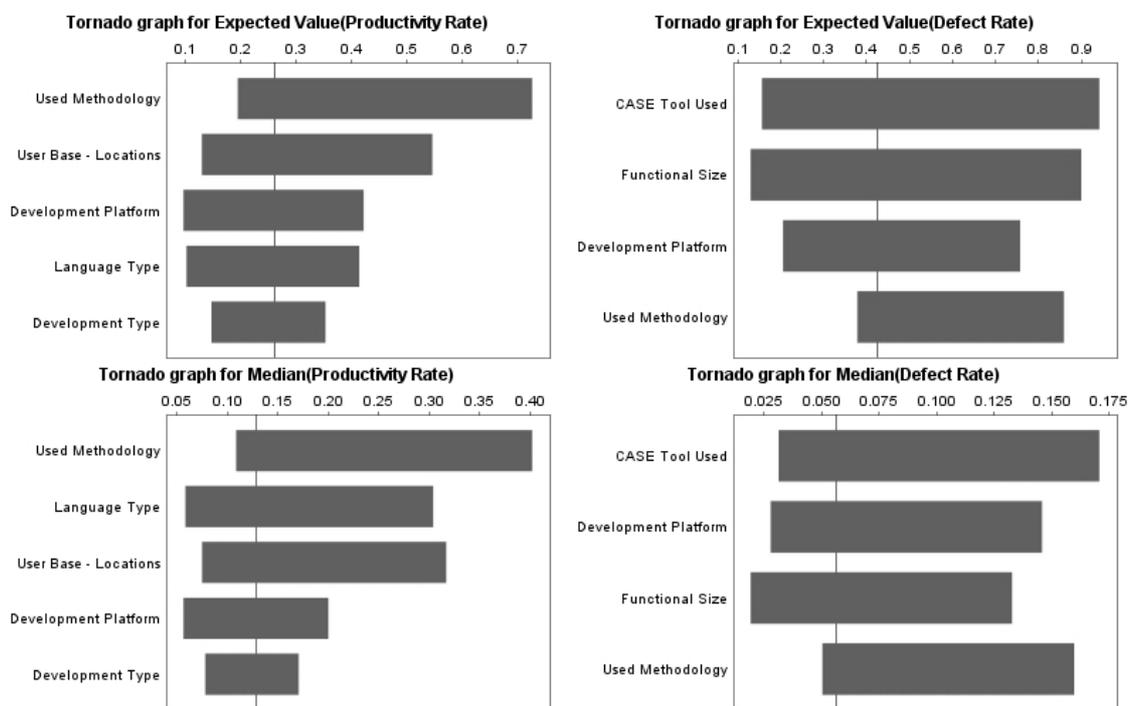


Figure 8-13 Tornado graphs illustrating sensitivity of dependent variables

We can observe that the strength of predictors' impact on *productivity rate* varies more among predictors than on *defect rate*. This strength of impact on dependent variables reflects in part relationships observed in ISBSG dataset adjusted by data from other sources and our expert knowledge.

Using PDR model in goal-seeking

The PDR model is a BN and as such could in theory be used for goal-seeking analysis where we set a target value for a dependent variable and let the model answer how to achieve such a target. However, we emphasize that the predictors we included in the model should be treated as uncontrollable values. This means the model should be

used for one-way analysis only – from predictors to dependent variables as in classical parametric models.

Such a limitation is caused by our selection of the predictors' list. For example, *development platform* is the second most important predictor for *productivity rate*. If *development platform* is 'PC' then *productivity rate* should be lower than when *development platform* is 'Mainframe'. However, it is not selection of *development platform* driving the *productivity rate* from a decision-makers' point of view. It is not about answering a question: "which development platform should we select to be more productive?" Some systems simply have to be developed on mainframes, and others on PCs. Hence, although it is technically possible to perform goal-seeking analysis using the PDR model, such analysis would be meaningless.

8.5 Summary

The PDR model discussed in this chapter provides an extension to the Productivity Model for use in situations where users cannot provide productivity and defect rates from similar past projects. The PDR model captures the results of statistical analysis of the ISBSG dataset adjusted by other sources of data and expert judgement. The validation shows that the PDR model provides reasonable predictions which can be passed as input data to the Productivity Model.

9 Model for predicting types of defects

Traditional defect prediction models treat all defects equally. However, software companies need to categorize defects found in their products in order to prioritize which are fixed first. Several categorizations are supported by popular defect tracking systems, such as Bugzilla [242]. The new Defect Types Model (DTM) introduced in this chapter enables us to perform estimates for proportions of defects of various types categorised by their severity. This model uses both controllable (process quality) and uncontrollable (describing the nature of software) factors. The model incorporates results from statistical analysis of the ISBSG dataset using the Statistica 8 tool [234]. The new contributions of this chapter are the statistical analysis of defect types' factors and the DTM. This chapter is partially based on [222].

9.1 The dataset

For this analysis we again used the ISBSG R9 dataset [118]. We had to significantly reduce the number of observations used in the analysis to 119 so that they all meet the following criteria:

- *data quality* has either 'A' or 'B' assigned (meaning respectively 'very good' and 'good' quality) as suggested in [120],
- Observations where *total defects delivered* is either 0 or missing are excluded.

ISBSG defines a defect as “a failure of some part of an application” [119]. They distinguish three types of defects in the dataset depending on their severity:

1. Minor – “A minor defect does not make the application unusable in any way, (e.g. a modification is required to a screen field or report)”.
2. Major – “A major defect causes part of the application to become unusable”.
3. Extreme – “A failure of some part of an application that causes the application to become totally unusable”.

9.2 Statistical analysis of dataset

The steps of this analysis are very similar to these performed in analysis of productivity and defect rates (Section 8.2). We discuss variations from this approach in appropriate places. Appendix D.1 contains detailed figures and tables illustrating the results of this statistical analysis.

1. Analysis of dependant variables

The whole model contains three predictive variables: proportion of minor defects (*prop minor*), proportion of major defects (*prop major*), proportion of extreme defects (*prop extreme*) defined respectively as shown in Equations 33-35.

$$\text{prop minor} = \frac{\text{minor defects}}{\text{total defects delivered}} \quad (33)$$

$$\text{prop major} = \frac{\text{major defects}}{\text{total defects delivered}} \quad (34)$$

$$\text{prop extreme} = \frac{\text{extreme defects}}{\text{total defects delivered}} \quad (35)$$

The range for each predictive variable is [0, 1]. Because these categories of defects are the only ones we analyze and they are disjunctive, the proportions of defect categories sum to 1. The basic summary statistics for the proportions of different types of defects are presented in Table 9-1. They confirm (mainly skeweness – substantially different from 0) that the distributions of these variables are not normally distributed and not even symmetrical.

Table 9-1 Summary of descriptive statistics for dependant variables

| Variable \ Statistic | Mean | Median | Lower Quartile | Upper Quartile | Skewness | Kurtosis |
|----------------------|------|--------|----------------|----------------|----------|----------|
| PropMinor | 0.68 | 0.77 | 0.50 | 1.00 | -0.86 | -0.33 |
| PropMajor | 0.25 | 0.19 | 0.00 | 0.36 | 1.44 | 1.67 |
| PropExtreme | 0.07 | 0.00 | 0.00 | 0.08 | 3.79 | 17.08 |

2. Preparing list of potential predictors

We nominated a set of possible predictor variables which we believe may influence the predictive variables (Table 9-2). We took descriptions for most of these potential predictors from [122].

Table 9-2 Summary of potential predictor variables used to build the model

| Name | Type | Description |
|----------------------|------------|--|
| Average Team Size | Continuous | The average number of people that worked on the project, (calculated from the team sizes per phase). |
| Defect Rate | Continuous | Number of defects per 1 FP |
| Functional Size | Continuous | The unadjusted function point count (before any adjustment by a Value Adjustment Factor if used). |
| Max Team Size | Continuous | The maximum number of people that worked at any time on the project, (peak team size). |
| Productivity Rate | Continuous | Number of delivered FPs per 1 person-hour |
| Project Elapsed Time | Continuous | Total elapsed time for the project in calendar months. |

| Name | Type | Description |
|------------------------------|------------|---|
| Summary Work Effort | Continuous | Provides the total effort in hours recorded against the project. |
| Value Adjustment Factor | Continuous | The adjustment to the function points, applied by the project submitter, that takes into account various technical and quality characteristics e.g.: data communications, end user efficiency etc. This data is not reported for some projects, (i.e. it equals 1). |
| Activity Build | Nominal | This indicates whether the project contained the Build stage. |
| Activity Design | Nominal | This indicates whether the project contained the Design stage. |
| Activity Implement | Nominal | This indicates whether the project contained the Implementation stage. |
| Activity Planning | Nominal | This indicates whether the project contained the Planning stage. |
| Activity Specification | Nominal | This indicates whether the project contained the Specification stage. |
| Activity Test | Nominal | This indicates whether the project contained the Testing stage. |
| Application Type | Nominal | This identifies the type of application being addressed by the project. (e.g.: information system, transaction/production system, process control.) |
| Architecture | Nominal | A derived attribute for the project to indicate if the application is Stand alone, Multi-tier, Client server, or Multi-tier with web public interface. |
| Business Area Type | Nominal | This identifies the type of business area being addressed by the project where this is different to the organisation type. (e.g.: Manufacturing, Personnel, Finance). |
| CASE Tool Used | Nominal | Whether the project used any CASE tool. |
| Client-Server | Nominal | Indicator of whether the application or product requires more than one computer to operate different components or parts of it. |
| Development Platform | Nominal | Defines the primary development platform, (as determined by the operating system used). Each project is classified as: PC, Mid Range, Main Frame or Multi platform. |
| Development Type | Nominal | This field describes whether the development was a new development, enhancement or re-development. |
| How Methodology Acquired | Nominal | Describes whether the development methodology was purchased or developed in-house, or a combination of these. |
| Intended Market | Nominal | This field describes the relationship between the project's customer, end users and development team. |
| Language Type | Nominal | Defines the language type used for the project: e.g. 3GL, 4GL, Application Generator etc. |
| Organisation Type | Nominal | This identifies the type of organisation that submitted the project. (e.g.: Banking, Manufacturing, Retail). |
| Package Customisation | Nominal | This indicates whether the project was a package customisation. |
| Primary Programming Language | Nominal | The primary language used for the development: JAVA, C++, PL/1, Natural, Cobol etc. |
| Used Methodology | Nominal | States whether a development methodology was used by the development team to build the software. |
| User Base – Business Units | Ordinal | Number of business units (or project business stakeholders) serviced by the software application. |
| User Base – Concurrent Users | Ordinal | Number of users using the system concurrently. |
| User Base – Locations | Ordinal | Number of physical locations being serviced/supported by the installed software application. |

3. Analyzing categories of potential nominal predictors

This step was very similar to the relevant step in the analysis of productivity and defect rates. We analyzed the frequencies for nominal predictors and we grouped categories with very few observations with others that had very close meaning. Again, we removed one predictor (*how methodology acquired*) for which the values were not clear enough and inconsistent with values in *used methodology*. Variables *user base – business units*, *user base – locations* and *user base – concurrent users* contained mixed types of values: point numeric and intervals. We categorized them using three intervals: ‘1’, ‘2-5’, ‘>5’. Continuous predictors were not normally distributed so we transformed them using the Ln (natural logarithm) function.

Table 9-3 summarizes predictors kept and removed in this and successive steps of the statistical analysis.

4. Identifying relationships between predictors and dependent variables

As in previous analysis we used we used *Spearman rank correlation coefficient* (SRCC) and *Kruskal-Wallis one-way analysis of variance* (KW-ANOVA). Again, we also analyzed if the identified relationships make sense from a causal perspective and for each nominal variable we analysed box-plots to determine how specific categories of nominal predictors influence dependent variables.

5. Identifying correlations / associations among predictors

We checked separately for each of the two datasets (productivity and defect rates):

- possible correlations between each pair of numeric predictors – using SRCC,
- possible associations between each numeric and each nominal predictor – using KW-ANOVA,
- possible associations between each pair of nominal predictors – using Phi, Cramer’s V and contingency coefficients.

This stage resulted in removing some predictors from further analysis.

After all steps of identifying predictors we got only *activity test* as predictor for *prop minor* and *prop major*. For *prop extreme* additional predictors are: *functional size* and *package customisation*.

Table 9-3 Predictors kept after each step of statistical analysis

| Dependent / Step Predictor | Prop Minor | | | Prop Major | | | Prop Extreme | | |
|---|------------|---|---|------------|---|---|--------------|---|---|
| | 3 | 4 | 5 | 3 | 4 | 5 | 3 | 4 | 5 |
| Average Team Size | + | - | | + | - | | + | - | |
| Defect Rate | + | + | - | + | + | - | + | + | - |
| Functional Size | + | - | | + | - | | + | + | + |
| Max Team Size | + | - | | + | - | | + | - | |
| Productivity Rate | + | - | | + | - | | + | + | - |
| Project Elapsed Time | + | - | | + | - | | + | - | |
| Summary Work Effort | + | - | | + | - | | + | - | |
| Value Adjustment Factor | + | - | | + | - | | + | - | |
| Activity Planning | + | - | | + | - | | + | - | |
| Activity Specification | + | - | | + | - | | + | - | |
| Activity Design | + | - | | + | - | | + | - | |
| Activity Build | + | - | | + | - | | + | - | |
| Activity Test | + | + | + | + | + | + | + | + | + |
| Activity Implement | + | + | - | + | + | - | + | + | - |
| Application Type | + | - | | + | - | | + | - | |
| Architecture | + | - | | + | - | | + | - | |
| Business Area Type | + | - | | + | - | | + | - | |
| CASE Tool Used | + | - | | + | - | | + | - | |
| Client-Server | + | - | | + | - | | + | - | |
| Development Platform | + | - | | + | - | | + | - | |
| Development Type | + | - | | + | - | | + | - | |
| How Methodology Acquired | - | | | - | | | - | | |
| Intended Market | + | - | | + | - | | + | + | - |
| Language Type | + | - | | + | - | | + | - | |
| Organisation Type | + | - | | + | - | | + | - | |
| Package Customisation | + | - | | + | - | | + | + | + |
| Primary Programming Language | + | - | | + | - | | + | - | |
| Used Methodology | + | - | | + | - | | + | + | - |
| User Base – Business Units | + | - | | + | - | | + | - | |
| User Base – Concurrent Users | + | - | | + | - | | + | - | |
| User Base – Locations | + | - | | + | - | | + | - | |
| ‘+’ indicates that particular predictor was kept in specific step ‘-’ indicates that particular predictor was removed in specific step | | | | | | | | | |

9.3 Bayesian net for estimating types of defects

The statistical analysis of the data resulted in identifying very few predictors for the dependent variables. This is caused by a general problem of predicting defect severity which is highly subjective: the same defect may be categorised as *minor* by one person but as *major* by another. As a result the dataset becomes noisy and identifying most influential predictors from such a dataset may not be possible.

Furthermore, this dataset did not contain variables which in our opinion (based on our experience) influence the proportions of types of defects. However, not all of them (e.g. code metrics) can be included in the model since the model is intended to be used

at an early stage of software development when little data describing project structure are available. As a result we decided:

- Not to build a Naïve Bayesian Classifier (NBC) with unsupervised learning using the ISBSG dataset but to build a model incorporating the mixture of expert knowledge and the results from this data analysis;
- To add more predictors which were either not included in the dataset or which were included in the dataset but not found correlated/associated with dependent variables but which we believe influence the dependent variables;
- To change the scale for Boolean predictors to ranked where we express not only ‘Yes’/‘No’ values but also intermediates on a 5-point scale.

Table 9-4 summarizes complete list of predictors used in the model.

Table 9-4 Summary of predictors in Defect Types Model

| Predictor | Predictor identified from | Description | Scale |
|------------------------|---------------------------------------|---|--|
| Activity Specification | analysis adjusted by expert’s opinion | Effectiveness of specification process (aggregation of process and people quality with effort) | Ranked: from ‘Very Low’ to ‘Very High’ |
| Activity Build | expert’s opinion | Effectiveness of coding process (aggregation of process and people quality with effort) | Ranked: from ‘Very Low’ to ‘Very High’ |
| Activity Test | analysis | Effectiveness of testing process (aggregation of process and people quality with effort) | Ranked: from ‘Very Low’ to ‘Very High’ |
| Deadline Pressure | expert’s opinion | Extent to which there is a pressure on delivering software faster than you normally would like to deliver such software | Ranked: from ‘Very Low’ to ‘Very High’ |
| Functional Size | analysis | Size of the project (on proportion of extreme defects) | Numeric: 2-36316 FPs |
| GUI usage | expert’s opinion | Extent at which graphical user interface (GUI) is used in the developed software | Ranked: from ‘None’ to ‘High’ |
| Package Customisation | analysis | Extent at which the project involves package customisation (on proportion of extreme defects) | Ranked: from ‘None’ to ‘High’ |
| Task complexity | analysis adjusted by expert’s opinion | How complex is the task/problem | Ranked: from ‘Very Low’ to ‘Very High’ |

This model (Figure 9-1) contains one labelled node representing the dependent variable which has the following three states:

- ‘prop minor’ – reflecting proportion of minor defects,
- ‘prop major’ – reflecting proportion of major defects,
- ‘prop extreme’ – reflecting proportion of extreme defects,

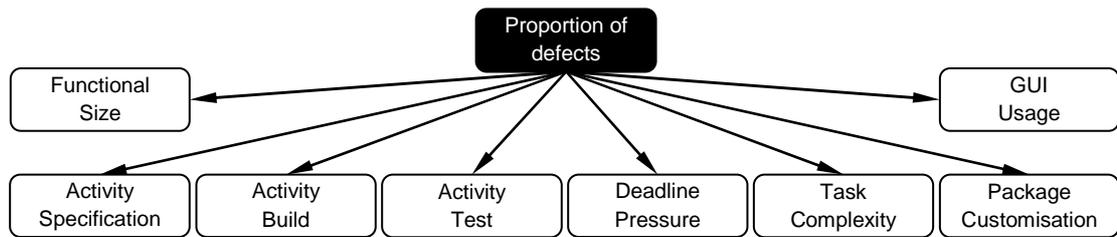


Figure 9-1 Structure of a BN for estimating types of defects

The probability of each state reflects the expected value of proportion of each type of defects. Such a structure can be easily calibrated because the sum of probabilities of these three states is always equal to 1. This reflects the situation where the sum of proportions of all types of defects also sums up to 1.

There is one drawback in such a model structure – we lose information on the probability distribution of the proportion of each type of defect. We only have the overall expected value. However, we still believe that the user can benefit from obtaining predictions from such a model.

The prior proportions of number of defects of each type are mixtures from ISBSG, Apache and Mozilla datasets (as published in [204]) based on the frequencies of observations and adjusted by our expert judgement (Figure 9-2).

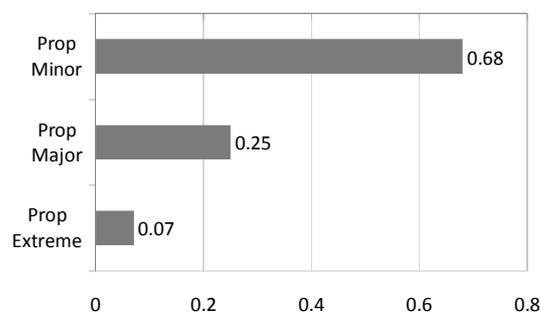


Figure 9-2 Prior proportions of each type of defects

CPTs entered into the model reflect the directions of impact presented in Table 9-5. For example, regarding *activity specification*: as the effectiveness of specification process increases, *prop extreme* and *prop major* decrease while *prop minor* increases. *Deadline pressure* has a completely reverse impact – as it increases, *prop extreme* and *prop major* increase and *prop minor* decreases.

Two of the predictors, *functional size* and *package customisation*, impact only on *prop extreme* (this was confirmed during data analysis). As values in these predictors increase, *prop extreme* defects also increases. And because of the constraint node

(described later) they also have impact on proportion of minor and major defects but it is not directly defined. Section D.2 contains a full specification of this model.

Table 9-5 Impact of predictors on dependent variables reflected in the model

| Predictor \ Dependent | Prop Minor | Prop Major | Prop Extreme |
|------------------------|-------------|-------------|--------------|
| Activity Specification | + | - | - |
| Activity Build | + | - | - |
| Activity Test | + | - | - |
| Deadline Pressure | + | - | - |
| Task Complexity | - | + | + |
| Functional Size | <i>none</i> | <i>none</i> | + |
| Package Customisation | <i>none</i> | <i>none</i> | + |

‘+’ indicates positive impact – as predictor increases, dependent also increases
‘-’ indicates negative impact – as predictor increases, dependent decreases

To estimate the number of defects of each type instead of the proportion, we need an estimate of the *total number of defects*. We can then extend the model from Figure 9-1 in a way shown on Figure 9-3. *Total number of defects* is an input to this model provided from other defect prediction models like [42, 75, 68, 69, 83, 90, 150, 206]. It can be either a point value provided by classic (parametric) defect prediction models or a probability distribution provided by BNs.

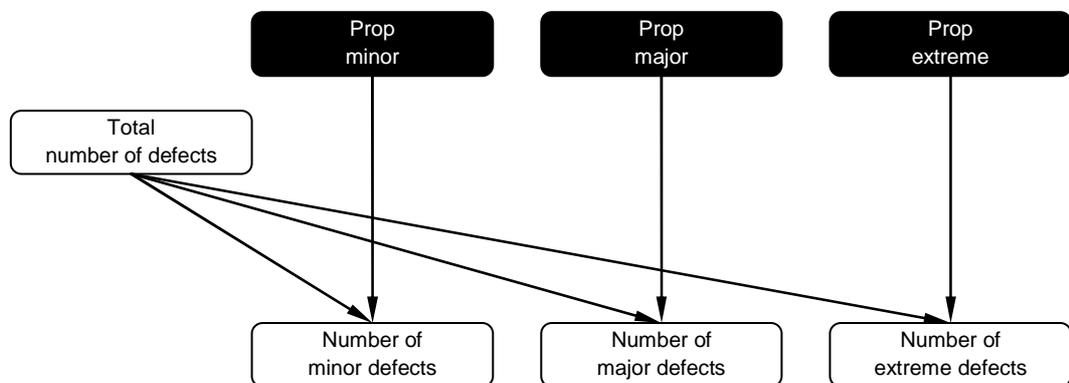


Figure 9-3 Linking a BN for estimating types of defects with another defect prediction model

The number of defects of a particular type is then calculated as shown in (Equations 36-38).

$$\text{number of minor defects} = \text{prop minor} * \text{total defects} \quad (36)$$

$$\text{number of major defects} = \text{prop major} * \text{total defects} \quad (37)$$

$$\text{number of extreme defects} = \text{prop extreme} * \text{total defects} \quad (38)$$

9.4 Model validation

Case 1 – Impact of functional size

In this case we analyze the impact of *functional size* on predicted proportions of defects. Figure 9-4 illustrates predicted proportions of defects for 5 selected *functional sizes*. The model predicts that as *functional size* increases *prop extreme* also increases. Predicted *prop minor* and *prop major* change at small degrees only as a response to change in *prop extreme*. These results are consistent with our findings in statistical analysis.

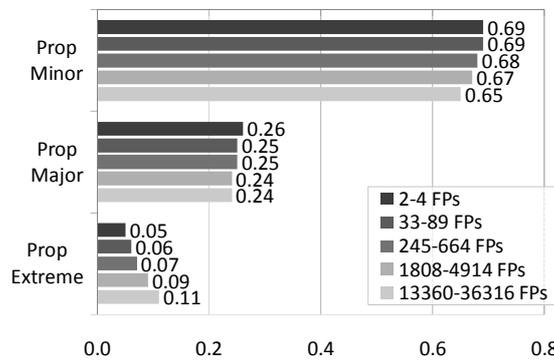


Figure 9-4 Predictions for different *functional sizes*

Case 2 – Impact of uncontrollable factors

Here we analyze the impact of particular uncontrollable factors on predicted proportions of particular types of defects. Figure 9-5 illustrates the impact of *task complexity*. We can observe that it has moderate impact on all proportions of defects. As a task becomes more complex, the model predicts increased *prop major* and *prop extreme* which cause a decrease in *prop minor*.

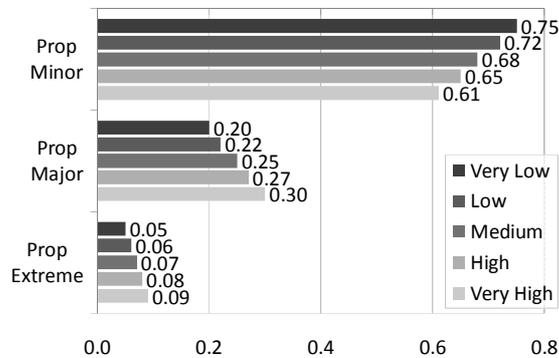


Figure 9-5 Impact of *task complexity* on predicted proportions of defects

Figure 9-6 illustrates impact of deadline pressure on predicted proportions of defects. It has the strongest impact on *prop minor*. DTM predicts that as *deadline pressure* increases *prop minor* should also increase and proportions of all other types of defects should decrease. This could be explained by the fact that with very high *deadline pressure* developers tend to implement the core part of software while there is often little or no time on implementing less important parts of software. But defects in these parts often are annoying to future users because they often touch incomplete user interface and cause minor inconvenience for their work.

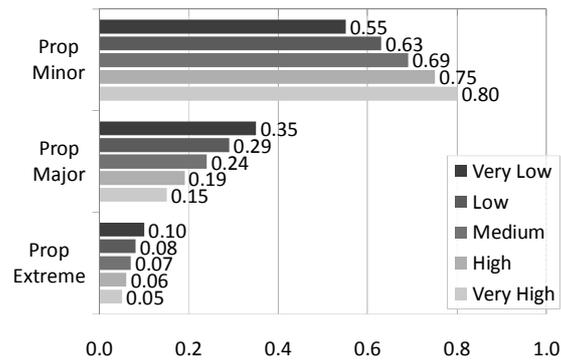


Figure 9-6 Impact of *deadline pressure* on predicted proportions of defects

As detected earlier during data analysis, *package customisation* has some influence on *prop extreme*. Proportions of other types of defects change only in response to changes in *prop extreme*. Because *package customisation* is a complex and difficult task it often leads to increase in *prop extreme*. This influence illustrates Figure 9-7.

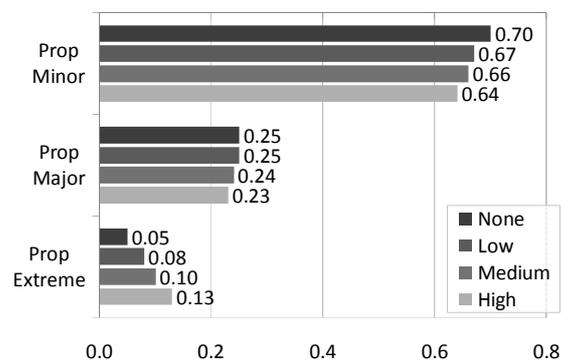


Figure 9-7 Impact of *package customisation* on predicted proportions of defects

Figure 9-8 illustrates impact of *GUI usage* on predicted proportions of defects. The model predicts that it has the highest impact on *prop minor*: as degree of *GUI*

usage in developed software increases *prop minor* also increases. Such behaviour can be explained by the fact that usually in heavily GUI-based systems many minor defects are reported in GUI part of software.

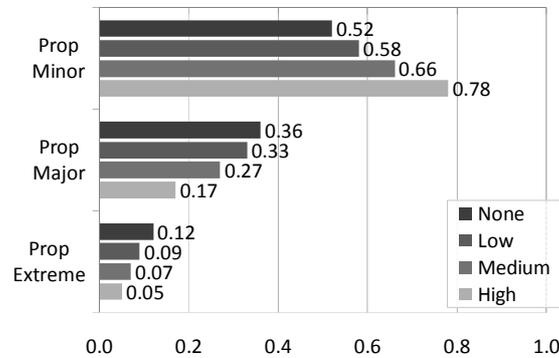


Figure 9-8 Impact of GUI usage on predicted proportions of defects

Case 3 – Combinations of uncontrollable factors

In previous cases we analyzed the impact of individual uncontrollable factors on different proportions of defect types. Here we analyze how different combinations of uncontrollable affect predicted proportions of defects. We compare six hypothetical projects. There are two groups of similar projects: the first three in one group, the last three in the other.

Table 9-6 Summary of analyzed sample projects

| Factor \ Project | Project 1 | Project 2 | Project 3 | Project 4 | Project 5 | Project 6 |
|-----------------------|-----------|-----------|-----------|------------|------------|------------|
| Deadline pressure | Low | Low | Low | Low | Low | High |
| Functional size | 33-89 | 33-89 | 33-89 | 4915-13359 | 4915-13359 | 4915-13359 |
| GUI usage | Low | Low | Low | Low | High | High |
| Package customisation | None | High | High | None | None | None |
| Task complexity | Low | Low | High | High | High | High |

Figure 9-9 illustrates predicted proportions of various types of defects depending on particular combination of uncontrollable factors. We can observe that:

- Project 2 is very similar to Project 1 but involves ‘high’ *package customisation* compared to ‘none’ in Project 2. This difference causes that predicted *prop extreme* is about 2.6 times higher in Project 2 than in Project 1.
- Project 3 is very similar to Project 3 but is assumed to be more complex. This difference causes that predicted *prop extreme* is higher by 0.033 in Project 3 than in Project 2.

- Comparing Project 3 to Project 1 (different *package customisation* and *task complexity*) we can observe that predicted *prop major* is similar in both projects but *prop extreme* is significantly higher in Project 3 than in Project 1.
- Project 5 is very similar to Project 4 but assumes ‘high’ *GUI usage* comparing to ‘low’ in Project 4. This difference cause that the model predicts significantly lower *prop major* and *prop extreme* in Project 5 than in Project 4.
- Project 6 is very similar to Project 5 but assumes ‘high’ *deadline pressure* comparing to ‘low’ in Project 5. This difference causes that predicted *prop major* and *prop extreme* are both lower in Project 6 than in Project 5.
- Comparing Project 6 to Project 4 (different *GUI usage* and *deadline pressure*) we can observe around 66% increase of *prop minor* in Project 6 which caused significant decrease in both *prop major* and *prop extreme*.
- Comparing Project 4 to Project 1 (different *functional size* and *task complexity*) we can observe higher *prop major* and *prop extreme* in Project 4 than in Project 1.

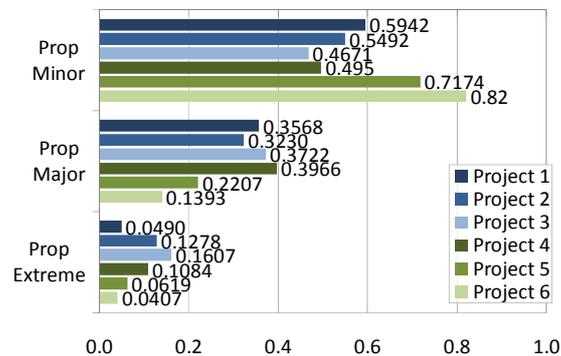


Figure 9-9 Predictions for different combinations of uncontrollable factors

Case 4 – Impact of controllable factors

In this case we analyze the impact of controllable factors describing development process quality on proportions of various types of defects. Figure 9-10 illustrates predictions for most (‘very low’) and least desired (‘very high’) process factors: specification, coding and testing. We can observe that *activity test* has the greatest impact on reducing *prop major* and *prop extreme* and increasing *prop minor*. *Activity specification* and *activity build* have significantly lower impact.

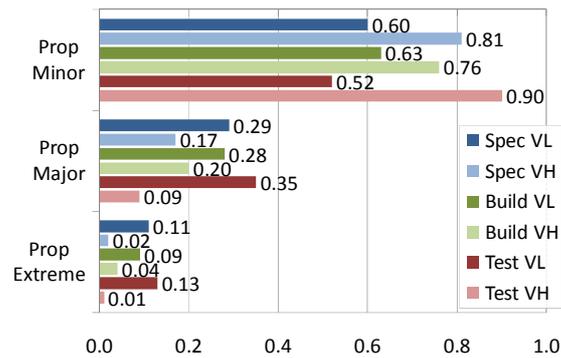


Figure 9-10 Predictions for different states of controllable factors

Case 5 – Combinations of controllable and uncontrollable factors

Here we assume that there are 2 different projects described by various uncontrollable factors (Table 9-7). We analyze how different combinations of controllable factors influence predicted proportions of various types of defects in those 2 projects.

Table 9-7 Summary of two analyzed sample projects

| Factor \ Project | Project 1 | Project 2 |
|-----------------------|-----------|------------|
| Deadline pressure | Low | High |
| Functional size | 12-32 | 4915-13359 |
| GUI usage | None | High |
| Package customisation | None | Medium |
| Task complexity | Low | High |

There are three controllable factors in the model: *activity specification*, *activity build* and *activity test*. We assume that all these process factors change in the same way – when *activity specification* is ‘very low’, also *activity build* and *activity test* are ‘very low’; similarly with other states for these factors.

Figure 9-11 illustrates changes in predicted proportions of various types of defects according to changes in controllable factors. We can observe that:

- Project 2 is expected to contain more *prop minor* but less *prop major* and *prop extreme* than Project 1 when comparing the same combination of controllable factors;
- Differences between projects in predicted *prop major* are higher than in predicted *prop minor* and *prop extreme*;

- Differences between projects in predicted *prop extreme* are very narrow but because Project 2 is significantly larger than Project 1 more extreme defects is expected to be delivered in Project 2.
- For both projects predicted *prop extreme* is expected to vary less when process factors are above average level while it is expected to vary more when process factors are below average level. This would indicate that it is ‘easier’ to do bad software than to do it right.

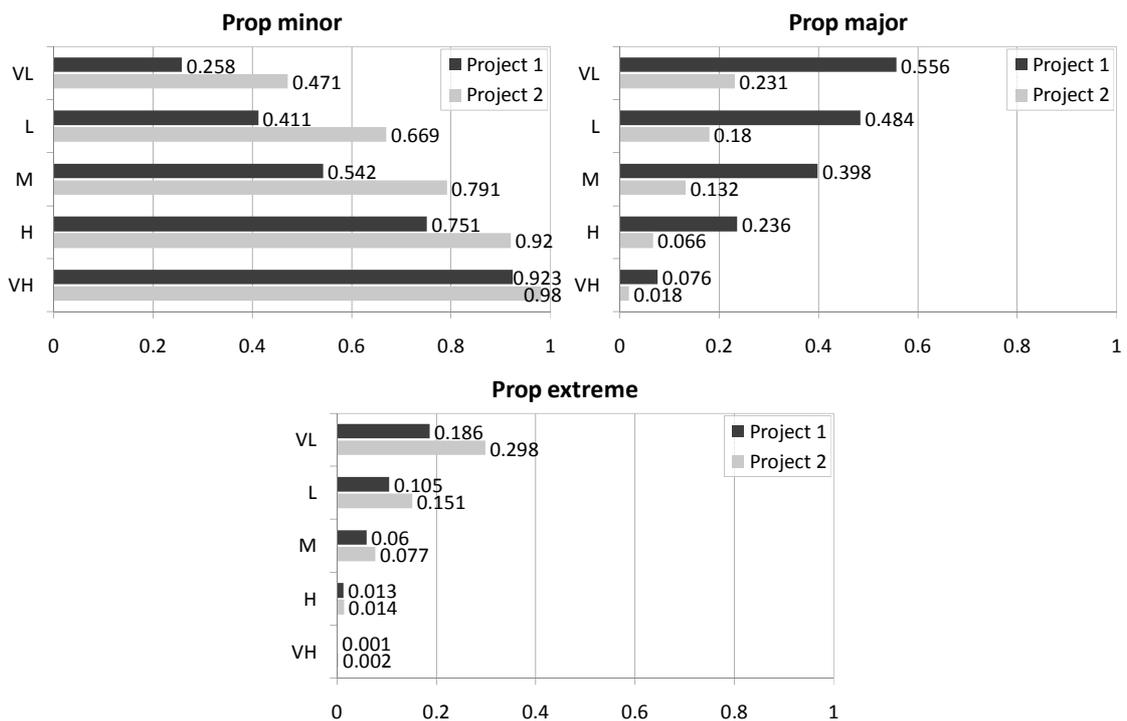


Figure 9-11 Predictions for different combinations of controllable factors

Case 6 – Most and least favourable scenarios

In this case we analyze the impact of most and least favourable combinations of predictors on proportions of each type of defect. These combinations are summarized in Table 9-8. For the purposes of this analysis we assume that the most favourable is to have high value for *prop minor* and low for *prop major* and *prop extreme*. Although in Table 9-5 we point that *functional size* and *package customisation* have no influence on *prop minor* and *prop major* here we use both of these predictors. This is due to the fact that change in *prop extreme* always causes the change in *prop minor* and *prop major* because all proportions need to sum up to 1. Therefore, although we did not find significant impact of these predictors on *prop minor* and *prop major* we still use them in this analysis.

Table 9-8 Most and least favourable scenarios

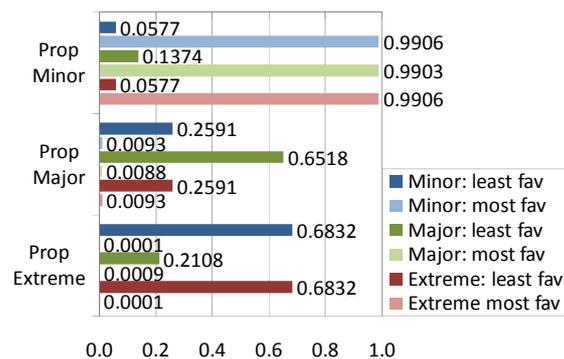
| Dependent & scenario Predictor | Prop minor | | Prop major | | Prop extreme | |
|-----------------------------------|------------------|-----------------|------------------|-----------------|------------------|-----------------|
| | Least favourable | Most favourable | Least favourable | Most favourable | Least favourable | Most favourable |
| Activity build | VL | VH | VL | VH | VL | VH |
| Activity specification | VL | VH | VL | VH | VL | VH |
| Activity test | VH | VL | VL | VH | VL | VH |
| Deadline pressure | VL | VH | VL | VH | VL | VH |
| Functional size | 13360-36316 | 2-4 | 2-4 | 13360-36316 | 2-4 | 13360-36316 |
| GUI usage | None | H | None | H | None | H |
| Package customisation | H | None | None | H | H | None |
| Task complexity | VH | VL | VH | VL | VH | VL |

‘VL’: ‘very low’, ‘H’: ‘high’, ‘VH’: ‘very high’

Figure 9-12 illustrates model predictions for most and least favourable scenarios.

We can observe that:

- least favourable scenarios for *prop minor* and *prop extreme* are the same – also predictions are the same;
- most favourable scenarios for *prop minor* and *prop extreme* are the same – also predictions are the same;
- range for specific proportions of defects depending on scenario is following:
 - for *prop minor*: 0.0577–0.9906,
 - for *prop major*: 0.0088–0.6518,
 - for *prop extreme*: 0.0001–0.6832.

**Figure 9-12 Predictions for most and least favourable scenarios**

Case 7 – Using soft evidence

The cases discussed above assumed that when observation was entered to the variable it meant that we were 100% certain about the value of this variable. Here we analyze the use of soft evidence which enables us entering observation about which there is some uncertainty but about which we have some feeling. Let us assume that we

have a project of around 1000 FP involving ‘low’ *package customisation*, ‘medium’ *GUI usage*, with ‘high’ *task complexity* and ‘medium’ *deadline pressure*.

We analyze the effect of process factors on predicted proportions of various types of defects in the following scenarios:

- all process factors are ‘very low’,
- all process factors are ‘low’,
- expected process factors can be ‘very low’ or ‘low’ – soft evidence,
- all process factors are ‘very high,
- all process factors are ‘high’,
- expected process factors can be ‘very high’ or ‘high’ – soft evidence,

Figure 9-13 illustrates model predictions for analyzed project in the above scenarios. We can observe that

- predictions in scenario assuming soft evidence ‘very low’ or ‘low’ are between predictions in scenarios assuming hard evidence ‘very low’ or ‘low’,
- predictions in scenario assuming soft evidence ‘very high or ‘high’ are between predictions in scenarios assuming hard evidence ‘very high’ or ‘high’.

These results show that soft evidence can be effectively used for ranked nodes (as process factors in our model) where it there is no 100% certainty about particular value of such variable but where 2 or more values are probable.

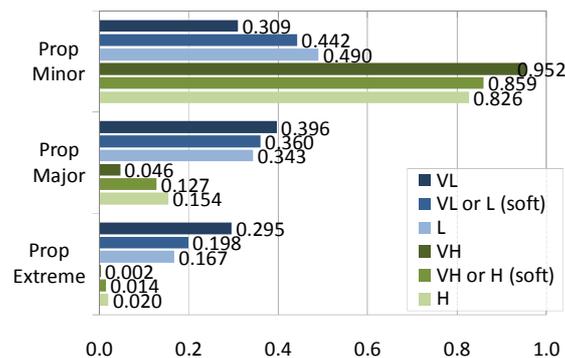


Figure 9-13 Predictions after entering soft evidence

Case 8 – Linking with Productivity Model

In this case we demonstrate how the DTM can be linked with the Productivity Model. Such linking enables predicting not just proportions of defects of specific type but also the number of them. Let us assume that a company has to deliver software of 1000 FPs by using 3000 person-hours which indicates ‘medium’ *deadline pressure*. The

software about to be developed is of ‘medium’ complexity, involves ‘low’ *package customisation* and where *GUI usage* is ‘high’. We know that in the past this company delivered similar type of software by using 2500 person-hours, reaching productivity rate of 0.3 FP per person-hour and defect rate of 0.1 defects per FP.

Assuming all other factors on their average level Productivity Model predicts that we should expect about 102 defects (median) left in software after release. DTM predicts that we should expect the following proportions of defects: 0.79 for *prop minor*, 0.16 for *prop major* and 0.05 for *prop extreme*. We use these predictions distribution to estimate the number of defects of specific type using the extended DTM from Figure 9-3. In this case we get predicted probability distributions with medians around 78 for minor, 17 for major and 5 for extreme defects (Figure 9-14, initial scenario).

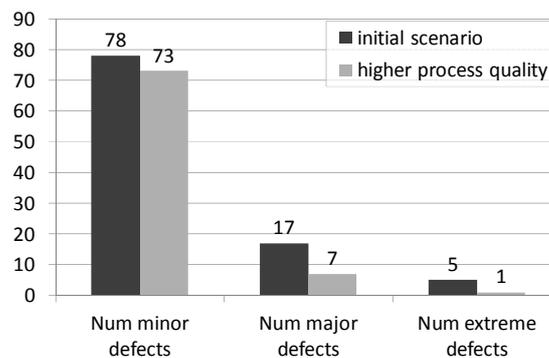


Figure 9-14 Predicted number of defects after linking with Productivity Model

Let us now assume that these numbers are too high to be accepted by this company and its customers. We analyze what impact would it make if process quality on all activities (specification, coding and testing) is higher. Now, the Productivity Model predicts 82 residual defects and DTM predicts 0.91 for *prop minor*, 0.08 for *prop major* and 0.01 for *prop extreme*. After combining these predictions in extended DTM we obtain the following predictions: 73 minor, 7 major and 1 extreme defect (Figure 9-14, scenario: higher process quality). Note that, although *prop minor* is higher with increased process quality, the actual number of minor defects is lower. This is because the increased process quality leads to decrease in the total number of defects.

Sensitivity analysis

In this sensitivity analysis we validate if the model behaves in a way expected by us. Specifically we check if the impact of each predictor is correctly reflected in the

model as we designed it and illustrated in Table 9-5. Figure 9-15 illustrates sensitivity analysis presented as tornado graphs. We can observe that:

- *activity test* has the greatest influence on all proportions of defects;
- *GUI usage* and *deadline pressure* have strong impact on *prop minor* and *prop major*;
- *activity specification* has moderate impact on *prop minor* and *prop major* but strong on *prop extreme*;
- *functional size* and *package customisation* have strong impact on *prop extreme* but not for proportions of other types of defects;
- for *prop minor* and *prop major* there are 3 factors significantly more important than the other while the impact of particular factors on *prop extreme* does not vary so much.

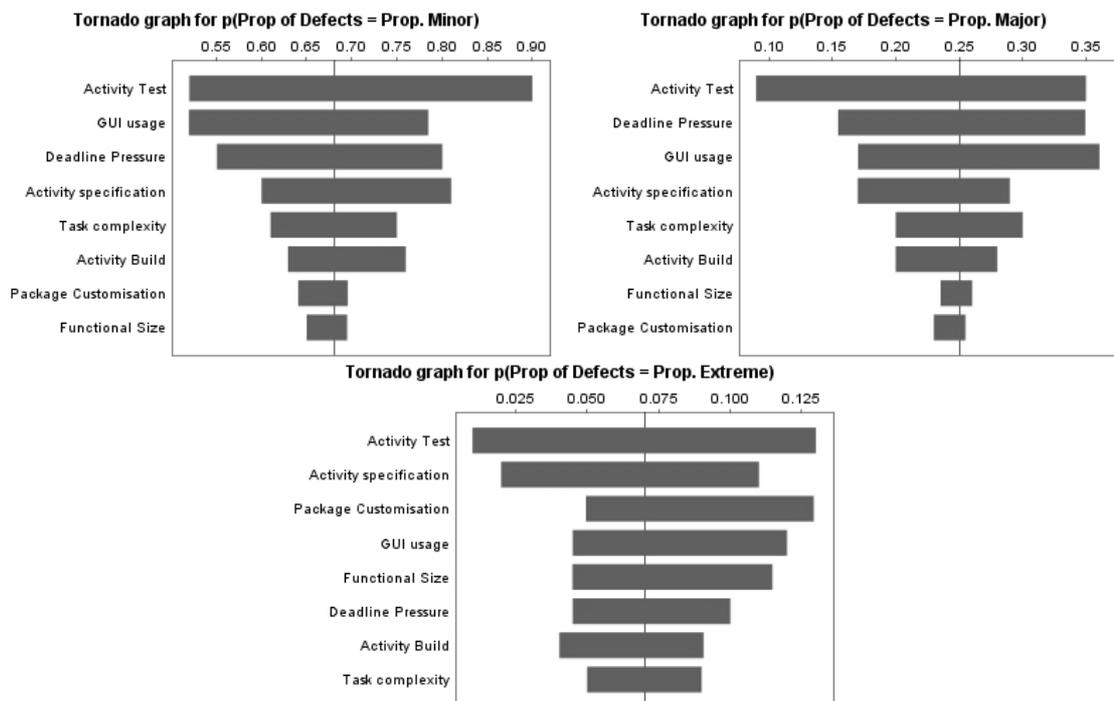


Figure 9-15 Tornado graphs illustrating sensitivity of dependant variables

We accept these results obtained in analyzed cases and in sensitivity analysis. According to our knowledge we can say that they are consistent with reality. Because some of model predictors were not included in analyzed ISBSG dataset we could not validate if their relationships with dependent variables are properly reflected in the model. The impact of these predictors which were included in ISBSG dataset is correctly reflected in our model.

9.5 Summary

The Defect Types Model discussed in this chapter predicts the proportions of defects categorized by their severity. This model incorporates the results from statistical analysis of ISBSG dataset, Mozilla and Apache projects and our expert knowledge. The validation shows that predictions provided by this model can be a useful extension to traditional defect prediction models which typically provide only the total number of defects. Since this model has an NBC structure it is easy to extend the model by adding new predictors, for example code metrics. However adding such predictors shifts the time when the model can be used to a stage of development when the values of such metrics are available. The next chapter discusses another extension to defect prediction – a learning model for predicting number of defects found and fixed in successive testing and fixing iterations.

10 Learning Model for Iterative Testing and Fixing Defects

This chapter extends the discussion on defect prediction by introducing a new model LMITFD (Learning Model for Iterative Testing and Fixing Defects) for predicting the number of defects found and fixed in successive testing and fixing iterations. Unlike previously discussed models, this model does not incorporate known influence of various process factors on number of defects found and fixed. Rather, it *learns* various parameters which reflect the nature of such influences based on past process and defects data. A unique feature of this model is the ability to analyze how various combinations of testing and fixing process factors occurring in the future impact predictions for defects found and fixed in the future. Sections 10.1–10.5 are mainly based on our previous paper [205] and Sections 10.6–10.7 partially on [66].

10.1 Background

A model for iterative testing and defect fixing incorporates the following assumptions about the software development process:

- once the coding process is finished, the testing process starts,
- during testing and defect fixing no new code is added or changed apart from the parts which need to be fixed,
- testing and fixing is an iteration of several phases (sometimes these could be of fixed duration like daily, weekly).

Generally, three types of models for predicting number of defects in a specific testing iteration can be developed:

1. Models following a specific trend line

Before developing a model a trend type has to be determined. Then an expression that is a function of time has to be developed. At the end the parameters for this expression describing the trend line have to be estimated using some of the data from the available dataset. The prediction is made by entering the iteration number as an input to the expression (possibly with other variables describing the testing process).

2. Learning models following a specific trend line

As in the previous type both a trend type and an expression describing the trend have to be determined. But the expression parameters do not need to be estimated

outside the model. Rather the values of early observations of defects found are entered in the model. Then the model learns its parameters – in BNs this is done using back-propagation. As in the previous type, the prediction is made by entering the iteration number as an input to the model (also possibly with other variables describing the testing process).

3. Learning models without the specific trend line assumption

In this type of model there is no assumption about the type of trend line followed by data. It can be modelled as a network – set of variables influencing each other. The unknown values of variables are learnt after entering observations for number of defects found in early testing iterations. The prediction is made after generating more instances of single iterations and linking them in the same way as they were defined for learning.

We analyze the possibility of developing a model of the latter type – since this imposes the fewest assumptions. Furthermore, we want to expand the model by other actions and consequences related to testing such as: fixing defects and introducing new defects as a result of imperfect fixes. To learn the variations in defects found we incorporate various process factors, such as effort and process and people quality.

10.2 The datasets

In our study we have initially used some of the publicly available datasets of projects developed in cooperation with NASA [165]: JM1, KC1, PC1, PC3, PC4. These were the only NASA datasets that met the following criteria:

- defects have an assigned a date of discovery – datasets without this information were excluded as we cannot calculate the aggregated number of defects found in a testing iteration,
- the values follow some kind of a trend – there may be some deviations from the trend line but generally some kind of a trend is followed,
- the testing phase lasted at least 20 iterations – so that our model can learn using some early observed values and makes predictions.

Typically, the number of defects found in a particular testing iteration follows one of the trend lines illustrated on Figure 10-1 [130 p. 217]. The line ‘A’ reflects the situation where number of defects found in successive iterations decreases (‘exponential’). The explanation of this is that after finding some of the defects it is

harder to find the new ones. The line 'B' ('delayed S') reflects the situation when the number of defects found per iteration increases at the beginning and after reaching its peak slowly decreases towards 0. The initial increase in the number of defects found in successive testing iterations is explained by the fact that early in testing the knowledge of the software to be tested is relatively low. After familiarisation testers are more easily able to find defects. But at some point their knowledge reaches saturation point and since the remaining defects are increasingly hard to find. The line 'C' ('inflection S') reflects a similar situation to line 'B' but with a later and sharper peak.

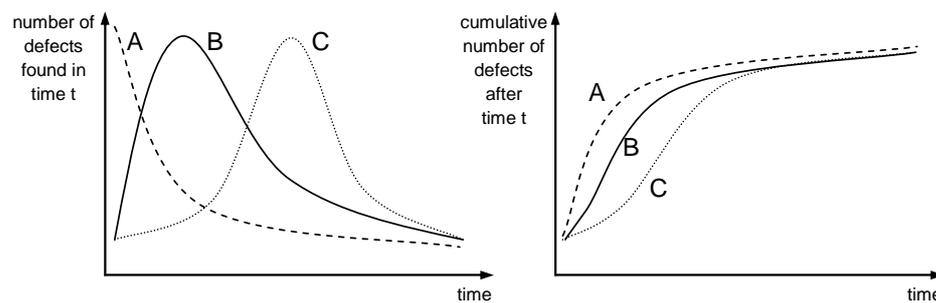


Figure 10-1 Different shapes of trend lines for number of defects found depending on time; adopted from [130 p. 217]

We decided to perform an analysis for testing iterations of 1 month duration (the datasets for shorter durations were too noisy meaning the model would not be able to learn any reasonable trend). The monthly defect data from NASA generally follow a 'delayed S' trend line (Figure 3-2). However, there are some values far away from the general trend line in each dataset. The variation between subsequent iterations is much smaller in monthly aggregations than in the case of weekly data.

10.3 Basic learning model for iterative testing

In this section we discuss the basic Learning Model for Iterative Testing (LMIT). We use the term 'basic' when referring to this model because of its simplicity – it does not capture any causal factors influencing *testing effectiveness* because the NASA dataset did not contain such data. The model attempts to learn the number of *residual defects* remaining after past testing iterations and use this estimate to predict the number of defect likely to be found in successive future testing iterations. The model consists of the following parts (Figure 10-2):

1. **Priors** – This subnet contains initial (prior) probability distributions for variables that are to be learnt later: *testing effectiveness* and *residual defects*.
2. **Learning** – This subnet contains sets of variables linked together as a single BN. Its purpose is to learn the unknown *residual defects* after each iteration and *testing effectiveness* in each iteration.
3. **Prediction** – The first k iterations (e.g. k as illustrated on Figure 10-2) are used for learning the model – we have to enter observations for defects found in the first k testing iterations. Starting from iteration $k+1$ the model predicts the number of *defects found* which are likely to occur in each future testing iteration. It also predicts the *testing effectiveness* in each testing iteration and number of *residual defects* remaining after each future testing iteration. For reasons of computational efficiency, this part of the model is a Dynamic Bayesian Net (DBN) with testing iterations linked sequentially (not as a single BN object). Here the model does not learn anything from the data, but rather uses the previously learnt *testing effectiveness* and number of *residual defects* to predict the number of *defects found* in the future.

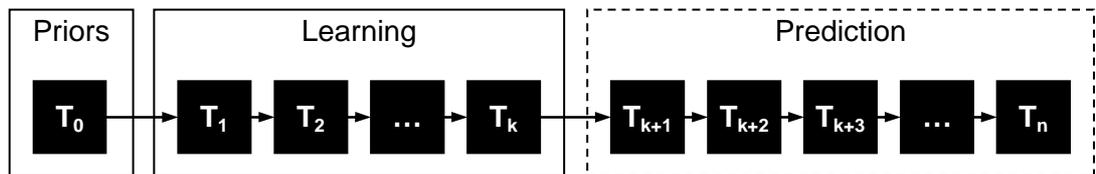


Figure 10-2 General structure of the Learning Model for Iterative Testing

The detailed structure of the basic iterative testing model is illustrated on Figure 10-3 with the expressions used in the model listed in Table 10-1.

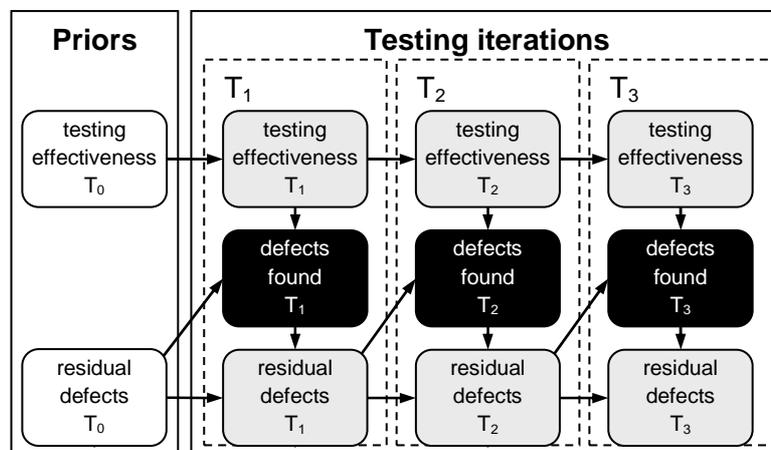


Figure 10-3 Detailed structure of LMIT

Table 10-1 Expressions in the LMIT

| Node | Expression |
|-----------------------------|---|
| testing effectiveness T_0 | Uniform(0, 1) |
| residual defects T_0 | Normal(1000, 1000000) |
| testing effectiveness T_i | TNormal(<i>testing effectiveness</i> T_{i-1} , 0.001, 0, 1) |
| residual defects T_i | Max(0, <i>residual defects</i> T_{i-1} - <i>defects found</i> T_i) |
| defects found T_i | <i>testing effectiveness</i> T_i * <i>residual defects</i> T_{i-1} |

We have tested the model with various NASA datasets. The results using the PC1 dataset are illustrated on Figure 10-4. Predictions using other datasets shared similar features as with the PC1.

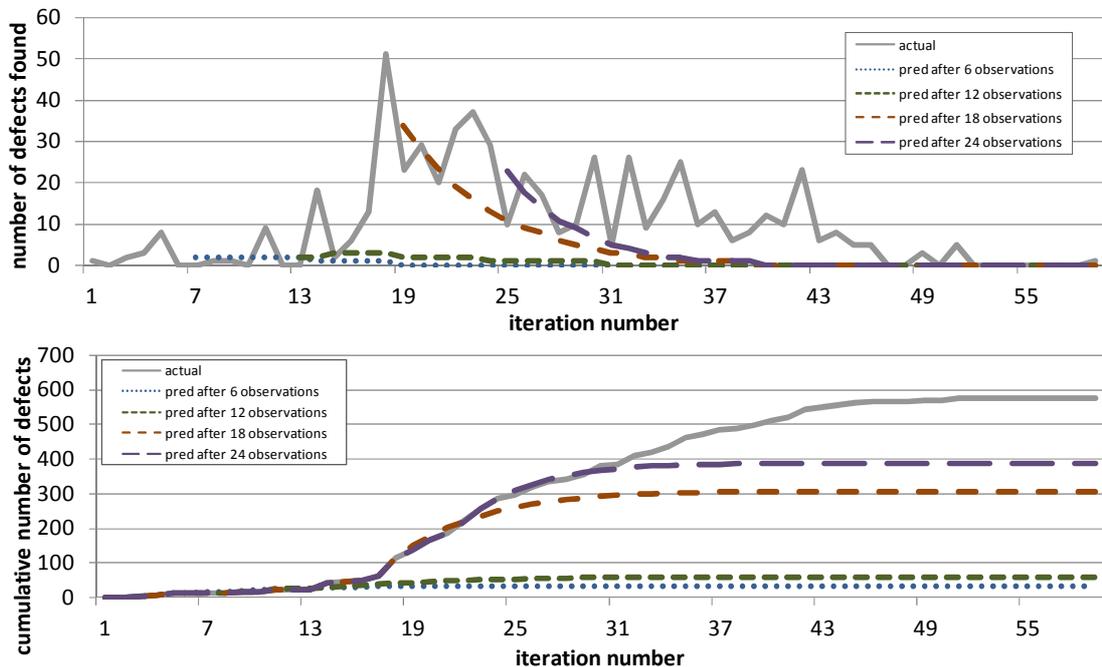


Figure 10-4 Initial prediction results from LMIT with PC1 dataset

10.4 Lessons learned from basic model creation and validation

1. Influence of the prior distribution in residual defects

There are two important parameters in the expression in this variable: its mean and the variance. Both of them may have a significant impact on the predictions given by the model. Ideally we would like to have a point value distribution in this node reflecting the true number of residual defects with no variance (certain value). But we know it is impossible to estimate the number of *residual defects* with 100% accuracy using any method. Still, it is important to set the distribution of this node with a mean as close to the real number of *residual defects* and the variance as low as possible. Setting

a variance to a very low value means that the model has more ‘trust’ in the entered distribution in this variable than the observations entered for *defects found* in some first testing iterations. This is especially important when the mean value in this variable has been set far from the real number of the residual defects. In such cases the model under- or overestimates the predicted number of *defects found* (in the iterations following those for which we entered observations in *defects found*) and the number of defects remaining after a specific testing iteration.

2. Influence of the variance in expression in *testing effectiveness*

The prediction for number of *defects found* in future testing iterations depends on the value of the variance entered in the variable *testing effectiveness*. If the variance entered is low (e.g. 0.0001) the model predicts that the defects will be found over a longer period of time. If the variance is high (e.g. 0.05) then in many scenarios the model predicts that few or no defects will be found in the testing iterations following the last iteration with entered observation in the number of *defects found*. The reason for such contrasting predictions is that in the second case (higher variance) the model believes that the majority of the defects have already been found during past testing iterations. Unusually higher values in the number of *defects found* in specific iteration are explained in the model by the higher *testing effectiveness* in a particular iteration. Such explanations can be given by the model because with higher variance in the equation the model can ‘understand’ higher variation in the real *testing effectiveness* which might have been caused by allocating more effort and/or more testers taking part in the process of finding defects. In the scenarios that we tested we found that the most reasonable predictions can be given when the variance is set to value around 0.001.

3. Influence of the prior distribution in *testing effectiveness*

The variable *testing effectiveness* reflects the probability of finding a defect in a specific testing iteration. Because of this assumption it must be within the range from 0 to 1. But without any other process factors in the model we cannot make any further assumptions about its value. For example, we cannot say if the value of 0.05, meaning that we should expect around 5% of the defects to be found during specific iteration, is high or low. It depends on the granularity of the data we use in our model. If the testing iteration is brief, e.g. one day, then such a value appears to be attractively high. On the other hand, if the testing iteration is long, (e.g. three months), than it does not seem so

attractive anymore. Clearly, the value also depends on the software itself, especially the size which influences the number of defects the most. The more *residual defects* in the software the less likely it is that the testers will find the majority of them in a single iteration. It all means that we cannot make any assumption about the prior probability distribution in the *testing effectiveness*. That is why we entered the Uniform distribution over the range from 0 to 1 in which any of the values within this range is equally likely to happen. After entering observations in *defects found* in some early testing iterations the distributions in this variable in all iterations are revised – they are not that flat anymore.

4. Influence of the expression in *defects found*

We have tested our model the number of *defects found* expressed as a Binomial distribution with number of trials equal to remaining residual defects before the specific testing iteration and with probability of success equal to the *testing effectiveness* in the current iteration. Such a distribution has been used in earlier models for predicting number of defects [68, 69, 75]. We have also tested this model with number of defects expressed simply as a multiple of *testing effectiveness* and remaining *residual defects* before the specific testing iteration. We found that with the simpler multiplication the predictions were slightly more accurate and calculation times significantly shorter (by around 50-70%).

5. Prediction accuracy

The basic version of the model does not give accurate predictions for number of *defects found* in future iterations. This can be explained by the lack of variables in the model which would be able to explain the variation in the number of defects found between iterations, i.e. why the *testing effectiveness* varies between subsequent iterations so much. However, given the observations entered in the model, we actually get reasonable predictions. The model is highly inaccurate on observations for small number of iterations (<10). But if the values entered are very low (as they usually are in the analyzed datasets) why should the model predict the increase of number of *defects found* in future iterations without any information about the number of residual defects and, more importantly, the factors affecting the *testing effectiveness*? To have improved and useful predictions we need the extended model with additional variables.

10.5 *Extended learning model for iterative testing*

To improve the model's accuracy and realism we added new variables to the model. Thus we created an Extended Learning Model for Iterative Testing (ELMIT). Two factors seem to be most influential on the *testing effectiveness*:

- *testing effort*,
- *testing process and people quality*.

These new variables are not expressed on an absolute scale. Rather, they reflect the ratio of the value in the given iteration to the value in the previous iteration. We assume that users can provide observations for these variables in the first iterations used to learn the model. For example, if the effort in the current iteration increased by 20% compared to the previous one the users should enter a value '1.2'.

In certain cases it may happen that, although both *testing effort* and *process and people quality* increase, we observe lower *testing effectiveness*. Therefore, we added yet another variable *testing bias* which reflects the aggregation of all negative factors occurring in the specific testing iteration. This includes, for example, the need to prepare additional test cases (which causes the situation whereby some effort is not effectively used on purely finding defects) or testing a component that was not tested much before (and testers need to learn this component to know where and how it should be tested).

We added an intermediate node *testing effectiveness change* T_i which aggregates the testing factors. It reflects the extent at which the testing effectiveness changed in the given iteration compared to the previous one. This node is a parent node for *testing effectiveness* T_i together with the *testing effectiveness* T_{i-1} and *testing effectiveness limit*. Introducing the latter ensures that the change of testing factors to some degree does not cause the same degree of change in testing effectiveness but lower (the law of 'diminishing returns'). The structure of the single iteration in the extended model is illustrated on Figure 10-5. Table 10-2 contains the expressions in the new or updated nodes.

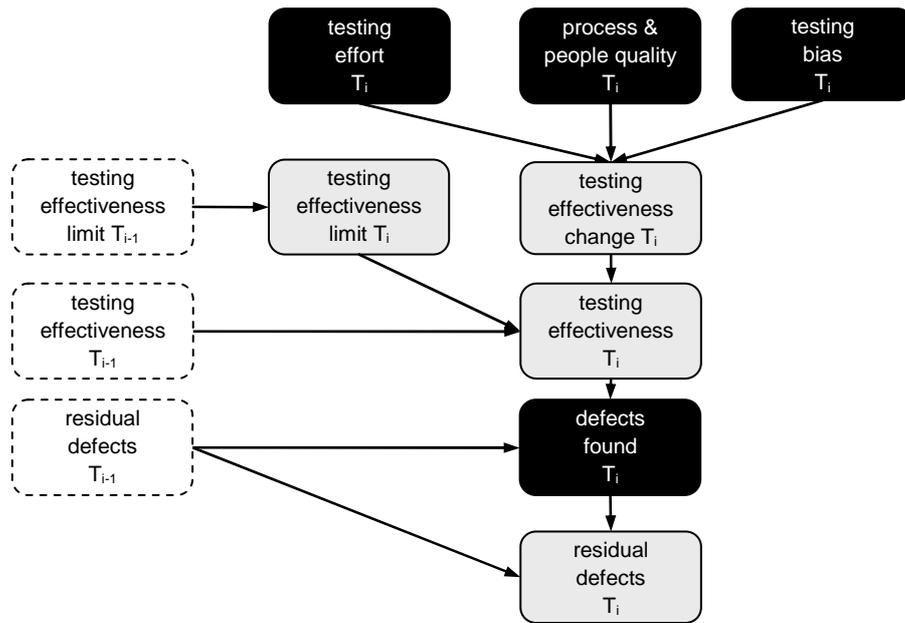


Figure 10-5 Detailed structure of the Extended Learning Model for Iterative Testing (single iteration)

Table 10-2 Expressions in the Extended Model for Iterative Testing

| Node | Expression |
|------------------------------------|---|
| testing effort T_i | Normal(1, 0.1) |
| process and people quality T_i | Normal(1, 0.1) |
| testing bias T_i | Normal(1, 0.1) |
| testing effectiveness change T_i | Normal(<i>process and people quality</i> $T_i * ((\textit{effort } T_i - 1) * 0.8 + 1)$ / <i>bias</i> T_i , 0.01) |
| testing effectiveness limit T_i | <i>testing effectiveness limit</i> T_{i-1} |
| testing effectiveness T_i | <i>testing effectiveness</i> $T_{i-1} * ((\textit{testing effectiveness change } T_i - 1) * \textit{testing effectiveness limit } T_i) + 1$ |

The observations in the extended model should be assigned for all known variables (*testing effort*, *process and people quality*, *testing bias*, *defects found*) in the first iterations used for learning the model. For prediction – only the observations for *testing effort*, *process and people quality* and possibly for *testing bias*. These observations are treated as anticipated values – planned to be achieved. Based on these observations the model predicts the number of *defects found* in future testing iterations.

The NASA datasets used earlier do not contain any values reflecting testing effectiveness like *testing effort* or *process and people quality*. Therefore we do not use these datasets to validate an extended version of the model. Instead we generated a sample dataset solely for the purpose of validating the extended version of the model. When generating this dataset we set an initial number of *residual defects* and generated random values for changes in *testing effort*, *process and people quality* and *testing bias*. Then, using these values and equations as in the model we generated the values for the

number of *defects found* in each iteration. Then we manually changed some of the values to ensure that the shape of the trend line follows the one typically occurring in real projects (initial increase of the number of defects found, followed by a decrease towards 0).

We used this dataset by entering the values describing each iteration in the model. We did not enter the precise value of number of *residual defects* as in practice this would not be possible. Also, the values for *testing effort*, *process and people quality* and *testing bias* we entered with lower precision – in the dataset they were up to the second decimal place, in the model we entered these values rounded to the first decimal place. This is to simulate the accuracy of estimating these factors in real projects which will never be 100% accurate.

The aim of this validation is to find out how fast the model learns the *testing effectiveness* and real number of *residual defects* that were in at the beginning of the testing process. The results are illustrated on Figure 10-6.

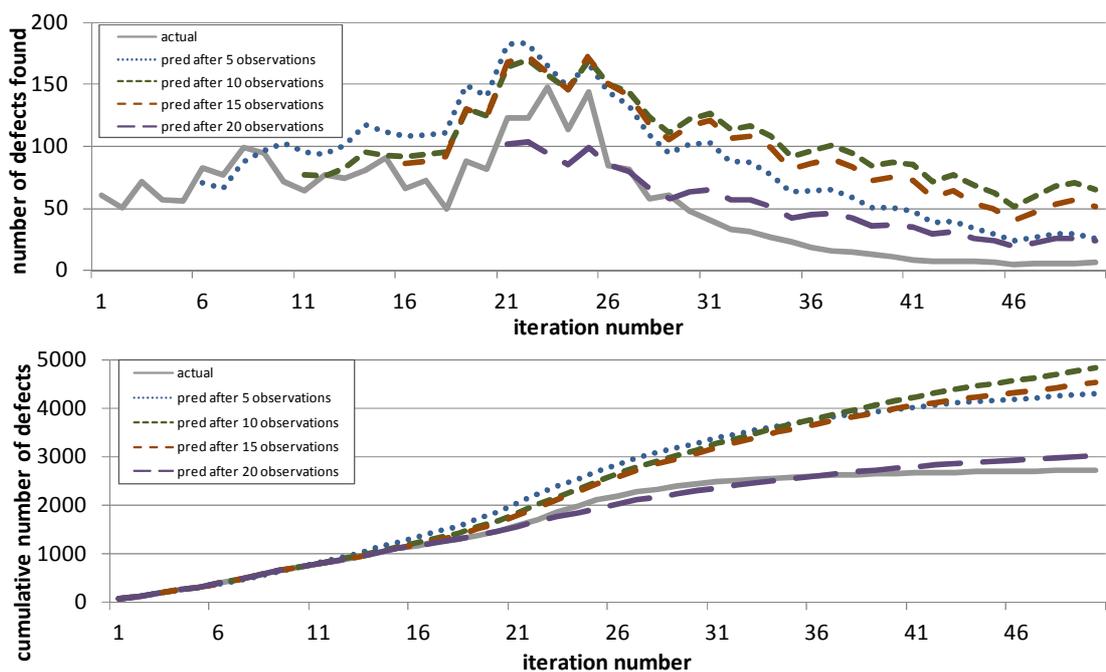


Figure 10-6 Prediction results from ELMIT with semi-randomly generated dataset

We can observe that the prediction accuracy for the first couple of iterations following the iterations used for learning is generally acceptable in every case. However, predicting the number of *defects found* far in the future requires more iterations used to learn the model.

10.6 Model for iterative testing and fixing defects

ELMIT captures only the testing phase without incorporating any factors related to fixing the defects. Because of this it also cannot incorporate any variables related to *defects inserted* – they are inserted during fixing defects detected earlier due to imperfect fixing process. The Learning Model for Iterative Testing and Fixing Defects (LMITFD) is a further enhancement of the iterative model for testing process. Thus, the main two enhancements are:

- incorporating defects fixing process,
- incorporating inserting new defects as a result of imperfect fixes.

The structure of the core part of LMITFD is illustrated on Figure 10-7 and details of testing and fixing processes subnets of this model on Figure 10-8. Full definition of model structure contains Appendix E.

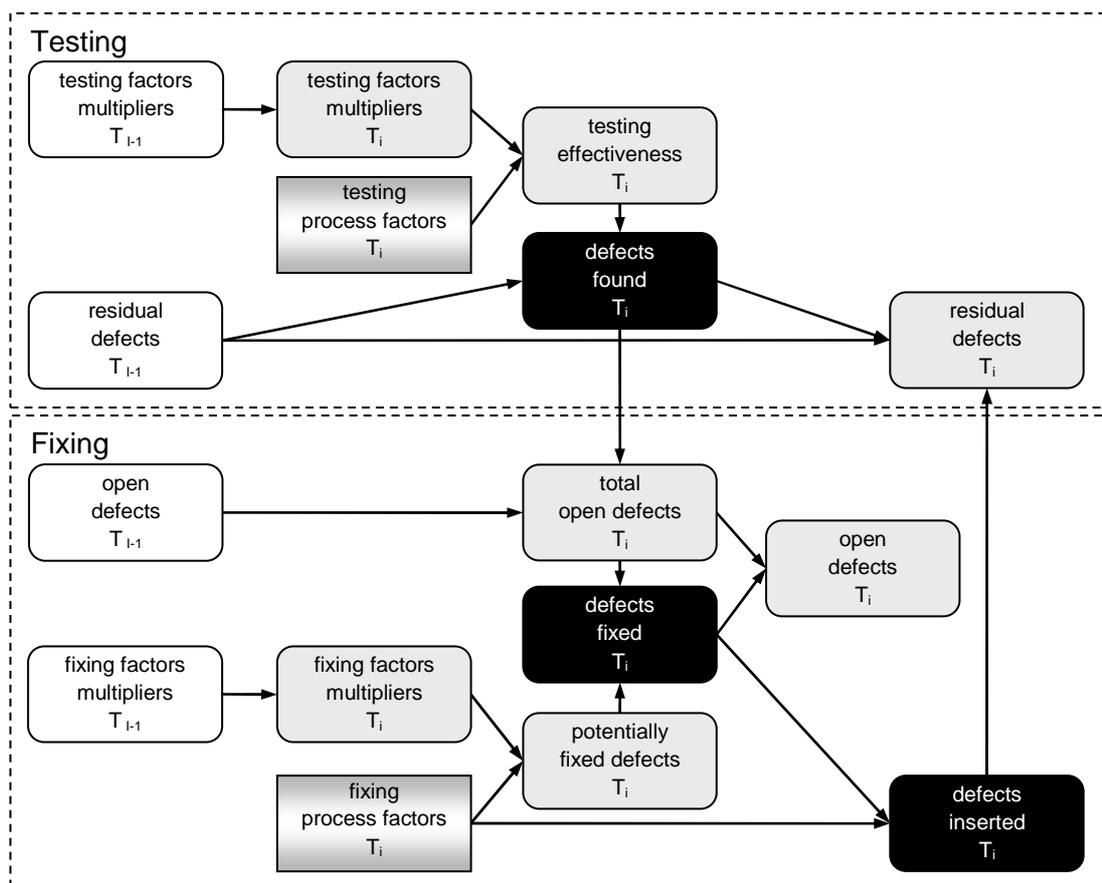


Figure 10-7 Structure of the Learning Model for Iterative Testing and Fixing Defects (single iteration)

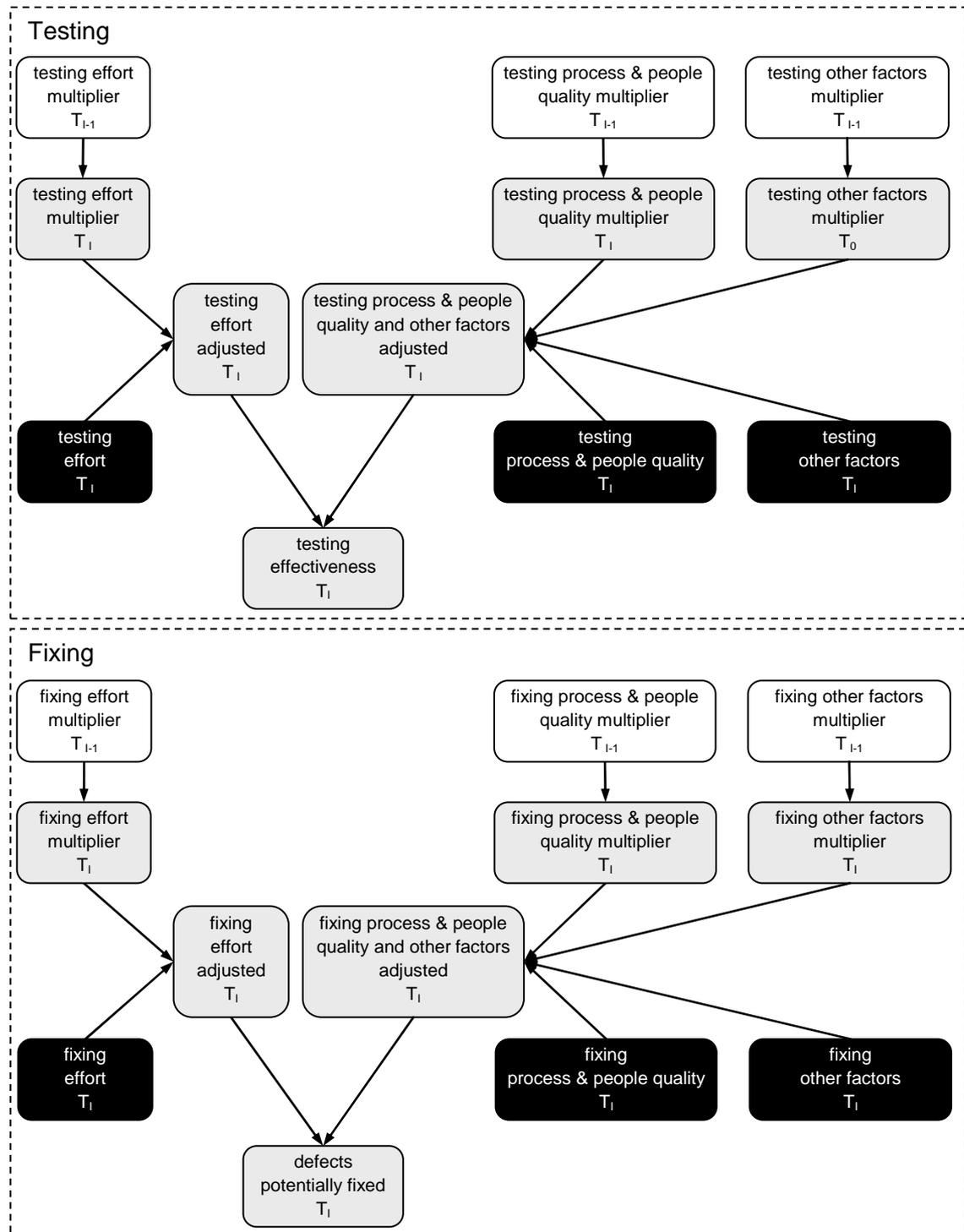


Figure 10-8 Detailed structure of testing and fixing process in Learning Model for Iterative testing and Fixing Defects (single iteration)

This model contains the following changes compared to previously built ELMIT:

- Introduced *multipliers* learnt by the model for each process factor (*effort*, *process and people quality* and *other factors*). With this change the model does not assume fixed impact of process factors on *testing effectiveness* and *potentially fixed defects* encoded in the model.

- Initial number of *residual defects* is defined as Normal(400, 100000) – it has a mean of 400 and standard deviation of about 316 defects. In the dataset used to validate this model the actual number of *residual defects* is assumed to be 300 with total number of *defects found* and *defects fixed* during the whole testing and fixing process is 334. This means that 34 defects are assumed to be inserted as a result of imperfect fixing. We believe that in reality it is possible to provide a prior for *residual defects* before starting the testing process with such accuracy using one of the defect prediction models discussed in this thesis. We have also tested other types of distributions for *residual defects*: beta, gamma, lognormal and triangular. Our tests confirmed that the Normal distribution provides the most accurate predictions, but differences in predictions using other types of distributions were narrow.
- Variable *testing bias* from the previous model renamed to *testing other factors* in the current model. Similarly, we introduced *fixing other factors* in fixing process subnet. This change in variable name removes possible confusion about what we actually mean by *bias*. These *other factors* now incorporate all factors other than *effort* and *process and people quality* which may be known to affect testing and fixing processes.
- Process factors (*effort, process and people quality* and *other factors*) are expressed on an absolute scale. *Effort* is a number expressed in custom units of measurement. So a value ‘100’ can mean 100 person-hours, person-days or any other unit as long as the same unit is used throughout. *Process and people quality* and *other factors* are expressed on 7-point ranked scale from ‘lowest’ to ‘highest’.
- The model contains two variables reflecting defects which have been found but not yet fixed: *open defects* (defects found but not fixed after specific iteration) *total open defects* (defects found but not fixed after previous iteration plus defects found in the current iteration – prior to fixing).
- *Defects inserted* depend on the number of *defects fixed* in given iteration and fixing process effectiveness (fixing process and people quality and other factors). The model assumes that number of *defects inserted*:
 - increases as *defects fixed* increase and

- decreases as *fixing process and people* and *fixing process and people* increase.

The motivation for such relationships comes from our belief that as more defects are fixed the more likely it is that new defects are inserted as a result of imperfect fixing. This number of defects inserted should increase if the fixing process is poor.

- *Defects fixed* depends on the number of *potentially fixed defects* (how many defects could be fixed given specific effort and process data) adjusted by *total open defects*. This relationship reflects the fact that no matter how good the fixing process is you cannot fix more defects than you have previously found but did not yet fix.

10.7 Validating learning model for iterative testing and fixing defects

Prediction accuracy

Our aim in this analysis is to validate the model's accuracy against a semi-randomly generated dataset. We generated this dataset in the same way as for ELMIF with the only differences as follows:

- We shortened the testing and fixing process down to 30 iterations. This model is much more complicated than ELMIF and takes about 2 hours to calculate with this smaller dataset.
- This dataset contains number of defects fixed and inserted as well as fixing process data.

Data used to learn the model contained both process data and number of *defects found* and *defects fixed*. Data used in prediction contained only process data. Figure 10-9 illustrates predictions from LMITFD for selected number of iterations used to learn the model: 2, 5, 10 and 15. We can observe that after two testing and fixing iterations used to learn the model's parameters the model predicts the overall trend line for *defects found* close to the actual but predictions are not accurate in specific iterations. Starting from 5 iterations we get reasonably accurate predictions for *defects found*. This accuracy generally increases as the number iterations used to learn model's parameters increases. But there are some exceptions. For example, predictions with 10 iterations are worse than 5. This can be explained by the higher fluctuations of number of actual *defects found* in the dataset within the first 10 iterations than within the first 5

iterations. Such fluctuations mean that this model needs more observations to learn the *multipliers* and *residual defects* properly.

Similar conclusions can be drawn for *defect fixed* as for *defects found*. However, predicted *defects fixed* are generally less accurate than predicted *defects found*. This is a result of uncertain (predicted) number of *defects found* which impact on number of *defects fixed*. So, to analyze *defects fixed* the model has to learn more parameters (*defects found* can be treated as a parameter from this point of view).

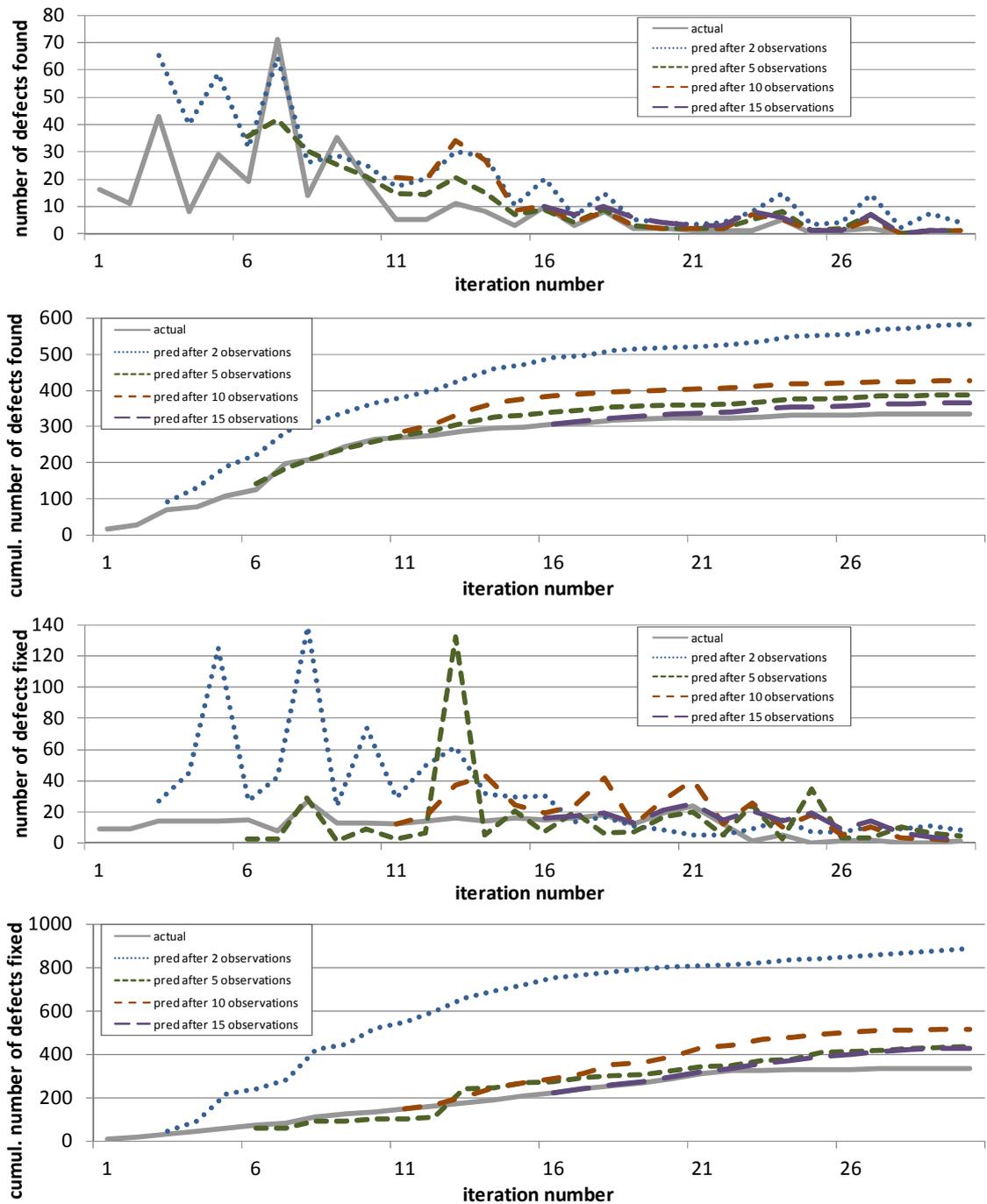


Figure 10-9 Prediction results for the LMITFD with semi-randomly generated dataset

Figure 10-10 illustrates predicted total number of *defects found* and *defects fixed* predicted after using different number of testing and fixing iterations to learn the model. First we used a single learning iteration with 29 iterations where values of defects found and fixed were predicted. This was followed by two iterations for learning and 28 for prediction, and so on. Also these results confirm that, after relatively few observations (5), the model predicts the total number of *defects found* with the reasonable accuracy. Similarly for *defects fixed* but, as indicated earlier, estimates for *defects fixed* are less accurate because of higher uncertainty in factors influencing them, especially in *defects found*.

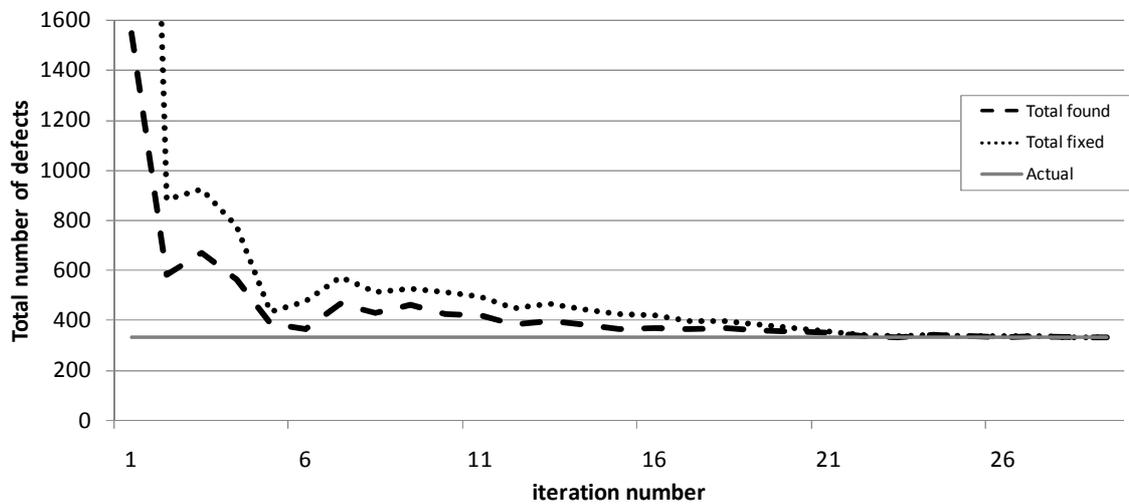


Figure 10-10 Predicted total number of *defects found* and *defects fixed* depending on different number of iterations used to learn the model

Figure 10-11 illustrates the values of relative errors in predicted total number of *defects found* and *defects fixed* as estimated after a different number of learning iterations. This relative error is defined as shown in Equation 39.

$$relative\ error = \frac{|total\ predicted - total\ actual|}{total\ actual} \quad (39)$$

These results confirm that after only 5 learning iterations, the model predicts the total number of *defects found* to within a 0.16 relative error of the actual value and predicts total number of *defects fixed* to within a 0.30 relative error of the actual value. Predictions then become less accurate because of higher fluctuations in actual number of *defects found* in the dataset. But from 8 learning iterations the accuracy again increases (relative error decreases). These results show that the model is capable of

learning its parameters after very few iterations and then generates predictions with reasonable accuracy.

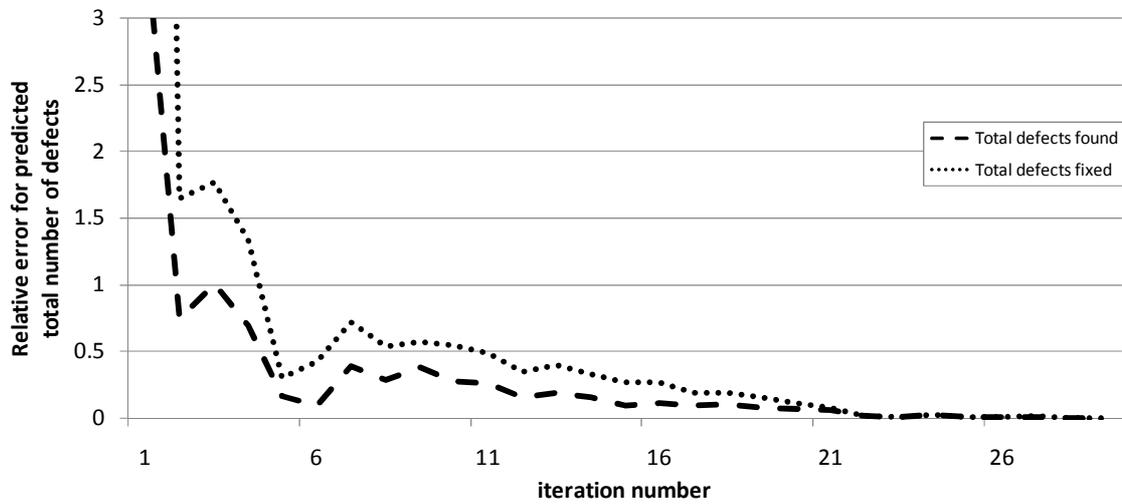


Figure 10-11 Relative errors in predictions for total number of *defects found* and *defects fixed* depending on number of iterations used to learn the model

Figure 10-12 illustrates predictions for different values of the parameter *maximum number of iterations* used by DD algorithm. Again, we used 5 testing and fixing iterations to learn model parameters. We observed similar differences in predictions for different number of testing and fixing iterations used in learning the model. We can see that initially as the value of this parameter increases, prediction accuracy also increases – predicted number of *defects found* and *defects fixed* are closer to actual values.

However, after reaching the threshold value of *maximum number of iterations* the model starts to produce unexpected results. For some future testing and fixing iterations predicted number of *defects found* and *defects fixed* are many times higher than actual (e.g. iteration 7 for *defects found*). There is an explanation for this behaviour. Generally, higher value of *maximum number of iterations* causes the model to learn its parameters (*multipliers*) more quickly. But, although with learnt *multipliers* most predictions are more accurate, these *multipliers* have learnt values different from those assumed while generating the dataset. After reaching certain combinations of process factors (in predicted iterations) the model actually adjusts the values of these multipliers to ensure that *testing effectiveness* does not fall outside the range [0, 1]. These adjusted *multipliers* are then used in successive iterations. So, even though the model should not adjust learnt *multipliers* during prediction, it actually still does – more often with higher *maximum number of iterations*.

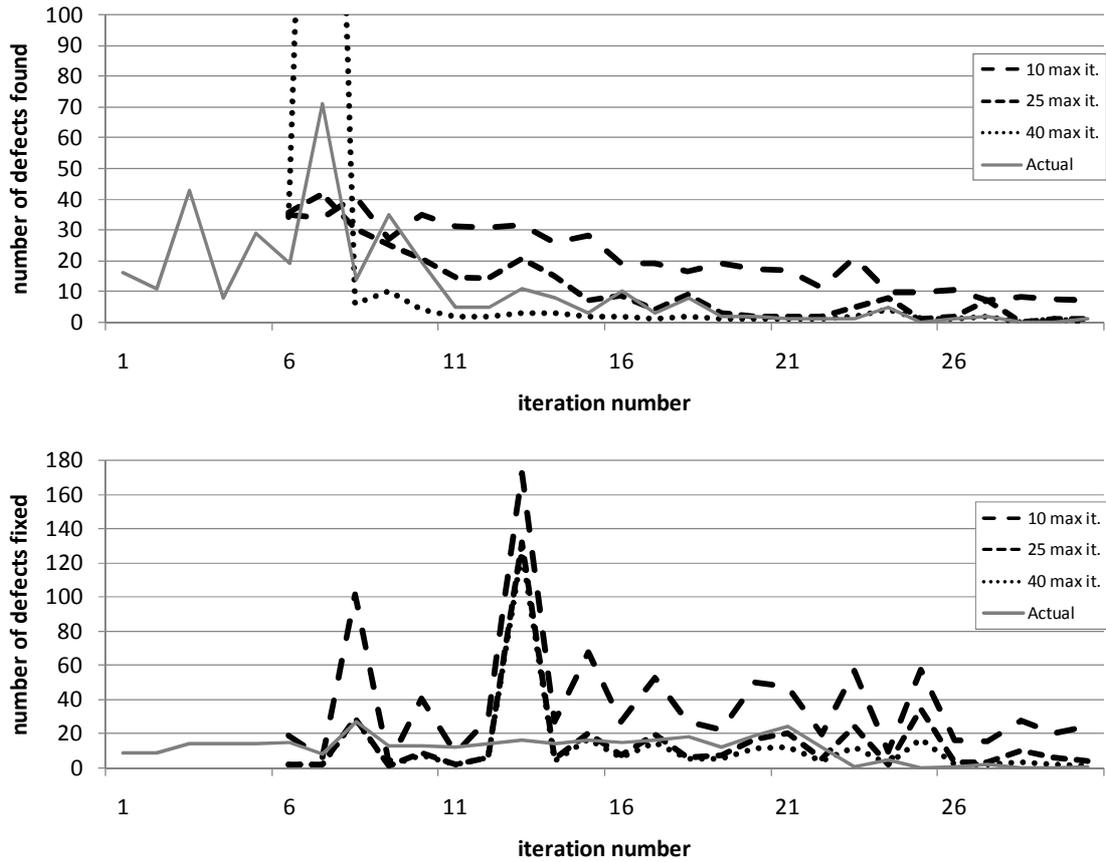


Figure 10-12 Impact of parameter *maximum number of iterations* for DD algorithm

Table 10-3 illustrates the ‘cost’ of increased accuracy in predictions in terms of model execution time. A value of around 25 for *maximum number of iterations* seems to provide an optimal balance between accuracy and speed.

Table 10-3 Comparison of calculation times for LMITFD depending on different *maximum number of iterations*

| Value of Maximum Number of Iterations | Shortest time | Longest time |
|---------------------------------------|---------------|--------------|
| 10 | 0:05:30 | 0:08:05 |
| 25 | 0:18:45 | 0:34:30 |
| 40 | 1:08:50 | 2:17:45 |

Analyzing different combinations of process factors

In this analysis we assume that some process factors used in the prediction stage (not learning) are different from the original dataset. This demonstrates how different combinations of process factors can influence predicted number of *defects found* and *defects fixed* even if the model was learnt using the same data. The results show that such a model is more powerful than classical reliability growth models which do not

incorporate process factors and thus do not enable such ‘what-if’ analysis. The analysis involves 30 iterations (5 testing and fixing iterations to learn the model and 25 to predict).

First, we analyze the effect of halving and doubling testing and fixing effort (Figure 10-13). As expected the model predicts that more defects can be found and fixed with increased effort. Differences in predicted *defects found* and *defects fixed* are higher when these predicted values are also higher, for example in iterations 13, 23 and 27 for *defects found* and in iterations 8, 13 and 25 for *defects fixed*. However, predicted differences in *defects found* and *defects fixed* are lower than we expected. This would indicate that there are other process factors which impact more on predicted *defects found* and *defects fixed* than effort.

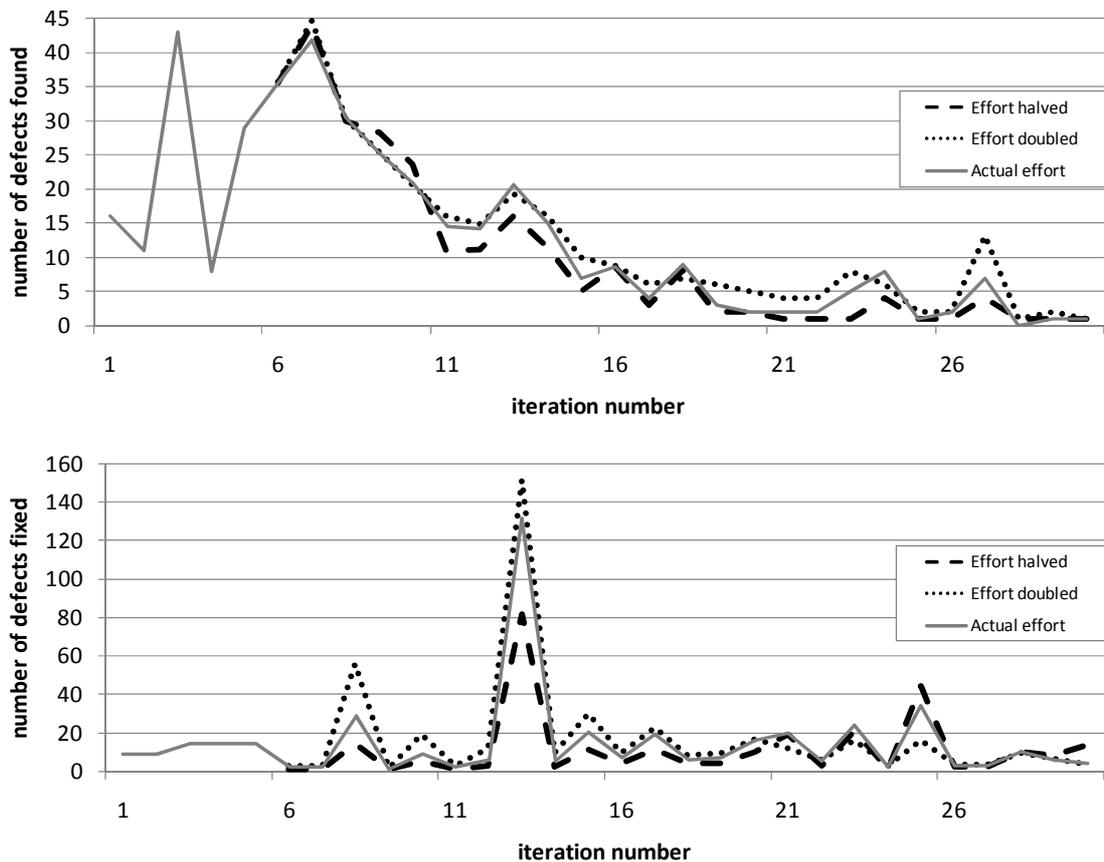


Figure 10-13 Predictions from LMITFD with testing and fixing *effort* halved and doubled after 5 iterations used to learn the model

The data in a dataset assume that *process and people quality* increases gradually to reflect the ‘learning curve’ for the testers. In this case we assume that starting from iteration 6 process factors are fixed: *process and people quality* is ‘high’ and *other*

factors are at ‘medium’ level throughout the whole remaining part of testing and fixing process.

Figure 10-14 illustrates model predictions for this case. The model predicts that we will find residual defects very rapidly – within just a couple of testing iterations. These results also show that the model believes that these process factors have a much higher impact on *defects found* and *defects fixed* than effort on these activities. Also, in the final iterations predicted *defects fixed* with fixed process factors are lower than with actual process factors. The explanation for is that in these late iterations actual process factors are more favourable than in the scenario with fixed process factors. Thus, with more favourable process factors more defects can later be fixed. Such a trend cannot be observed for *defects found*. It is caused by the fact that in late testing iterations there are no defects which can be found because they were already found much earlier.

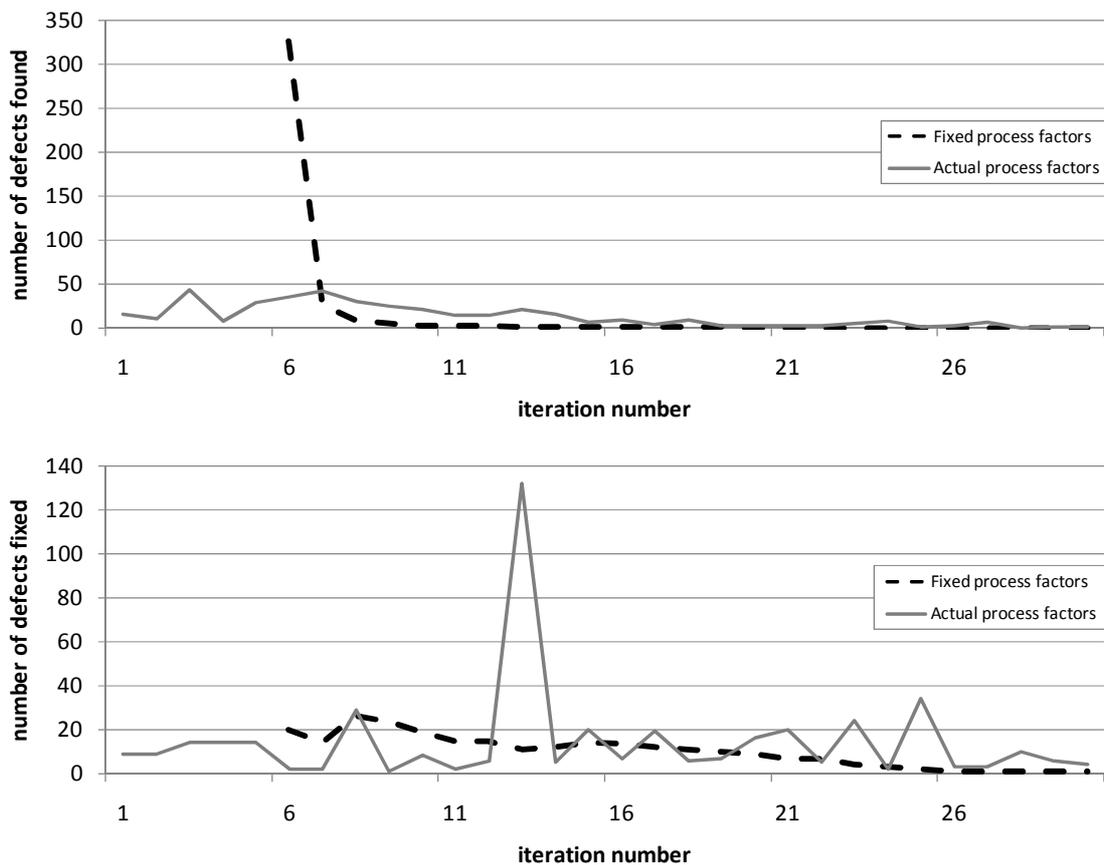


Figure 10-14 Predictions from LMITFD with very high *process and people quality* and medium *other factors* starting from iteration 6

In the final example (Figure 10-15) we analyze model predictions in two scenarios when *fixing process and people quality* is ‘very low’ or ‘very high’. Values of other process factors are the actual values in iterations used for prediction. As expected, with

lower *fixing process and people quality*, fewer defects are likely to be fixed in future iterations. These differences are very high, confirming that the model updated its *multipliers* and *residual defects* to reflect the fact qualitative process factors are more influential on *defects found* and *defects fixed* than *effort*.

We can see that, although we have only modified our observations of *fixing process and people quality*, the predicted values of *defects found* are also different in these two scenarios. The reason is: Lower *fixing process and people quality* with *fixing effort* unchanged leads to increased number of *defects inserted* as a result of imperfect fixing. Higher number of *defects inserted* causes more *residual defects* in subsequent iterations. Higher number of *residual defects* in turn causes there to be more *defects found* – the more *residual defects* the easier it is to find them.

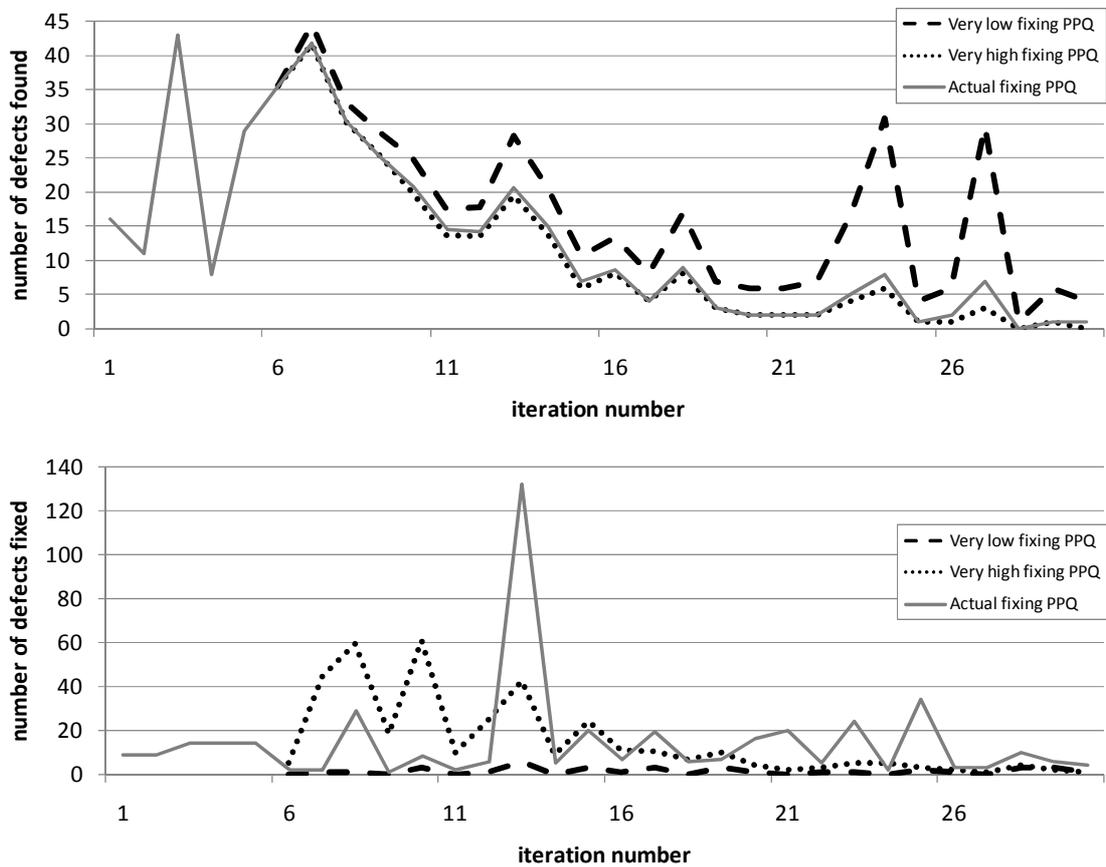


Figure 10-15 Predictions from LMITFD with very high and very low *fixing process and people quality* after 5 iterations used to learn the model

Another issue which may be surprising in the beginning is the fact that predicted number of *defects fixed* in the late iterations is lower both with ‘very low’ and ‘very high’ *fixing process and people quality* compared with the scenario with the original *fixing process and people quality*. But there is also an explanation for such predictions.

With ‘very low’ *fixing process and people quality* it is simply not possible to fix more defects without assigning significantly more *fixing effort*. On the other hand, with ‘very high’ *fixing process and people quality* defects which were found earlier were also fixed earlier. So in these late iterations there are fewer defects still to be fixed.

The examples discussed above are just some of many that demonstrate the types of analyses which can be performed using this model.

10.8 Summary

In this chapter we introduced models for defect prediction that learn their parameters using the current project’s process data. These models are especially powerful because they are not based on metrics from past projects (which may be completely different to the current project). The ability to perform various ‘what-if’ analyses seems to be a clear advantage of such causal learning models over classic reliability growth models which do not incorporate process factors and thus have significantly lower potential in terms of providing useful information for project managers.

11 Summary and future work

11.1 *Novel contributions*

Integrated model for project trade-off analysis

The main contribution of this thesis is the development of the Productivity Model. It can perform resource and quality predictions for software projects, and powerful what-if and trade-off analysis to support project managers. Although such analyses were partially available in a previous BN model (the MODIST Project-level Model), the Productivity Model overcomes some significant weaknesses of the MODIST model:

- It fully captures trade-off relationships between key project variables adjusted by several factors describing the project and development process.
- It is much easier to use basic metrics, such as defect and productivity rates, extracted from past project databases to adjust the model for a specific software company's needs.
- It incorporates defect information that was outside the scope of the MODIST Project-level Model.
- It is independent of the units of measurement for effort and functionality.
- It provides more accurate predictions by virtue of the use of dynamic discretisation for numeric nodes.
- It can be calibrated to an individual company needs using the provided questionnaire.

The Productivity Model, together with various analyses which were performed in order to validate the model's predictions (Chapters 6-7), confirms Hypothesis 1 stated in Chapter 1: It is possible to build a BN for software project risk assessment which overcomes key limitations of existing BNs without compromising their basic philosophy. In particular, such a BN enables trade-off analysis between key project factors (effort, functionality and product quality) and can be tailored to individual company needs.

Estimating productivity and defect rates based on environmental factors

One of the aims for this thesis was to analyse how uncontrollable environmental factors impact the productivity and defect rates achieved in software projects. These environmental factors describe a project and a development process. My analysis:

- confirmed that these environmental factors influence the productivity and defect rates;
- identified the nature of the above relationships;
- led to developing a BN (PDR model) for estimating productivity and defect rates;
- confirmed that the PDR model properly captures known empirical data adjusted by expert-knowledge;
- confirmed that estimates provided by the PDR model can be passed as useful inputs to the Productivity Model to impact the estimates provided by the Productivity Model.

Hence, this part of the research (Chapter 8) confirms Hypothesis 2 stated in Chapter 1: It is possible to build a BN for estimating team productivity rate and software defect rate which incorporates environmental factors describing the nature of the software development, and includes factors describing the development process identified through the statistical analysis of empirical data and experts' knowledge.

Predicting types of defects

One of the aims of this research was to analyse factors influencing proportions of defects of various types depending on their severity. My analysis:

- identified factors influencing proportions of types of defects;
- identified the nature of the above relationships;
- led to developing a BN (DTM model) for predicting proportions of types of defects;
- confirmed that the DTM properly captures known empirical data adjusted by expert-knowledge;
- estimates provided by DTM can be linked with estimates provided by the Productivity Model to predict the number of defects categorised by their type.

Hence, Chapter 9 confirms Hypothesis 3 stated in Chapter 1: It is possible to build a BN for proportions of different types of defects categorized by their severity which incorporates environmental factors describing the nature of the software development, and includes factors describing the development process identified through the statistical analysis of empirical data and experts' knowledge.

Learning model for predicting defects found and fixed in iterative testing and fixing process

The fourth main aim of this work was to develop a model for predicting the defects likely to be found and fixed during successive testing and fixing iterations. This model should learn its parameters using the data from past testing and fixing iterations.

When analysing various developed learning models which differ in factors used in each model the following conclusions can be drawn:

- Predicting the number of defects found in future iterations based only on the number of previously observed defects found was not very accurate because of the fluctuations in process effectiveness among testing iterations.
- Adding various process factors, such as process and people quality and effort, to the model significantly improves the accuracy of model predictions.
- With process factors incorporated in the model it is possible to perform various what-if analyses in which scenarios with different levels of process factors in future iterations can be compared.

Developing this learning model also involved developing a software tool, Defect Removal Model Manager, which enables us to create dynamic models, simplifying entering observations, running the model and exporting the results.

Hence, Chapter 10 confirms Hypothesis 4 stated in Chapter 1: It is possible to build a BN for predicting the number of defects likely to be found and fixed in future testing and fixing iterations which learns the values of its parameters using observations from past testing and fixing iterations.

Other contributions

This thesis also provides the following more minor contributions:

- discussion on using constants in existing and new models;
- discussion on using causal and indicator approach in modelling aggregated variables;
- discussion on using dynamic discretisation for numeric nodes for existing and new models;
- discussion on incorporating new empirical data to the existing and new models,
- review of recent empirical data used to inform the existing and new models,

11.2 Future work

Extensions to Productivity Model

As discussed in Section 6.8 the Productivity Model could be enhanced in the following ways:

- ensuring compatibility with the causal risk approach,
- incorporating reused code to model estimates,
- extending defect prediction by providing estimates of defects according to various development activities like in the Phase-based Defect Prediction Model and Revised Defect Prediction Model,
- extending software quality prediction by adding user satisfaction.

Extending software quality prediction

The models could be extended in the following ways:

- *Capture extended notions of software quality.* The models characterise software quality by four variables: number of defects (total or categorized in some ways), defect rate, quality delivered (ranked), and user satisfaction (ranked). However, there are other quality attributes incorporated in various types of static software quality models [24, 25, 59, 78, 86, 91, 117, 159, 202, 247]. A possible extension of our BNs would be to incorporate such attributes, which include: correctness, efficiency, intuitiveness, maintainability, testability, usability etc.
- *Move to lower level defect prediction.* The models predict the number of defects but give no indication where to find them. The challenge is to extend these models to predict fault-prone parts of developed software (at differing levels of granularity).
- *Incorporate the effects of post-release defects.* These defects cause failures in operational use and hence are more urgent for end-users.
- *Incorporate software process improvement more extensively.* The models currently incorporate only limited information on software process improvement (SPI) programs.

User access

A BN tool like AgenaRisk allows end-users to access models in a different way to the way BN designers do by hiding the complexity of the underlying model. However, even with AgenaRisk it is not possible to create multiple end-user profiles for a single

BN. This limitation could be overcome if we could develop an interface manipulation language. It could be similar to the Query by Example (QBE) graphical language for access to database.

11.3 Final conclusions

The last 30 years has seen many proposed predictive models in software engineering. Because of key limitations few models have been successfully used in practice. This thesis introduced models that overcome some key limitations of existing models, specifically providing advances that enable:

- powerful trade-off analysis for project managers;
- enhanced defect prediction

The fact that these models are BNs made it possible to incorporate results from a wide range of sources including empirical data and expert-knowledge.

The models developed are significant improvements in previous BN models that have already achieved some significant penetration in companies such as Philips, Motorola and Siemens. Although, there is still room for further improvements, I believe that the contributions provided by the new models will lead to a more complete understanding of software processes and thus ultimately to delivering better software faster and cheaper.

References

1. Aamodt A., Plaza E., Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches, *Artificial Intelligence Communications*, vol. 7 no. 1, pp. 39–52, Mar. 1994.
2. Agena, AgenaRisk User Manual, 2008, www.agenarisk.com.
3. Agena, AgenaRisk. Bayesian Network Software Tool, 2008, www.agenarisk.com.
4. Agena, Software Project Risk Models Manual, Version 01.00, 2004.
5. Agrawal M., Chari K., Software Effort, Quality, and Cycle Time: A Study of CMM Level 5 Projects, *IEEE Transactions on Software Engineering*, vol. 33 no. 3, pp. 145–156, Mar. 2007.
6. Akiyama F., An Example of Software System Debugging, *Proc. Int. Federation for Information Processing Congress*, vol. 71, Ljubljana, pp. 353–379, 1971.
7. Almering V., van Genuchten M., Cloudt G., Sonnemans P.J.M., Using Software Reliability Growth in Practice, *IEEE Software*, vol. 24 no. 6, pp. 82–88, Nov.–Dec. 2007
8. Alpaydin E., *Introduction to Machine Learning*, The MIT Press, 2004.
9. Angelis L., Stamelos I., A Simulation Tool for Efficient Analogy Based Cost Estimation, *Empirical Software Engineering*, vol. 5 no. 1, pp. 35–68, Mar. 2000.
10. Babu A.J.G., Suresh N., Project management with time, cost, and quality considerations, *Journal of Operational Research*, vol. 88 no. 2, pp. 320–327, Jan. 1996.
11. Bai C.G., Hu Q.P., Xie M., Ng S.H., Software failure prediction based on a Markov Bayesian network model, *Journal of Systems and Software*, vol. 74 no. 3, pp. 275–282, Feb. 2005.
12. Bajaj N., Tyagi A., Agarwal R., Software Estimation – A Fuzzy Approach, *ACM SIGSOFT Software Engineering Notes*, vol. 31 no. 3, May 2006.
13. Becker-Kornstaedt U., Neu H., Learning and Understanding a Software Process through Simulation of its Underlying Model, *Lecture Notes in Computer Science*, vol. 2640, Springer, Berlin, 2003, pp. 81–93.
14. Beecham S., Hall T., Rainer A., Software Process Improvement Problems in Twelve Software Companies: An Empirical Analysis, *Empirical Software Engineering*, vol. 8 no. 1, pp. 7–42, Mar. 2003.
15. Bell R.M., Weyuker E.J., Ostrand T.J., Predicting the Location and number of Faults in Large Software Systems, *IEEE Transactions on Software Engineering*, vol. 34 no. 4, pp. 340–355, Apr. 2005.
16. Bergadano F., Gunetti D., *Inductive Logic Programming: From Machine Learning to Software Engineering*, MIT Press, Cambridge, MA, 1995.
17. Bernardo J.M., Smith A.F.M., *Bayesian Theory*, John Wiley, New York, 1994.

18. Bibi S., Stamelos I., Software Process Modeling with Bayesian Belief Networks, *Proc. of 10th International Software Metrics Symposium* (14–16 Sept. 2004), Chicago, 2004.
19. Bishop C., *Neural Networks for Pattern Recognition*, University Press, Oxford, 1995.
20. Bishop C.M., *Pattern Recognition and Machine Learning*, Springer, 2007.
21. Boehm B., Abts C., Brown A.W., Chulani S., Clark B.K., Horowitz E., Madachy R., Reifer D.J., Steece B., *Software Cost Estimation with COCOMO II*, Prentice Hall, Upper Saddle River, NJ, 2000.
22. Boehm B., Clark B., Horowitz E., Westland C., Madachy R., Selby R., Cost models for future software life cycle process: COCOMO 2.0, *Annals of Software Engineering*, vol. 1 no. 1, pp. 57–94, 1995.
23. Boehm B., *Software Engineering Economics*, Prentice Hall, 1981.
24. Boehm B.W., Brown J.R., Lipow M., Quantitative evaluation of software quality, *Proc. 2nd International Conference on Software Engineering* (Oct. 13–15 1976), San Francisco, 1976, pp. 592–605.
25. Boehm B.W., Brown J.R., Kaspar H., Lipow M., McLeod G., Merritt M., *Characteristics of Software Quality*, North Holland, 1978.
26. Boehm B.W., Software Risk Management: Principles and Practices, *IEEE Software*, vol. 8 no. 1, pp. 32–41, Jan. 1991.
27. Boetticher G., Menzies T., Ostrand T., PROMISE Repository of Empirical Software Engineering Data, West Virginia University, Department of Computer Science, 2008, <http://promisedata.org/repository>.
28. Breiman L., Friedman J., Stone C.J., Olshen R.A., *Classification and Regression Trees*, Wadsworth, Belmont, CA, 1984.
29. Briand L.C., Melo W.L., Wüst J., Assessing the Applicability of Fault-Proneness Models across Object-Oriented Software Projects, *IEEE Transactions on Software Engineering*, vol. 28 no. 7, pp. 706–720, July 2002.
30. British Council, The Polish-British Young Scientists Programme, British Council, 2008, <http://www.britishcouncil.org/poland-young-scientists-programme.htm>.
31. Brodman J., Johnson., Return on Investment (ROI) from Software Process Improvement as Measured by US Industry, *Software Process: Improvement and Practice*, vol. 1 no. 1, 1995.
32. Buchman C., Software Process Improvement at AlliedSignal Aerospace, *Proc. 29th Annual Hawaii International Conference on Systems Science* (Jan. 3–6 1996), vol. 1: Software Technology and Architecture, 1996, pp. 673–680.
33. Calantone R.J., Di Benedetto C.A., Performance and time to market: accelerating cycle time with overlapping stages, *IEEE Transactions on Engineering Management*, vol. 27 no 2, May 2000, pp. 232–244.
34. Cangussu J.W., DeCarlo R.A., Mathur A.P., A Formal Model of the Software Test Process, *IEEE Transactions on Software Engineering*, vol. 28 no. 8, Aug. 2002, pp. 782–796.

35. Carr J.J., Requirements engineering and management: the key to designing quality complex systems, *The TQM Magazine*, vol. 12 no. 6, pp. 400–407, 2000.
36. Ceylan E., Kutlubay F.O., Bener A.B., Software Defect Identification Using Machine Learning Techniques, *Proc. 32nd EUROMICRO Conf. on Software Engineering and Advanced Applications*, Croatia, Aug. 2006, pp. 240–247.
37. Challagulla V.U.B., Bastani F.B., I-Ling Yen, Paul R.A., Empirical assessment of machine learning based software defect prediction techniques, *Proc. 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems*, Feb. 2005, pp. 263–270.
38. Chatzoglou P.D., Macaulay L.A., A Rule-Based Approach to Developing Software Development Prediction Models, *Automated Software Engineering*, vol. 5 no. 2, pp. 211–243, Apr. 1998.
39. Chiu N., Huang S., The adjusted analogy-based software effort estimation based on similarity distances, *Journal of Systems and Software*, vol. 80 no. 4, pp. 628–640, Apr. 2007.
40. Christie A.M., Staley M.J., Organizational and Social Simulation of a Software Requirements Development Process, *Software Process Improvement and Practice*, vol. 5 no. 2–3, pp. 103–110, Jun.–Sep. 2000.
41. Chulani S, Santhanam P., Moore D., Leszkowicz B., Davidson G., Deriving a Software Quality View from Customer Satisfaction and Service Data, *Proc. European Software Conference on Metrics and Measurement* (Apr. 2–4 2001), London, 2001.
42. Chulani S., Boehm B., Modelling Software Defect Introduction and Removal: COQUALMO (COConstructive QUALity MOdel), Technical Report USC-CSE-99-510, University of Southern California, Center for Software Engineering, 1999.
43. Chulani S., Boehm B., Steece B., Bayesian Analysis of Empirical Software Engineering Cost Models, *IEEE Transactions on Software Engineering*, vol. 25 no. 4, pp. 573–583, Jul. 1999.
44. Chulani S., Ray B., Santhanam P., Leszkowicz, R., Metrics for Managing Customer View of Quality, *IEEE Metrics Conference*, Sep. 2003, p. 189.
45. Cockram T., Gaining confidence in Software Inspection using a Bayesian Belief Model, *Software Quality Journal*, vol. 9 no. 1, pp. 31–42, Jan. 2001.
46. Cohen W.W., Devanbu P., A Comparative Study of Inductive Logic Programming Methods for Software Fault Prediction, *Proc. 14th Int. Conf. on Machine Learning* (Jul. 08–12 1997), Morgan Kaufmann Publishers, San Francisco, pp. 66-74.
47. Collofello J.S., Yang Z., Tvedt J.D., Merrill D., Rus J., Modelling Software Testing Processes, *Proc. IEEE 15th Int. Phoenix Conference on Computers and Communications*, 1995.
48. Compton T., Withrow C., Prediction and Control of Ada Software Defects, *Journal of Systems and Software*, vol. 12 no. 3, pp. 199–207, Jul. 1990.
49. Cottengim D., Prerequisites for Success: Why Process Improvement Programs Fail, *Crosstalk. Journal Defense Software Engineering*, Apr. 2002.

50. Cristianini N., Shawe-Taylor J., *An Introduction to Support Vector Machines and other kernel-based learning methods*, Cambridge University Press, Cambridge, UK, 2000.
51. Crosby J.L., *Computer Simulation in Genetics*, John Wiley & Sons, London, 1973.
52. Dai Y., Xie M., Long Q., Ng S., Uncertainty Analysis in Software Reliability Modeling by Bayesian Analysis with Maximum-Entropy Principle, *IEEE Transactions on Software Engineering*, vol. 33 no. 11, pp. 781–795, Nov. 2007.
53. Dalal S.R., Lyu M.R., Mallows C.L., Software Reliability, in: Armitage P., Colton T. (eds.) *Encyclopedia on Biostatistics*, vol. 5, Wiley, 1998, pp. 4550–4555.
54. Dasarathy B.V., *Nearest Neighbor (Nn) Norms*, IEEE Computer Society Press, Washington, 1991.
55. de Barcelos Tronto I.F., da Silva J.D., Sant'Anna N., An investigation of artificial neural networks based prediction systems in software project management, *Journal of Systems and Software*, vol. 81 no. 3, pp. 356–367, Mar. 2008.
56. Devnani-Chulani S., Bayesian Analysis of Software Cost and Quality Models, PhD Thesis, Faculty of The Graduate School, University of Southern California, 1999
57. Dion R., Process Improvement and the Corporate Balance Sheet, *IEEE Software*, vol. 10 no. 4, pp. 28–35, Jul. 1993.
58. Dohi T., Nishio Y., Osaki, S., Optimal software release scheduling based on artificial neural networks, *Annals of Software Engineering*, vol. 8 no. 1–4, pp. 167–185, 1999.
59. Dromey R.G., A model for software product quality, *IEEE Transactions on Software Engineering*, vol. 21 no. 2, pp. 146–163, Feb. 1995.
60. Dubois D., Prade H., *Fuzzy Sets and Systems*, Academic Press, New York, 1988.
61. Ehrlich W., Lee S., Molisani R., Applying reliability measurement: a case study, *IEEE Software*, vol. 7 no. 2, pp. 56–64, Mar.–Apr. 1990.
62. El Emam K., Benlarbi S., Goel N., Rai S.N., Comparing case-based reasoning classifiers for predicting high risk software components, *Journal of Systems and Software*, vol. 55 no. 3, pp. 301–320, Jan. 2001.
63. El Emam K., Briand L., Costs and Benefits of Software Process Improvement, in: Messnarz R., Tully T. (eds.), *Better Software Practice for Business Benefits: Principles and Experience*, IEEE CS Press, Los Alamitos, 1999
64. Emer M.C.F.P., Vergilio S.R., Selection and Evaluation of Test Data Based on Genetic Programming, *Software Quality Journal*, vol. 11 no. 2, pp. 167–186, Jun. 2003.
65. Evett M.P., Khoshgoftaar T.M., Chien P.D., Allen E.B., GP-based Software Quality Prediction, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, Morgan Kaufmann Publishers, Madison, WI, 1998, pp. 60–65.
66. Fenton N., Hearty P., Neil M., Radliński Ł., Software Project and Quality Modelling Using Bayesian Networks, manuscript submitted to: Meziane F., Vadera S. (eds.), *Artificial Intelligence Applications for Improved Software Engineering Development: New Prospects*, 2008.

67. Fenton N., Marsh W., Neil M., Cates P., Forey S., Tailor M., Making Resource Decisions for Software Projects, *Proc. 26th Int. Conf. on Software Engineering* (May 23–28 2004), IEEE Computer Society, Washington, DC, 2004, pp. 397–406.
68. Fenton N., Neil M., Marsh W., Hearty P., Radliński Ł., Krause P., Project Data Incorporating Qualitative Factors for Improved Software Defect Prediction, *Proc. 3rd Int. Workshop on Predictor Models in Software Engineering. Int. Conf. on Software Engineering* (May 20–26, 2007), IEEE Computer Society, Washington, DC, 2.
69. Fenton N., Neil M., Marsh W., Hearty P., Radliński Ł., Krause P., On the effectiveness of early life cycle defect prediction with Bayesian Nets, *Empirical Software Engineering*, vol. 13 no. 5, pp. 499–537, Oct. 2008.
70. Fenton N., Neil M., Measuring your risks, Agena White Paper, 2005, www.agenarisk.com.
71. Fenton N., Neil M., Visualising your Risks, Agena White Paper, 2005, www.agenarisk.com.
72. Fenton N., Radliński Ł., Neil M., Improved Bayesian Networks for Software Project Risk Assessment Using Dynamic Discretisation, in: Sacha K. (ed.) *Software Engineering Techniques: Design for Quality*, IFIP International Federation for Information Processing, vol. 227, Springer, Boston 2006, pp. 139–148.
73. Fenton N.E., Neil M., A Critique of Software Defect Prediction Models, *IEEE Transactions on Software Engineering*, vol. 25 no. 5, pp. 675–689, Sep.–Oct. 1999.
74. Fenton N.E., Neil M., Caballero J.G., Using Ranked Nodes to Model Qualitative Judgements in Bayesian Networks, *IEEE Transactions on Knowledge and Data Engineering*, vol. 19 no. 10, pp. 1420–1432, Oct. 2007.
75. Fenton N.E., Neil M., Marsh W., Krause P., Mishra R., Predicting Software Defects in Varying Development Lifecycles using Bayesian Nets, *Information and Software Technology*, vol. 43 no. 1, pp. 32–43, Jan. 2007.
76. Fenton N.E., Neil M., Software Metrics: Roadmap, in: Finkelstein A. (ed.), *The Future of Software Engineering. Proc. 22nd Int. Conf. on Software Engineering*, ACM Press, 2000, pp. 357–370.
77. Fenton N.E., Ohlsson N., Quantitative Analysis of Faults and Failures in a Complex Software System, *IEEE Transactions on Software Engineering*, vol. 26 no. 8, pp. 797–814, Aug. 2000.
78. Fenton N.E., Pfleeger S.L., *Software Metrics. A Rigorous and Practical Approach*, PWS Publishing Company, Boston, 1997.
79. Ferdinand A.E., A Theory of System Complexity, *Int. Journal of General Systems*, vol. 1, pp. 19–33, 1974.
80. Forrester J.W., *Industrial Dynamics*, Productivity Press, Cambridge, MA, 1961.
81. Francis P., Leon D., Minch M., Podgurski A, Tree-Based Methods for Classifying Software Failures, *Proc. 15th Int. Symp. on Software Reliability Engineering*, Nov. 2004, pp. 451–462.

82. Fraser A., Burnell D., *Computer Models in Genetics*, McGraw-Hill, New York, 1970.
83. Gaffney J.R., Estimating the Number of Faults in Code, *IEEE Transactions on Software Engineering*, vol. 10 no. 4, 1984, pp. 141–152.
84. Galorath D.D., Fischman L., McRitchie K., Driving Quality Through Parametrics, *Crosstalk. Journal of Defense Software Engineering*, pp. 3–5, Nov. 1998.
85. Gelman A., Carlin J.B., Stern H.S., Rubin D.B., *Bayesian Data Analysis*, Chapman and Hall, London, 2003.
86. Georgiadou E., GEQUAMO — A Generic, Multilayered, Customisable, Software Quality Model., *Software Quality Control*, vol. 11 no. 4, pp. 313–323, Nov. 2003.
87. Goel A.L., Okumoto K., An Analysis of Recurrent Software Failures in Real-Time Control System, *Proc. ACM Annual Technology Conference*, Washington, D.C., 1978, pp. 496–500.
88. Goel A.L., Okumoto K., Time-Dependent Error Detection Rate Model for Software Reliability and Other Performance Measures, *IEEE Transactions on Reliability*, vol. 28 no. 3, pp. 206–211, 1979.
89. Goel A.L., Software Reliability Modeling and Estimation Techniques, Report RADC-TR-82-263, Rome Air Development Center, Oct. 1982.
90. Gokhale S.S., Lyu M.R., Regression Tree Modelling for the Prediction of Software Quality, *Proc. ISSAT Int. Conf. on Reliability and Quality in Design*, Anaheim, 1997, pp. 31–36.
91. Grady R.B., Caswell D.L., *Software Metrics: Establishing a Company-Wide Program*, Prentice-Hall, Englewood Cliffs, 1986.
92. Graves S.B., Why costs increase when projects accelerate, *Research Technology Management*, vol. 32 no. 2, pp. 16–18, Mar.–Apr. 1989.
93. Gray A., MacDonell S., Applications of fuzzy logic to software metric models for development effort estimation, *Proc. 1997 Annual Meeting North American Fuzzy Information Processing Society*, Sep. 1997, pp. 394–399.
94. Greer D., Ruhe, G., Software release planning: an evolutionary and iterative approach, *Information and Software Technology*, vol. 46 no. 4, pp. 243–253, 2004.
95. Guo Q., Hierons R.M., Harman M., Derderian K., Heuristics for fault diagnosis when testing from finite state machines, *Software Testing, Verification and Reliability*, vol. 17 no. 1, pp. 41–57, Mar. 2007.
96. Gurney K., *An Introduction to Neural Networks*, Routledge, London, 1997.
97. Haider S.W., Cangussu J.W., Cooper K.M.L., Dantu R., Estimation of Defects Based on Defect Decay Model: ED3M, *IEEE Transactions on Software Engineering*, vol. 34 no. 3, pp. 336–356, May–Jun. 2008.
98. Hall T., Beecham S., Verner J., Wilson D., The impact of staff turnover on software projects: the importance of understanding what makes software practitioners tick, *Proc. 2008 ACM SIGMIS CPR Conference on Computer Personnel Doctoral Consortium and Research* (Charlottesville, VA, Apr. 3–5 2008), ACM, New York, 2008, pp. 30–39.

99. Hall T., Rainer A., Baddoo N., Implementing software process improvement: an empirical study, *Software Process: Improvement and Practice*, vol. 7 no. 1, pp. 3–15, Mar. 2002.
100. Hall T., Wilson D., Rainer A., Jagielska D., Communication: the neglected technical skill?, *Proc. 2007 ACM SIGMIS CPR Conference on Computer Personnel Research: the Global information Technology Workforce* (St. Louis, Missouri, USA, Apr. 19–21, 2007), ACM, New York, 2007, pp. 196–202.
101. Halstead M.H., *Elements of Software Science*, Elsevier North-Holland, New York, 1977.
102. Harman M., Danicic S., Using program slicing to simplify testing, *Journal of Software Testing, Verification and Reliability*, vol. 5 no. 3, pp. 143–162, Sep. 1995.
103. Harman M., Gallager K.B., Program slicing, *Information and Software Technology*, vol. 40 no. 11–12, pp. 577–581, Dec. 1998.
104. Harman M., Hu L., Hierons R., Wegener J., Sthamer H., Baresel A., Roper M., Testability Transformation, *IEEE Transactions on Software Engineering*, vol. 30 no. 1, pp. 3–16, Jan. 2004.
105. Harter D.E., Krishnan M.S., Slaughter S.A., Effects of process maturity on quality, cycle time, and effort in software product development, *Management Science*, vol. 46 no. 4, pp. 451–466, Apr. 2000.
106. Hayes W., Zubrow D., Moving On Up: Data and Experience Doing CMM-Based Software Process Improvement, presentation at: *7th Software Engineering Process Group Conference*, Boston, MA, May 23, 1995.
107. Haykin S., *Neural Networks: A Comprehensive Foundation*, Prentice Hall, Upper Saddle River, NJ, 1999.
108. Henry S., Kafura D., The Evaluation of Software System's Structure Using Quantitative Software Metrics, *Software: Practice and Experience*, vol. 14 no. 6, pp. 561–573, Jun. 1984.
109. Herbrich R., *Learning Kernel Classifiers*, MIT Press, Cambridge, MA, 2002.
110. Herbsleb J., Zubrow D., Siegel J., Rozum J., Carleton A., Software Process Improvement: State of the Payoff, *American Programmer*, vol. 7 no. 9, pp. 2–12, Sep. 1994.
111. Hewett R., Kijisanayothin P., On modeling software defect repair time, *Empirical Software Engineering*, DOI: 10.1007/s10664-008-9064-x, 2008.
112. Hochman R., Allen E., Hudepohl J., Khoshgoftaar T., Evolutionary neural networks: A robust approach to software reliability problems, *Proc. 8th Int. Symp. on Software Reliability Engineering* (Albuquerque, NM, Nov. 2–5 1997), IEEE Computer Society, Washington, DC, 1997, p. 13.
113. Holland J.H., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.
114. Huang S.-J., Chiu N.-H., Optimization of analogy weights by genetic algorithm for software effort estimation, *Information and Software Technology*, vol. 48 no. 11, pp. 1034–1045, Nov. 2006.

115. Humphrey W., *A Discipline for Software Engineering*, Addison-Wesley, Reading, 2002.
116. Humphrey W., Snyder T., Willis R., Software Process Improvement at Hughes Aircraft, *IEEE Software*, vol. 8 no. 4, pp. 11–23, Jul. 1991.
117. Hyatt L., Rosenberg L., A Software Quality Model and Metrics for Identifying Project Risks and Assessing Software Quality, *Proc. 8th Annual Software Technology Conference*, Salt Lake City, Apr. 1996.
118. ISBSG, Estimating, Benchmarking & Research Suite Release 9, International Software Benchmarking Standards Group, 2005, www.isbsg.org.
119. ISBSG, Glossary of Terms, V5.9.1, International Software Benchmarking Standards Group, Feb. 28 2006, www.isbsg.org.
120. ISBSG, ISBSG Comparative Estimating Tool V4.0 – User Guide, International Software Benchmarking Standards Group, 2005, www.isbsg.org.
121. ISBSG, Repository Data Release 10, International Software Benchmarking Standards Group, 2007, www.isbsg.org.
122. ISBSG, Repository Data Release 9 – Field Descriptions, ISBSG, 2005, www.isbsg.org.
123. Jelinski Z., Moranda P., Software Reliability Research, in: Freiburger W. (ed.), *Statistical Computer Performance Evaluation*, Academic Press, New York, 1972, pp. 465–484.
124. Jensen F.V., *An Introduction to Bayesian Networks*, UCL Press, London, 1996.
125. Jones C., *Assessment and Control of Software Risks*, Prentice-Hall PTR, Englewood Cliffs, NJ, 1994.
126. Jones C., *Software Assessments, Benchmarks, and Best Practices*, Addison-Wesley, Boston, 2000.
127. Jones C., Software Quality in 2002: A Survey of the State of the Art, Software Productivity Research, Inc., 2002.
128. Jones C., The Impact of Poor Quality and Canceled Projects on the Software Labor Shortage, Technical Report, Software Productivity Research, Inc., 1998.
129. Jørgensen M., Indahl U., Sjøberg D., Software effort estimation by analogy and "regression toward the mean", *Journal of Systems and Software*, vol. 68 no. 3, pp. 253–262, Dec. 2003.
130. Kan S. H., *Metrics and Models in Software Quality Engineering*, Addison-Wesley, Boston, 2003.
131. Karlsson C., Ahlstrom P., Technological level and product development cycle time, *Journal of Product Innovation Management*, vol. 16 no. 4, pp. 352–362, Jul. 1999.
132. Kemerer C.F., *Software Project Management Readings and Cases*, McGraw-Hill, Boston, 1997.
133. Khoshgoftaar T., Allen E., Xu Z., Predicting testability of program modules using a neural network, *Proc. IEEE Symposium on Application-Specific Systems and Software Engineering Technology*, Mar. 2002, pp. 57–62.

134. Khoshgoftaar T., Pandya A., Lanning D., Application of neural networks for predicting program faults, *Annals of Software Engineering*, vol. 1 no. 1, pp. 141–154, Dec. 1995.
135. Khoshgoftaar T.M., Allen E.B., Deng J., Using regression trees to classify fault-prone software modules, *IEEE Transactions on Reliability*, vol. 51 no. 4, pp. 455–462, Dec. 2002.
136. Kirsopp C., Shepperd M., Hart J., Search heuristics, case-based reasoning and software project effort prediction, *Proc. Genetic and Evolutionary Computation Conference*, San Francisco, CA, 2002, pp. 1367–1374.
137. Kirsopp C., Shepperd M., Making inferences with small numbers of training sets, *IEE Proceedings – Software*, vol. 149 no. 5, pp. 123–130, Oct. 2002.
138. Knox S.T., Modeling the Cost of Software Quality, *Digital Technology Journal*, vol. 5 no. 4, pp. 9–16, 1993.
139. Kolodner J., *Case-Based Reasoning*, Morgan Kaufmann, San Mateo, 1993.
140. Krasner H., Using the Cost of Quality Approach for Software, *Crosstalk. Journal of Defense Software Engineering*, pp. 6–11, Nov. 1998.
141. Kumar S., Krishna B.A., Satsangi P.S., Fuzzy systems and neural networks in software engineering project management, *Journal of Applied Intelligence*, vol. 4, pp. 31–52, 1994.
142. Laplante P.A., Neill C.J., Modeling uncertainty in software engineering using rough sets, *Innovations in Systems and Software Engineering*, vol. 1 no. 1, pp. 71–78, Apr. 2005.
143. Laslo Z., Activity time-cost tradeoffs under time and cost chance constraints, *Computers and Industrial Engineering*, vol. 44 no. 3, pp. 365–384, Mar. 2003.
144. Lavrac N., Dzeroski S., *Inductive Logic Programming: Techniques and Applications*, Ellis Horwood, New York, 1994.
145. Lee P.M., *Bayesian statistics. An Introduction*, Arnold, London, 2004.
146. Leishman T.R., Risk Factor: Confronting the Risks That Impact Software Project Success, *Crosstalk. Journal of Defense Software Engineering*, May 2004.
147. Levendel Y., Reliability Analysis of Large Software Systems: Defect Data Modelling, *IEEE Transactions on Software Engineering*, vol. 16 no. 2, Feb. 1990, pp. 141–152.
148. Leyden J., Open and closed source software defects reloaded, *The Register*, http://www.theregister.co.uk/2003/07/02/open_and_closed_source_software/, Jul. 7 2003.
149. Li J., Ruhe G., Analysis of attribute weighting heuristics for analogy-based software effort estimation method AQUA+, *Empirical Software Engineering*, vol. 13 no. 1, pp. 63–96, Feb. 2008.
150. Lipow M., Number of Faults per Line of Code, *IEEE Transactions on Software Engineering*, vol. 8 no. 4, pp. 437–439, Jul. 1982.
151. Littlewood B., Stochastic Reliability Growth: A Model for Fault Removal in Computer Programs and Hardware Designs, *IEEE Transactions on Reliability*, vol. R-30, pp. 313–320, 1981.

152. Lyu M.R., *Handbook of Software Reliability Engineering*, McGraw-Hill, 1996.
153. MacCormack A., Kamerer C.F., Cusumano M., Crandall B., Trade-offs between Productivity and Quality in Selecting Software Development Practices, *IEEE Software*, vol. 20 no. 5, pp. 78–85, Sep.–Oct. 2003.
154. Mair C., Kadoda G., Lefley M., Phalp K., Schofield C., Shepperd M., Webster S., An investigation of machine learning based prediction systems, *Journal of Systems and Software*, vol. 53 no. 1, pp. 23–29, Jul. 2000.
155. Maxwell K.D., *Applied Statistics for Software Managers*, Prentice Hall PTR, Upper Saddle River, NJ, 2002.
156. Maybeck P.S., *Stochastic models, estimation, and control*, vol. I, Academic Press, New York, 1979.
157. Maybeck P.S., *Stochastic models, estimation, and control*, vol. II, Academic Press, New York, 1982.
158. Mays R.G., Jones C.L., Holloway G.J., Studinski D.P., Experiences with defect prevention, *IBM Systems Journal*, vol. 29 no. 1, pp. 4–32, 1990.
159. McCall J.A., Richards P.K., Walters G.F., *Factors in Software Quality*, National Technical Information Service, vols. I–III, Springfield, VA, 1977.
160. Mendel J.M., *Uncertain Rule-Based Fuzzy Logic Systems: Introduction and New Directions*, Prentice Hall PTR, Upper Saddle River, NJ, 2000.
161. Mendes E., Mosley N., Counsell S., Exploring Case-based Reasoning for Web Hypermedia Project Cost Estimation, *Int. Journal of Web Engineering and Technology*, vol. 2 no. 1, pp. 117–143, 2005.
162. Mendes E., Mosley N., Watson I., A Comparison of Case-Based Reasoning Approaches to Web Hypermedia Project Cost Estimation, *Proc. 11th Int. Conf. on World Wide Web*, Honolulu, May 2002, pp. 272–280.
163. Menzies T., Greenwald J., Frank A., Data Mining Static Code Attributes to Learn Defect Predictors, *IEEE Transactions on Software Engineering*, vol. 33 no. 11, pp. 1–12, Jan. 2007.
164. Merriam-Webster's Online Dictionary, 2008, <http://www.merriam-webster.com/>.
165. Metrics Data Program, NASA IV&V facility, 2008, <http://mdp.ivv.nasa.gov/>.
166. Michie D., Spiegelhalter D.J., Taylor C.C. (eds.), *Machine Learning, Neural and Statistical Classification*, Ellis Horwood, Chichester, UK, 1994.
167. Mitchell M., *An Introduction to Genetic Algorithms*, MIT Press, Cambridge, MA, 1996.
168. Mitchell T.M., *Machine Learning*, McGraw-Hill, 1997.
169. Mockus A., Weiss D.M., Zhang P., Understanding and Predicting Effort in Software Projects, *Proc. 25th Int. Conf. on Software Engineering* (Portland, OR, May 3–10 2003), IEEE Computer Society, Washington, DC, 2003, pp. 274–284.
170. MODIST, Models of Uncertainty and Risk for Distributed Software Development. EC Information Society Technologies Project IST-2000-28749, 2003, www.modist.org.

171. Morasca S., Ruhe G., A Hybrid Approach to Analyze Empirical Software Engineering Data and its Application to Predict Module Fault-Proneness in Maintenance, *Journal of Systems and Software*, vol. 53 no. 3, pp. 225–237, Sep. 2000.
172. Moses J., Bayesian probability distributions for assessing measurement of subjective software attributes, *Information and Software Technology*, vol. 42 no. 8, pp. 533–546, May 2000.
173. Moses J., Farrow M., Parrington N., Smith P., A productivity benchmarking case study using Bayesian credible intervals, *Software Quality Control*, vol. 14 no. 1, pp. 37–52, Mar. 2006.
174. Muggleton S., (ed.), *Inductive Logic Programming*, Academic Press, New York, 1991.
175. Musa J.D., Iannino A., Okumoto K., *Software Reliability: Measurement, Prediction, Application*, McGraw-Hill, 1989.
176. Musa J.D., Okumoto K., A Logarithmic Poisson Execution Time Model for Software Reliability Measurement, *Proc. 7th Int. Conf. on Software Engineering*, IEEE Press, Los Alamitos, 1984, pp. 230–238.
177. Neapolitan R.E., *Learning Bayesian Networks*, Pearson Prentice Hall, Upper Saddle River, 2004.
178. Neil M., Tailor M., Marquez D., Bayesian statistical inference using dynamic discretisation, RADAR Technical Report, Queen Mary, University of London, 2005.
179. Neil M., Tailor M., Marquez D., Fenton N., Hearty P., Modelling Dependable Systems using Hybrid Bayesian Networks, *Proc. 1st Int. Conf. on Availability, Reliability and Security* (Apr. 20–22 2006), Vienna, 2006.
180. Neil M., Tailor M., Marquez D., Inference in Hybrid Bayesian Networks using dynamic discretisation, RADAR Technical Report, Queen Mary, University of London, 2005.
181. Ohba M., Software Reliability Analysis Models, *IBM Journal of Research and Development*, vol. 28, pp. 428–443, 1984.
182. Ohlsson N., Alberg H., Predicting fault-prone software modules in telephone switches, *IEEE Transactions on Software Engineering*, vol. 22 no. 12, pp. 886–894, Dec. 1996.
183. Ostrand T.J., Weyuker E.J., Bell R.M., Predicting the location and number of faults in large software systems, *IEEE Transactions on Software Engineering*, vol. 31 no. 4, pp. 340–355, Apr. 2005.
184. Pai G.I., Dugan J.B., Lateef K., Bayesian Networks applied to Software IV&V, *Proc. 29th Annual IEEE/NASA Software Engineering Workshop*, 2005, pp. 293–304.
185. Pai G.J., Dugan J.B., Empirical Analysis of Software Fault Content and Fault Proneness Using Bayesian Methods, *IEEE Transactions on Software Engineering*, vol. 33 no. 10, pp. 675–686, Oct. 2007.
186. Patterson D., *Artificial Neural Networks*, Prentice Hall, Singapore, 1996.

187. Pawlak Z., *Rough Sets: Theoretical Aspects of Reasoning About Data*, Kluwer Academic Publishing, Dordrecht, 1991.
188. Pawlak Z., Wong S.K.M., Ziarko W., Rough sets: Probabilistic versus deterministic approach, *Int. Journal of Man-Machine Studies*, vol. 29, pp. 81–95, 1988.
189. Pearl J., *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann, San Francisco, 1988.
190. Peters J.F., Ramanna S., Towards a Software Change Classification System: A Rough Set Approach, *Software Quality Control*, vol. 11 no. 2, pp. 121–147, Jun. 2003.
191. Pfahl D., Al-Emran A., Ruhe G., A System Dynamics Simulation Model for Analyzing the Stability of Software Release Plans: Research Sections, *Software Process: Improvement and Practice*, vol. 12 no. 5, pp. 475–490, Sep. 2007.
192. Pfahl D., Ruhe G., System Dynamics as an Enabling Technology for Learning in Software Organisations, Fraunhofer Institute for Experimental Software Engineering, IESE Report 025.01/E, 2001.
193. Pollack-Johnson B., Liberatore M.J., Incorporating Quality Considerations Into Project Time/Cost Tradeoff Analysis and Decision Making, *IEEE Transactions on Engineering Management*, vol. 53 no. 4, pp. 534–542, Nov. 2006.
194. Pulat P.S., Horn S.J., Time-Resource Tradeoff Problem, *IEEE Transactions on Engineering Management*, vol. 43 no. 4, pp. 411–417, Nov. 1996.
195. Putnam D., Exploring the Outer Limits: How much software can be developed in a year?, *Quantitative Software Management*, Mar. 2002, http://www.qsm.com/Develop_12%20months.pdf.
196. Putnam D., Team Size Can Be the Key to a Successful Project, *Quantitative Software Management*, McLean, VA, 2005.
197. Putnam L.H., A general empirical solution to the macro software sizing and estimating problem, *IEEE Transactions on Software Engineering*, vol. 4 no. 4, pp. 345–61, Apr. 1978.
198. Quah T.S., Thwin M.M.T., Application of Neural Networks for Software Quality Prediction Using Object-Oriented Metrics, *Proc. 19th IEEE Int. Conf. on Software Maintenance*, 2003, p. 116.
199. Radliński Ł., Fenton N., Neil M., Marquez D., Modelling Prior Productivity and Defect Rates in a Causal Model for Software Project Risk Assessment, *Polish Journal of Environmental Studies*, vol. 16 no. 4A, pp. 256–260, 2007.
200. Radliński Ł., A Review of Publicly Available Databases of Software Projects, accepted to: *Studia Informatica*, vol. 22, (in Polish – original title: Przegląd publicznie dostępnych baz danych przedsięwzięć informatycznych).
201. Radliński Ł., A Survey of Bayesian Networks for Risk Assessment in Software Engineering, accepted to: *Studia Informatica*, vol. 21, (in Polish – original title: Przegląd sieci Bayesa do szacowania ryzyka w inżynierii oprogramowania).
202. Radliński Ł., Comparative Analysis of Software Quality Models, *Studia Informatica*, vol. 19, pp. 131–150, 2006 (in Polish – original title: Analiza porównawcza modeli jakości oprogramowania).

203. Radliński Ł., Fenton N., Marquez D., Estimating Productivity and Defects Rates Based on Environmental Factors, *Information Systems Architecture and Technology: Models of the Organisation's Risk Management*, Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław, Poland, 2008, pp. 103–113.
204. Radliński Ł., Fenton N., Marquez D., Hearty P., Empirical Analysis of Software Defect Types, *Information Systems Architecture and Technology. Information Technology and Web Engineering: Models, Concepts & Challenges*, Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław, Poland 2007, pp. 223–231.
205. Radliński Ł., Fenton N., Neil M., A Learning Bayesian Net for Predicting Number of Software Defects Found in a Sequence of Testing, *Polish Journal of Environmental Studies*, vol. 17 no. 3B, pp. 359–364, 2008.
206. Radliński Ł., Fenton N., Neil M., Marquez D., Improved Decision-Making for Software Managers Using Bayesian Networks, *Proc. 11th IASTED Int. Conf. Software Engineering and Applications*, Cambridge, MA, 2007, pp. 13–19.
207. Radliński Ł., Modelling Complex Nodes in Bayesian Nets for Software Project Risk Assessment, *Polish Journal of Environmental Studies*, vol. 15 no. 4C, pp. 149–152, 2006.
208. Radliński Ł., Selected Problems of User Requirements Specification Quality Assessment, *Advanced Information Technologies for Management. Research Papers*, no. 1044, Publishing House of the University of Economics, Wrocław, Poland, 2003, pp. 69–77, (in Polish – original title: Wybrane problemy zapewnienia jakości specyfikacji wymagań użytkowników).
209. Radliński Ł., Software Process Improvement in Small Software Companies, in: Kisielnicki J. (ed.), *Informatics as Tool in Modern Management*, Wydawnictwo Polsko-Japońskiej Wyższej Szkoły Technik Komputerowych, Warsaw, Poland, 2004, pp. 437–446, (in Polish – original title: Poprawa procesów programowych w małych organizacjach programistycznych; book title: Informatyka narzędziem współczesnego zarządzania).
210. Radliński Ł., Software Quality Measurement – Opportunities and Limitations, *Firma i Rynek*, no. 2–4 (27–29), pp. 136–139, 2003, (in Polish – original title: Pomiar jakości oprogramowania – możliwości i ograniczenia).
211. Radliński Ł., Software reliability estimation and prediction, in: *Advanced Information Technologies for Management. Research Papers*, no. 1044, Publishing House of the University of Economics, Wrocław, Poland, 2004, pp. 82–90.
212. Raffo D.M., Harrison W., Vandeville J., Coordinating Models and Metrics to Manage Software Projects, *Software Process Improvement and Practice*, vol. 5 no. 2–3, pp. 159–168, Jun.–Sep. 2000.
213. Rainer A., Hall T., A quantitative and qualitative analysis of factors affecting software processes, *Journal of Systems and Software*, vol. 6 no. 1, pp. 7–21, Apr. 2003.
214. Rainer A., Hall T., Identifying the causes of poor progress in software projects, *Proc. 10th Int. Symposium on Software Metrics*, Sep. 2004, pp. 184–195.
215. Ramanna S., Rough Neural Network for Software Change Prediction, *Proc. 3rd Int. Conf. on Rough Sets and Current Trends in Computing, Lecture Notes in Computer Science*, vol. 2475, Springer-Verlag, London, Oct. 2002, pp. 602–609.

216. Ramírez Cid J.E., Achcar J.A., Bayesian inference for nonhomogeneous Poisson processes in software reliability models assuming nonmonotonic intensity functions, *Computational Statistics & Data Analysis*, vol. 32 no. 2, pp. 147–159, Dec. 1999.
217. Rico D.F., Using Cost Benefit Analyses to Develop Software Process Improvement (SPI) Strategies, A DACS State-of-the-Art Report, Data & Analysis Center for Software, 2000.
218. Ruhe G., Eberlein A., Pfahl D., Trade-off Analysis for Requirements Selection, *Int. Journal on Software Engineering and Knowledge Engineering*, vol. 13 no. 4, pp. 345–366, Aug. 2003.
219. Ruhe G., Rough Set Based Data Analysis in Goal Oriented Software Measurement, *Proc. 3rd Int. Symposium on Software Metrics*, 1996, p. 10.
220. Russell S., Norvig P., *Artificial Intelligence. A Modern Approach*, Second Edition, Pearson Education, Inc., Upper Saddle River, 2003.
221. Saltelli A., What is Sensitivity Analysis, in: Saltelli A., Chan K., Scott E.M. (eds.), *Sensitivity Analysis*, John Wiley & Sons, 2000, pp. 4–13.
222. Sassenburg J.A., Design of a Methodology to Support Software Release Decisions (Do the numbers Really Matter?), PhD Thesis, University of Groningen, 2006.
223. Sastry M.A., Sterman J.D., Desert Island Dynamics: An Annotated Survey of the Essential System Dynamics Literature, System Dynamics Group, Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA, 1992.
224. Schank R., *Dynamic Memory: A Theory of Learning in Computers and People*, Cambridge University Press, New York, 1982.
225. Schölkopf B., Burges C., Smola A. (eds), *Advances in Kernel Methods – Support Vector Learning*, MIT Press, Cambridge, MA, 1999.
226. Schröter A., Zimmermann T., Zeller A., Predicting component failures at design time, *Proc. 2006 ACM/IEEE Int. Symposium on Empirical Software Engineering* (Rio de Janeiro, Sep. 21–22, 2006), ACM, New York, NY, 2006, pp. 18–27.
227. Seok J.Y., Simmons D.B., Continuous productivity assessment and effort prediction based on bayesian analysis, *Proc. 28th Annual Int. Computer Software and Applications Conference. Workshops and Fast Abstracts*, Hong Kong, Sep. 2004, pp. 40–49.
228. Shakhnarovich G., Darrell T., Indyk P. (eds.), *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice*, MIT Press, Cambridge, MA, 2006.
229. Shepperd M., Kadoda G., Comparing Software Prediction Techniques Using Simulation, *IEEE Transactions on Software Engineering*, vol. 27 no. 11, pp. 1014–1022, Nov. 2001.
230. Shepperd M., Schofield C., Estimating Software Project Effort Using Analogies, *IEEE Transactions on Software Engineering*, vol. 23 no.11, pp. 736–743, Nov. 1997.
231. Sitte R., Comparison of software-reliability-growth predictions: neural networks vs parametric-recalibration, *IEEE Transactions on Reliability*, vol. 48 no. 3, pp. 285–291, Sep. 1999.

232. Smith P.G., Reinertsen D.G., *Developing Products in Half the Time: New Rules, New Tools*, John Wiley & Sons Inc., New York, 1998.
233. Srinivasan K., Fisher D., Machine learning approaches to estimating software development effort, *IEEE Transactions on Software Engineering*, vol. 21 no. 2, pp. 126–137, Feb. 1995.
234. Statsoft, Inc., STATISTICA (data analysis software system), version 8.0, 2007, www.statsoft.com.
235. Stefanowski J., An empirical study of using rule induction and rough sets to software cost estimation, *Fundamenta Informaticae*, vol. 71 no. 1, pp. 63–82, May 2006.
236. Stensrud E., Foss T., Kitchenham B., Myrtveit I., An Empirical Validation of the Relationship Between the Magnitude of Relative Error and Project Size, *Proc. 8th Int. Symposium on Software Metrics*, Washington, DC, 2002, p. 3.
237. Stewart B., Predicting project delivery rates using the Naive–Bayes classifier, *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 14, pp. 161–179, May 2002.
238. Stringfellow C, Amschler Andrews A, An empirical method for selecting software reliability growth models, *Empirical Software Engineering*, vol. 7 no. 4, pp. 319–343, 2002.
239. Swink M., Talluri S., Pandejpong T., Faster, better, cheaper: A study of NPD project efficiency and performance tradeoffs, *Journal of Operations Management*, vol. 24, pp. 542–562, 2006.
240. Texas Department of Information Resources, Generic Software Project Risk Factors, www.dir.state.tx.us/eod/qa/risk/swrisk.htm, 2003.
241. The Mozilla Organisation, Bugzilla Installation List, 2007, <http://www.bugzilla.org/installation-list>.
242. The Mozilla Organisation, Bugzilla, 2008, <http://www.bugzilla.org>.
243. The System Dynamics in Education Project (SDEP), Massachusetts Institute of Technology, Cambridge, MA, 2003, <http://sysdyn.clexchange.org>.
244. Tvedt J.D., Collofello J.S., Evaluating the Effectiveness of Process Improvements on Software Development Cycle Time via System Dynamics Modelling, *Proc. 19th Int. Computer Software and Applications Conference*, 1995, p. 318.
245. Upchurch R., System (Dynamics) Models in Teaching/Learning, 2005, <http://www2.umassd.edu/systemdynamics/papers.html>.
246. van Koten C., Bayesian statistical models for predicting software development effort, Discussion Paper 2005/08, Department of Information Science, University of Otago, Dunedin, New Zealand, 2005.
247. van Veenendaal E., Hendriks R., van Vonderen R., Measuring Software Product Quality, *Software Quality Professional*, vol. 5 no. 1, Dec. 2002.
248. Wagner S., Global Sensitivity Analysis of Predictor Models in Software Engineering, *Proc. 3rd Int. Workshop on Predictor Models in Software Engineering, Int. Conf. on Software Engineering*. IEEE Comp. Soc., Washington, DC, 2007, p. 3.

249. Walkerden F., Jeffery R., An Empirical Study of Analogy-based Software Effort Estimation, *Empirical Software Engineering*, vol. 4 no. 2, pp. 135–158, Jun. 1999.
250. Webster's Online Dictionary, 2008, <http://www.websters-online-dictionary.org/>.
251. Westfall L., Software Customer Satisfaction, *Proc. Applications in Software Measurement Conference*, 2002.
252. Winkler R.L., *An Introduction to Bayesian Inference and Decision*. Second Edition, Probabilistic Publishing, Gainesville, FL, 2003.
253. Wohlwend H., Rosenbaum S., Software Improvements in an International Company, *Proc. Int. Conf. on Software Engineering*, 1993, pp. 212–220,
254. Wooff D.A., Goldstein M., Coolen F.P.A., Bayesian Graphical Models for Software Testing, *IEEE Transactions on Software Engineering*, vol. 28, no. 5, pp. 510–525, May 2002.
255. Wu L., The Comparison of the Software Cost Estimating Methods, University of Calgary, 1997, <http://www.computing.dcu.ie/~renaat/ca421/LWu1.html>.
256. Yamada S., Ohba M., Osaki S., S-Shaped Reliability Growth Modeling for Software Error Detection, *IEEE Transactions on Reliability*, vol. 32 no. 5, pp. 475–478, 1983.
257. Yang T.Y., Computational approaches to Bayesian inference for software reliability, Ph.D. Thesis, Department of Statistics, University of Connecticut, Storrs, USA, 1994.
258. Zadeh L.A., Fuzzy sets, *Information and Control*, vol. 8 no. 3, 1965, pp. 338–353.
259. Zahran S., *Software Process Improvement*, Addison-Wesley, Harlow, 1998.
260. Zhang D., Tsai J.J.P., Machine Learning and Software Engineering, *Software Quality Journal*, vol. 11, no. 2, Jun. 2003.
261. Zhang Y., Harman M., Mansouri S.A., The Multi-Objective Next Release Problem, *Proc. Genetic and Evolutionary Computation Conference*, London, 2007, pp. 1129–1137.
262. Zhao Jianmin, Chan A.H.C., Roberts C., Madelin K.B., Reliability Evaluation and optimization of imperfect inspections for a component with multi-defects, *Reliability Engineering and System Safety*, vol. 92, pp. 65–73, 2007.

Appendix A Summary of empirical data used indirectly

A.1 Project management

Table A-1 summarizes work pattern for typical software engineer including only effective working time (no vacations, training, sick days etc.) and with other activities than working on actual projects factored out [222].

Table A-1 Software engineering effort by task [128 cited after 222 p. 5]

| Activities | Workdays | Percent |
|-----------------------------|----------|---------|
| Testing and defect repairs | 120 | 61% |
| Time on cancelled projects | 30 | 15% |
| Productive time on projects | 47 | 24% |
| Total | 197 | 100% |

Figure A-1 illustrates relationships between reliability with maintenance and development costs and with delivered defects. For example, delivering software with very low reliability involves spending low development costs, higher maintenance costs but more defects will be delivered. Delivering software with very high reliability requires spending higher development costs, few defects will be delivered and maintenance costs will be lower than development costs.

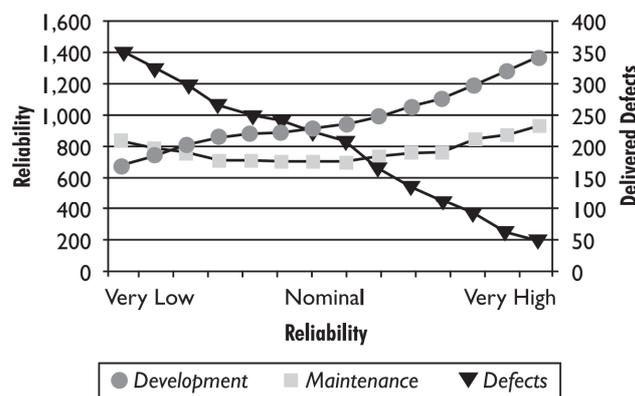


Figure A-1 Development and maintenance effort as quality assurance increases [84]

A.2 Software process improvement

El Emam and Briand made a research in which they analyzed reported costs and benefits of SPI [63]. Their results are summarized in Table A-2.

Table A-2 Organizational experiences illustrating the costs and benefits of SPI [63]

| Ref. | Organization & SPI Program | Costs | Benefits |
|-------|--|---|---|
| [116] | <ul style="list-style-type: none"> SPI effort at the Software Engineering Division of Hughes Aircraft The division had 500 professional employees at the time | <ul style="list-style-type: none"> The assessment itself cost US \$45,000 Cost of a 2 year SPI program was US \$400,000 Implementation of the action plan to move from ML1 to ML2 was 18 calendar months | <ul style="list-style-type: none"> Achieve annual savings of US \$2M Benefits were calculated to be 5 times the improvement expenditures Quality of work life had improved (e.g., fewer overtime hours by the software engineers) |
| [253] | <ul style="list-style-type: none"> SPI effort led by the Schlumberger Laboratory for Computer Science | <ul style="list-style-type: none"> Large engineering centers (120-180 engineers) have 1-5 full-time staff on SPI Smaller centers (50-120 engineers) have up to 3 full-time staff on SPI | <ul style="list-style-type: none"> Improved project communication Customer reports confirm that product quality has improved One group improved time to market by reducing requirements validation cycles to 15 from 34 One group more than doubled its productivity One group increased the percentage of projects completed on schedule from 51% to 94% One group almost halved the defect density in its products |
| [31] | <ul style="list-style-type: none"> Data was collected from 33 companies using questionnaires and/or interviews | <ul style="list-style-type: none"> The authors present examples of data on the costs of activities related to SPI. For example, some organizations increased from 7% to 8% of total effort on data collection, and increase up to 2% of project costs on fixing design defects. | <ul style="list-style-type: none"> Some organizations witnessed increased productivity, reduced defect levels, reduced rework effort, reduction in costs and greater within estimate project completions. In particular, some organizations achieved a ROI of 10:1 Other benefits included less overtime and employee turnover, and increased cooperation between functional groups |
| [32] | <ul style="list-style-type: none"> Corporate-wide SPI effort at AlliedSignal Aerospace starting in 1992 | <ul style="list-style-type: none"> Using data on SEPG investment measured in full-time equivalent headcount for 8 sites, the maximum was 4% | <ul style="list-style-type: none"> One site realized a 7:1 productivity increase in the calendar time to generate documents over 1000 pages with a 50% reduction in the cost per page Independent V&V deficiency reports on the documentation have decreased to approximately zero One site reported that the LOC maintained per person has increased by 50% and testing time has decreased by 60% with no evident increase in delivered defects |
| [57] | <ul style="list-style-type: none"> Organization is the Software Systems Laboratory in Raytheon, employing 400 software engineers SPI initiative started in 1988; results reported after five years Organization has progressed from Level 1 to Level 3 during that period | <ul style="list-style-type: none"> US\$1 million invested every year | <ul style="list-style-type: none"> A 7.7:1 return on every dollar invested Rework costs reduced from 41% of overall project costs to 11% More projects finish ahead of or on schedule and under or on budget Productivity increases by a factor of 2.3 in 4.5 years Software engineers spend fewer late nights and weekends on the job and improved general morale |

Hayes and Zubrow observed the following average times for moving from one CMM level to the next one [106]:

- from Level 1 to Level 2: 30 months,

- from Level 2 to Level 3: 25 months.

Hall et al. observed that there was no clear relationship between companies' maturity and the time they were using SPI programmes. The authors conclude that "that the acceleration of maturity is not necessarily linear" [99].

Boehm et al. observed that moving from one CMM to the next one usually implies increase in productivity of about 4 to 11 percent [21].

Table A-3 illustrates distribution of companies according to their CMM level. We can observe increasing proportion of companies with level 2 and higher while the proportion of companies with level 1 decreases.

Table A-3 Percentage distribution of software organizations by CMM levels

| Reference | Assessment period | CMM Level | | | | |
|---------------------------------|-------------------|-----------|-------|-------|------|------|
| | | 1 | 2 | 3 | 4 | 5 |
| [223] | | 51.1% | 28.9% | 15.7% | 3.7% | 0.6% |
| [130 p.42] | April 1996 | 68.8% | 18% | 11.3% | 1.5% | 0.4% |
| [130 p.42] | March 2000 | 39.3% | 36.3% | 17.7% | 4.8% | 1.8% |
| [99]: formal assessment | | 33% | 20% | 33% | 13% | 0% |
| [99]: informal assessment | | 21% | 46% | 16% | 3% | 0% |

Rico admits that "achieving CMM Level 5 is the rough equivalent of winning the Malcolm Baldrige National Quality Award" [223].

Herbsleb et al. made a research in which they identified benefits of CMM-based SPI [110]. They used data from 13 commercial and DoD organizations. Table A-4 presents the major results of their research.

Table A-4 Main benefits of CMM-based SPI [110]

| | Number of organizations | Median | Smallest | Largest |
|-------------------------------------|-------------------------|---------|----------|---------|
| Cost per software engineer per year | 5 | \$1,375 | \$490 | \$2,004 |
| Productivity gains per year | 4 | 35% | 9% | 67% |
| Gains in early detection of defects | 3 | 22% | 6% | 25% |
| Reduction in calendar time | 2 | 19% | 15% | 23% |
| Reduction in post-release defects | 5 | 39% | 10% | 94% |
| Return on investment | 5 | 500% | 420% | 880% |

Beecham et al. analyzed associations between a company's capability maturity and patterns of reported problems [14]. They analyzed data from 12 software companies. They notice that the sample size is too small for making general conclusions

(for example they had only one CMM Level 4 company). Table A-5 illustrates overall software development problem frequencies by CMM level. The authors observed that mature organizations face relatively more organizational problems to total number of their problems. Lower-level companies have relatively more problems in projects such as documentation, schedule, or using tools and technologies.

Table A-5 Overall software development problem frequencies by row percentages; adopted from [14]

| CMM Level | Number of companies | Problem types | | |
|-----------|---------------------|----------------|---------|-----------|
| | | Organizational | Project | Lifecycle |
| 1 | 6 | 38 | 40 | 22 |
| 2 | 2 | 34 | 40 | 26 |
| 3 | 3 | 41 | 37 | 22 |
| 4 | 1 | 68 | 27 | 5 |

Table A-6 summarizes typical impact of SPI depending on its maturity level on product quality and schedule. We can observe that applying SPI leads to increase in product quality and decrease of schedule.

Table A-6 Impact of SPI on some of software development factors

| Reference | Assessment period | Factor | Maturity Level | | | | |
|-------------|-------------------|-----------------|----------------|--------|-----|--------|------|
| | | | 1 | High 1 | 2 | High 2 | 3+ |
| [259 p. 26] | 1995 | Product Quality | 64% | 82% | 85% | 100% | 100% |
| [259 p. 27] | 1995 | Schedule | 32% | 44% | 58% | 50% | 80% |

Rico discussed numerous other case studies related to applying various SPI strategies in [217]. The main focus of his work was on different benefits caused by SPI, especially impact on schedule, productivity, defects and return on investment.

Table A-7 compares typically achieved effects that are caused by applying various SPI techniques. We can observe that none of them appears to produce most favourable results in every analysed factor. Therefore a careful selection needs to be made before applying them depending on what are the most important goals that should be achieved.

Table A-8 illustrates effects achieved by applying PSP, Inspection and Testing to software projects of the same size. These results show the highest effort, quality and cost benefit in applying PSP.

Table A-9 summarizes costs and benefits of applying various SPI strategies. These results confirm significant positive effects like increase in productivity, quality and return on investment while reducing development cycle time.

Table A-7 Construx comparison of SPI methods [217]

| SPI Method | Cycle-Time Reduction | Progress Visibility | Schedule Risk | First-Time Success | Long-Term Success |
|----------------------------------|----------------------|---------------------|---------------|--------------------|-------------------|
| Change Board | Fair | Fair | Decreased | Very Good | Excellent |
| Daily Build and Smoke Test | Good | Good | Decreased | Very Good | Excellent |
| Designing for Change | Fair | None | Decreased | Good | Excellent |
| Evolutionary Delivery | Good | Excellent | Decreased | Very Good | Excellent |
| Evolutionary Prototyping | Excellent | Excellent | Increased | Very Good | Excellent |
| Goal Setting (shortest schedule) | Very Good | None | Increased | Good | Very Good |
| Goal Setting (least risk) | None | Good | Decreased | Good | Very Good |
| Goal Setting (max visibility) | None | Excellent | Decreased | Good | Very Good |
| Software Inspection Process | Very Good | Fair | Decreased | Good | Excellent |
| Joint Application Development | Good | Fair | Decreased | Good | Excellent |
| Life Cycle Model Selection | Fair | Fair | Decreased | Very Good | Excellent |
| Measurement | Very Good | Good | Decreased | Good | Excellent |
| Miniature Milestones | Fair | Very Good | Decreased | Good | Excellent |
| Outsourcing | Excellent | None | Increased | Good | Very Good |
| Principled Negotiation | None | Very Good | Decreased | Very Good | Excellent |
| Productivity Environments | Good | None | No Effect | Good | Very Good |
| Rapid-Development Languages | Good | None | Increased | Good | Very Good |
| Requirements Scrubbing | Very Good | None | Decreased | Very Good | Excellent |
| Reuse | Excellent | None | Decreased | Poor | Very Good |
| Signing Up | Very Good | None | Increased | Fair | Good |
| Spiral Life Cycle Model | Fair | Very Good | Decreased | Good | Excellent |
| Staged Delivery | None | Good | Decreased | Very Good | Excellent |
| Theory-W Management | None | Very Good | Decreased | Excellent | Excellent |
| Throwaway Prototyping | Fair | None | Decreased | Excellent | Excellent |
| Timebox Development | Excellent | None | Decreased | Good | Excellent |
| Tools Group | Good | None | Decreased | Good | Very Good |
| Top-10 Risks List | None | Very Good | Decreased | Excellent | Excellent |
| User-Interface Prototyping | Good | Fair | Decreased | Excellent | Excellent |
| Voluntary Overtime | Good | None | Increased | Fair | Very Good |

Table A-8 PSP, Software Inspection Process and Testing Comparison [217]

| Cost/Benefits | PSP | Inspection | Test |
|-------------------|----------|------------|----------|
| Program Size | 10 KSLOC | 10 KSLOC | 10 KSLOC |
| Start Defects | 1,000 | 1,000 | 1,000 |
| Review Hours | 97.24 | 708 | n/a |
| Review Defects | 666 | 900 | n/a |
| Defects per Hour | 6.85 | 1.27 | n/a |
| Start Defects | 333 | 100 | 1,000 |
| Test Hours | 60.92 | 1,144 | 11,439 |
| Test Defects | 333 | 90 | 900 |
| Defects per Hour | 5.47 | 12.71 | 12.71 |
| Total Hours | 400 * | 1,852 | 11,439 |
| Total Defects | 1,000 | 990 | 900 |
| Quality Benefit | 100X | 10X | n/a |
| Delivered Defects | 0 | 10 | 100 |
| Cost Benefit | 29X | 6X | n/a |

* PSP hours include development time—others only validation time

Table A-9 Costs and benefits of eight SPI strategies [217]

| | PSP | Cleanroom | Reuse | Prevent | Inspect | Test | CMM | ISO | Average |
|-----------------------|----------|-----------|-----------|---------|---------|----------|----------|---------|----------|
| Breakeven Hours | 80 | 53 | 8,320 | 527 | 7 | 3,517 | 10,021 | 4,973 | 3,437 |
| Training Hours/Person | 80 | 201 | 3,316 | 31 | 19 | 78 | 227 | 64 | 502 |
| Training Cost/Person | \$7,456 | \$8,089 | \$298,440 | \$5,467 | \$1,794 | \$13,863 | \$12,668 | \$9,475 | \$44,656 |
| Effort (Hours) | 400 | 3,245 | 16,212 | 2,100 | 836 | 37,510 | 94,417 | 53,800 | 26,065 |
| Cycle Time Reduction | 164.03x | 3.53x | 3.69x | 1.69x | 5.47x | 6.15x | 2.99x | 1.14x | 23.58x |
| Productivity Increase | 109.49x | 4.27x | 2.70x | 1.88x | 5.47x | 6.15x | 2.92x | 1.13x | 16.75x |
| Quality Increase | 253.62x | 42.22x | 4.33x | 4.77x | 9.00x | 5.75x | 4.55x | 12.44x | 42.09x |
| Return-on-Investment | 1,290: 1 | 27: 1 | 3: 1 | 75: 1 | 133: 1 | 9: 1 | 6: 1 | 4: 1 | 193: 1 |

Rainer and Hall [213] performed quantitative and qualitative analysis of factors affecting software processes. The authors identified the most important factors affecting software processes: executive support, experienced staff, internal process ownership, metrics, procedures, reviews and training. The authors also found that two factors analysed, reward schemes and estimating tools, were not relevant to SPI.

Table A-10 summarizes overall benefits typically achieved and limitations typically faced in implementing SPI programmes. Table A-11 also summarizes such benefits and limitations but in respect with small software companies.

Table A-10 Summary of benefits and limitations of implementing SPI programs [209]

| Benefits | Limitations |
|--|--|
| <ul style="list-style-type: none"> • higher product reliability • higher financial credibility • increased employee satisfaction • installation and maintenance savings • easier schedule implementation • easier process monitoring • providing a base for more accurate schedule preparation • increased product quality • easier realisation of complex projects • increased long-term productivity • increased control over process and product | <ul style="list-style-type: none"> • requires spending additional effort • results difficult to be measured • high cost of SPI programmes • employee reluctance to changing habits • more work related with more accurate documentation • risk about the results |

Table A-11 Summary of benefits and limitations of small software companies in respect with implementing SPI [209]

| Benefits | Limitations |
|---|---|
| <ul style="list-style-type: none"> • usually high staff motivation • short reaction time on customer needs • linear organisational structure (low organisation costs) • generally higher productivity • employee commitment • higher employee engagement in projects • employees included in all software development activities: <ul style="list-style-type: none"> - cost effectiveness - broader experience gained by employees • founders' education in specific areas | <ul style="list-style-type: none"> • lack of historical data for planning and prediction • lack of investments and training • lack of commitment from new employees • lack of knowledge and leadership in respect with software process development • lack of resources required to implement full SPI programmes • informal development process: <ul style="list-style-type: none"> - lack of formalized requirements specification - unstable requirements - lack of common tools and repositories - lack of defined standards - informal testing - lack of quality management - lack of product documentation • required organisational changes with rapid company growth • talented founders often become managers (and commit many errors in this area) • organisation success dependent on highly motivated employees • employees included in all software development activities: <ul style="list-style-type: none"> - lack of cost effectiveness in growing company - broader staff experience limits their specialisation |

A.3 Software sizing

Table A-12 summarizes estimated number of LOC per 1 FP in various programming languages. As we should expect, implementing 1 FP in assembly languages involves significantly more statements than in higher level programming languages such as Smalltalk or SQL.

Table A-12 Estimated numbers of LOC per function point in different programming languages [130 p. 125]

| Language | | | | | | | | | | | | |
|----------------|----------------|-----|---------|-------|--------|------|-------|-----|-------|--------------|-----------|-----|
| Basic Assembly | Macro Assembly | C | FORTRAN | COBOL | Pascal | PL/I | ADA83 | C++ | ADA95 | Visual Basic | Smalltalk | SQL |
| 320 | 213 | 128 | 107 | 107 | 91 | 80 | 71 | 53 | 49 | 32 | 21 | 12 |

Table A-13 summarizes various attributes describing 6 software projects developed in object-oriented languages. We can observe that those projects which were developed in C++ contained more complicated code than those developed in Smalltalk.

However, we cannot generally say that developing projects in C++ produces more complicated code since it rather depends on the nature of the project itself.

Table A-13 Selected object-oriented metrics for 6 projects [130 p. 337]

| Project Metric | Project A (C++) | Project B (C++) | Project C (C++) | Project D (IBM Smalltalk) | Project E (OTI Smalltalk) | Project F (Digitalk Smalltalk) | Rules of Thumb |
|--------------------------------|--------------------|--------------------|--------------------|---------------------------------|---------------------------------|--------------------------------------|------------------------|
| Number of Classes | 5,741 | 2,513 | 3,000 | 100 | 566 | 492 | na |
| Methods per Class | 8 | 3 | 7 | 17 | 36 | 21 | <20 |
| LOC per Method | 21 | 19 | 15 | 5.3 | 5.2 | 5.7 | <8 (S)* <24 (C)* |
| LOC per Class | 207 | 60 | 100 | 97 | 188 | 117 | <160 (S)* <480 (C)* |
| Max Depth of Inheritance | 6 | na | 5 | 6 | 8 | na | <6 |
| Avg DIT | na | na | 3 | 4.8 | 2.8 | na | <4 (C)* |
| (S)* = Smalltalk; (C)* = C++ | | | | | | | |

A.4 Defect prediction

Kan estimates that:

- 95% of defects are found within 4 years from software release – for operating systems,
- most defects are found within 2 years of software release – for application software [130 p. 88].

Rico reported after Mays et al. that IBM Communications Systems “achieved a 50% defect insertion rate reduction for their SPI related efforts, involving 414 software developers” [217 cited after 158]. Figure A-2 illustrates reduced defect insertion rates (front row) at different stages of software development compared to their initial values (second row). In the best case they achieved total return on investment of over 482:1 from their SPI related efforts.

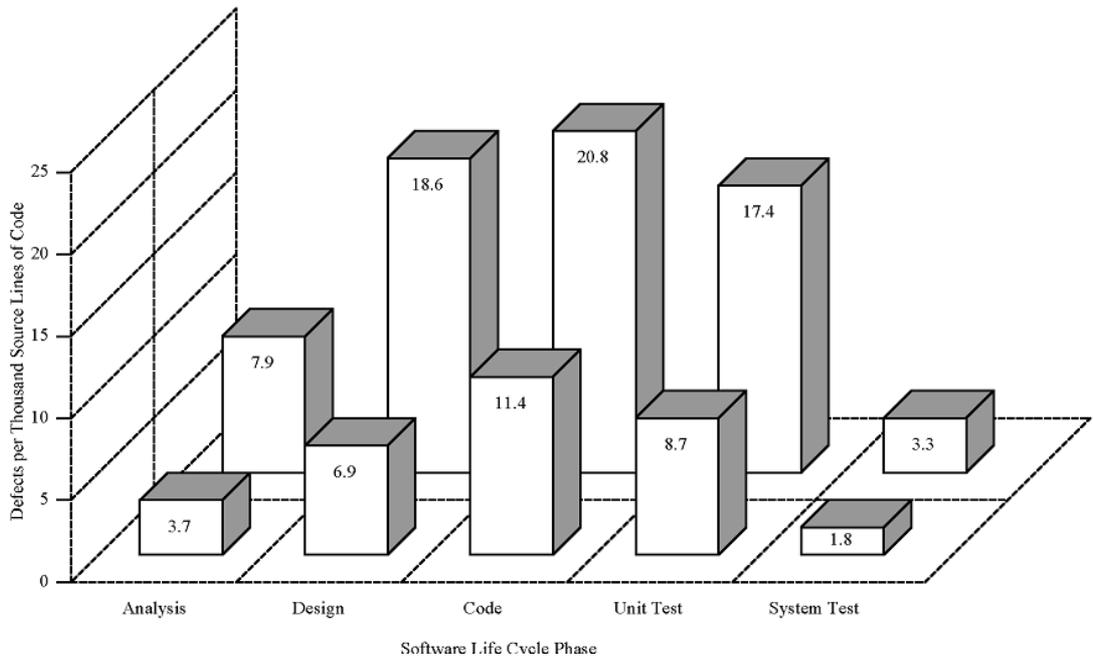


Figure A-2 Defect Prevention Results [217 cited after 158]

Figure A-3 illustrates predictions performed for percentages of various types of costs depending on the CMM level [138 cited after 140]. For example, total cost of software quality (TCoSQ) start at 60% of development costs in organisations with CMM level 1. Moving to CMM level 5 leads to decreasing TCoSQ by about 67%. The only proportion of costs that clearly increases as CMM level increase are prevention costs.

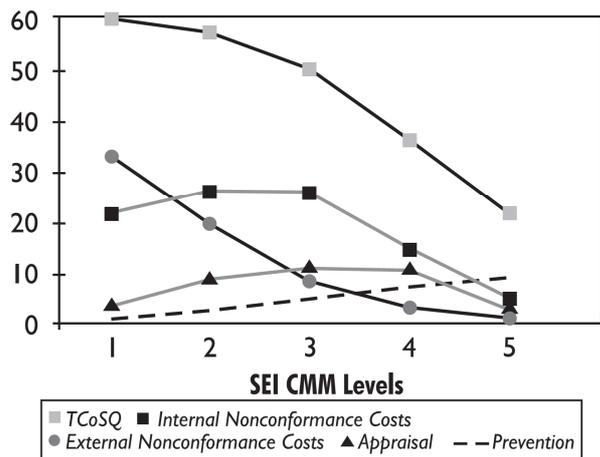


Figure A-3 Cost of software as percentage of development [138 cited after 140]

Appendix B The Productivity Model

B.1 Definition of the Productivity Model

```

<?xml version="1.0" encoding="iso-8859-2" ?>
<object name="Productivity" type="structure">
  <node nodeId="p_revised" name="revised productivity rate"
nodeType="Continuous Interval">
  <states>
    <state id="0" lowerBound="0.0" upperBound="1.0"/>
    <state id="1" lowerBound="1.0" upperBound="Infinity"/>
  </states>
  <npt type="expression">
    <expression>Arithmetic(p_revised_dummy / (10 ^ (4 *
change_total_effort - 2)))</expression>
  </npt>
  <constants>
    <constant name="adjustment_factor" value="1.0"/>
  </constants>
  <parentNodes>
    <parentNode name="p_revised_dummy" />
    <parentNode name="change_total_effort" />
  </parentNodes>
  <childNodes>
    <childNodes name="num_units" />
  </childNodes>
</node>
  <node nodeId="num_units" name="delivered number of units"
nodeType="Continuous Interval">
  <states>
    <state id="0" lowerBound="0.0" upperBound="100000.0"/>
    <state id="1" lowerBound="100000.0" upperBound="Infinity"/>
  </states>
  <npt type="expression">
    <expression>Arithmetic(p_revised*effort_revised)</expression>
  </npt>
  <parentNodes>
    <parentNode name="effort_revised" />
    <parentNode name="p_revised" />
  </parentNodes>
  <childNodes>
    <childNodes name="num_defects" />
  </childNodes>
</node>
  <node nodeId="num_defects" name="number of defects" nodeType="Integer
Interval">
  <states>
    <state id="0" lowerBound="0.0" upperBound="2.0"/>
    <state id="1" lowerBound="2.0" upperBound="Infinity"/>
  </states>
  <npt type="expression">
    <expression>Arithmetic(num_units*d_revised)</expression>
  </npt>
  <parentNodes>
    <parentNode name="num_units" />
    <parentNode name="d_revised" />
  </parentNodes>
  <childNodes>
    <childNodes name="defect_thres" />
  </childNodes>
</node>
  <node nodeId="prior_D" name="prior defect rate" nodeType="Continuous
Interval">
  <states>
    <state id="0" lowerBound="0.0" upperBound="1.0"/>

```



```

        <expression>Arithmetic(prior_D * 2.6 ^ (-4 * impact_on_D +
2))</expression>
    </npt>
    <constants>
        <constant name="adjustment_factor" value="1.0"/>
    </constants>
    <parentNodes>
        <parentNode name="prior_D" />
        <parentNode name="impact_on_D" />
    </parentNodes>
    <childNodes>
        <childNode name="num_defects" />
    </childNodes>
</node>
<node nodeId="test_effort" name="change in effort on testing"
nodeType="Ranked">
    <states>
        <state id="0" name="Extra Lower"/>
        <state id="1" name="Much Lower"/>
        <state id="2" name="Lower"/>
        <state id="3" name="The Same"/>
        <state id="4" name="Higher"/>
        <state id="5" name="Much Higher"/>
        <state id="6" name="Extra Higher"/>
    </states>
    <npt type="expression">
        <expression>TNormal(0.5,0.1,0.0,1.0)</expression>
    </npt>
    <parentNodes>
    </parentNodes>
    <childNodes>
        <childNode name="tot_effort_dummy_2" />
        <childNode name="testing_effort_testing_eff" />
    </childNodes>
</node>
<node nodeId="uncontrollable_on_P" name="uncontrollable on productivity
rate" nodeType="Ranked">
    <states>
        <state id="0" name="Extra Lower"/>
        <state id="1" name="Much Lower"/>
        <state id="2" name="Lower"/>
        <state id="3" name="The Same"/>
        <state id="4" name="Higher"/>
        <state id="5" name="Much Higher"/>
        <state id="6" name="Extra Higher"/>
    </states>
    <npt type="expression">
        <expression>Arithmetic(wmean(
21, 1 - uncontrollable_on_P_dummy_2,
16.3, uncontrollable_on_P_dummy_1,
7, dead_pres))</expression>
    </npt>
    <parentNodes>
        <parentNode name="dead_pres" />
        <parentNode name="uncontrollable_on_P_dummy_2" />
        <parentNode name="uncontrollable_on_P_dummy_1" />
    </parentNodes>
    <childNodes>
        <childNode name="impact_on_P" />
    </childNodes>
</node>
<node nodeId="uncontrollable_on_D" name="uncontrollable on defect rate"
nodeType="Ranked">
    <states>
        <state id="0" name="Extra Lower"/>
        <state id="1" name="Much Lower"/>
        <state id="2" name="Lower"/>
        <state id="3" name="The Same"/>

```

```

        <state id="4" name="Higher" />
        <state id="5" name="Much Higher" />
        <state id="6" name="Extra Higher" />
    </states>
    <npt type="expression">
        <expression>Arithmetic(wmean(
19, uncontrollable_on_D_dummy_1,
19.5, 1 - uncontrollable_on_D_dummy_2,
8.5, 1 - dead_pres))</expression>
    </npt>
    <parentNodes>
        <parentNode name="uncontrollable_on_D_dummy_1" />
        <parentNode name="uncontrollable_on_D_dummy_2" />
        <parentNode name="dead_pres" />
    </parentNodes>
    <childNodes>
        <childNode name="impact_on_D" />
    </childNodes>
</node>
<node nodeId="spec_effort" name="change in effort on specification"
nodeType="Ranked">
    <states>
        <state id="0" name="Extra Lower" />
        <state id="1" name="Much Lower" />
        <state id="2" name="Lower" />
        <state id="3" name="The Same" />
        <state id="4" name="Higher" />
        <state id="5" name="Much Higher" />
        <state id="6" name="Extra Higher" />
    </states>
    <npt type="expression">
        <expression>TNormal(0.5,0.1,0.0,1.0)</expression>
    </npt>
    <parentNodes>
    </parentNodes>
    <childNodes>
        <childNode name="tot_effort_dummy_2" />
        <childNode name="spec_effort_spec_eff" />
    </childNodes>
</node>
<node nodeId="doc_q" name="documentation quality" nodeType="Ranked">
    <states>
        <state id="0" name="Extra Lower" />
        <state id="1" name="Much Lower" />
        <state id="2" name="Lower" />
        <state id="3" name="The Same" />
        <state id="4" name="Higher" />
        <state id="5" name="Much Higher" />
        <state id="6" name="Extra Higher" />
    </states>
    <npt type="expression">
        <expression>Arithmetic(wmean(
7.8, spec_effort_spec_eff,
6.9, spec_ppq,
8.5, req_q))</expression>
    </npt>
    <parentNodes>
        <parentNode name="spec_effort_spec_eff" />
        <parentNode name="spec_ppq" />
        <parentNode name="req_q" />
    </parentNodes>
    <childNodes>
        <childNode name="spec_cod_impact_test" />
        <childNode name="spec_test_impact_on_D" />
        <childNode name="cod_eff" />
    </childNodes>
</node>

```

```

<node nodeId="P_uom" name="productivity rate - unit of measurement"
nodeType="Labelled">
  <states>
    <state id="0" name="FPs / Person-Hours"/>
    <state id="1" name="FPs / Person-Days"/>
    <state id="2" name="FPs / Person-Weeks"/>
    <state id="3" name="FPs / Person-Months"/>
    <state id="4" name="KLOC / Person-Hours"/>
    <state id="5" name="KLOC / Person-Days"/>
    <state id="6" name="KLOC / Person-Weeks"/>
    <state id="7" name="KLOC / Person-Months"/>
    <state id="8" name="Classes / Person-Hours"/>
    <state id="9" name="Classes / Person-Days"/>
    <state id="10" name="Classes / Person-Weeks"/>
    <state id="11" name="Classes / Person-Months"/>
    <state id="12" name="Other"/>
  </states>
  <npt type="manual">
    <value>1.0</value>
    <value>1.0</value>
  </npt>
  <parentNodes>
  </parentNodes>
  <childNodes>
    <childNodes name="prior_P" />
  </childNodes>
</node>
<node nodeId="D_uom" name="defect rate - unit of measurement"
nodeType="Labelled">
  <states>
    <state id="0" name="defects / FP"/>
    <state id="1" name="defects / KLOC"/>
    <state id="2" name="Other"/>
  </states>
  <npt type="manual">
    <value>1.0</value>
    <value>1.0</value>
    <value>1.0</value>
  </npt>
  <parentNodes>
  </parentNodes>
  <childNodes>
    <childNodes name="prior_D" />
  </childNodes>
</node>
<node nodeId="effort_uom" name="effort - unit of measurement"
nodeType="Labelled">
  <states>
    <state id="0" name="Person-Hours"/>
    <state id="1" name="Person-Days"/>
    <state id="2" name="Person-Weeks"/>
    <state id="3" name="Person-Months"/>
    <state id="4" name="Other"/>
  </states>
  <npt type="manual">
    <value>1.0</value>
    <value>1.0</value>
  </npt>

```

```

        <value>1.0</value>
        <value>1.0</value>
        <value>1.0</value>
    </npt>
    <parentNodes>
  </parentNodes>
  <childNodes>
    <childNodes name="prior_effort" />
  </childNodes>
</node>
<node nodeId="proj_compl" name="project complexity" nodeType="Ranked">
  <states>
    <state id="0" name="Extra Lower"/>
    <state id="1" name="Much Lower"/>
    <state id="2" name="Lower"/>
    <state id="3" name="The Same"/>
    <state id="4" name="Higher"/>
    <state id="5" name="Much Higher"/>
    <state id="6" name="Extra Higher"/>
  </states>
  <npt type="expression">
    <expression>TNormal(0.5,0.1,0.0,1.0)</expression>
  </npt>
  <parentNodes>
  </parentNodes>
  <childNodes>
    <childNodes name="uncontrollable_on_P_dummy_2" />
    <childNodes name="uncontrollable_on_D_dummy_2" />
  </childNodes>
</node>
<node nodeId="proj_novel" name="project novelty" nodeType="Ranked">
  <states>
    <state id="0" name="Extra Lower"/>
    <state id="1" name="Much Lower"/>
    <state id="2" name="Lower"/>
    <state id="3" name="The Same"/>
    <state id="4" name="Higher"/>
    <state id="5" name="Much Higher"/>
    <state id="6" name="Extra Higher"/>
  </states>
  <npt type="expression">
    <expression>TNormal(0.5,0.1,0.0,1.0)</expression>
  </npt>
  <parentNodes>
  </parentNodes>
  <childNodes>
    <childNodes name="uncontrollable_on_P_dummy_2" />
    <childNodes name="uncontrollable_on_D_dummy_2" />
  </childNodes>
</node>
<node nodeId="proj_scale" name="project scale" nodeType="Ranked">
  <states>
    <state id="0" name="Extra Lower"/>
    <state id="1" name="Much Lower"/>
    <state id="2" name="Lower"/>
    <state id="3" name="The Same"/>
    <state id="4" name="Higher"/>
    <state id="5" name="Much Higher"/>
    <state id="6" name="Extra Higher"/>
  </states>
  <npt type="expression">
    <expression>TNormal(n_prod_effort_spec_eff,0.01,0.0,1.0)</expression>
  </npt>
  <constants>
    <constant name="impact_of_non_productive_effort" value="3.0"/>
  </constants>
  <parentNodes>

```

```

      <parentNode name="n_prod_effort_spec_eff" />
    </parentNodes>
    <childNodes>
      <childNode name="uncontrollable_on_P_dummy_2" />
      <childNode name="uncontrollable_on_D_dummy_2" />
    </childNodes>
  </node>
  <node nodeId="test_eff" name="testing effectiveness" nodeType="Ranked">
    <states>
      <state id="0" name="Extra Lower"/>
      <state id="1" name="Much Lower"/>
      <state id="2" name="Lower"/>
      <state id="3" name="The Same"/>
      <state id="4" name="Higher"/>
      <state id="5" name="Much Higher"/>
      <state id="6" name="Extra Higher"/>
    </states>
    <npt type="expression">
      <expression>Arithmetic(wmean(
8.3, testing_effort_testing_eff,
8.1, test_ppq,
8.1, spec_cod_impact_test))</expression>
    </npt>
    <parentNodes>
      <parentNode name="testing_effort_testing_eff" />
      <parentNode name="test_ppq" />
      <parentNode name="spec_cod_impact_test" />
    </parentNodes>
    <childNodes>
      <childNode name="spec_test_impact_on_D" />
    </childNodes>
  </node>
  <node nodeId="q_input_doc" name="quality of input documentation"
nodeType="Ranked">
    <states>
      <state id="0" name="Extra Lower"/>
      <state id="1" name="Much Lower"/>
      <state id="2" name="Lower"/>
      <state id="3" name="The Same"/>
      <state id="4" name="Higher"/>
      <state id="5" name="Much Higher"/>
      <state id="6" name="Extra Higher"/>
    </states>
    <npt type="expression">
      <expression>TNormal(0.5,0.1,0.0,1.0)</expression>
    </npt>
    <parentNodes>
    </parentNodes>
    <childNodes>
      <childNode name="uncontrollable_on_D_dummy_1" />
      <childNode name="uncontrollable_on_P_dummy_1" />
    </childNodes>
  </node>
  <node nodeId="pos_cust_inv" name="positive customer involvement"
nodeType="Ranked">
    <states>
      <state id="0" name="Extra Lower"/>
      <state id="1" name="Much Lower"/>
      <state id="2" name="Lower"/>
      <state id="3" name="The Same"/>
      <state id="4" name="Higher"/>
      <state id="5" name="Much Higher"/>
      <state id="6" name="Extra Higher"/>
    </states>
    <npt type="expression">
      <expression>TNormal(0.5,0.1,0.0,1.0)</expression>
    </npt>
    <parentNodes>

```

```

</parentNodes>
<childNodes>
  <childNodes name="uncontrollable_on_D_dummy_1" />
  <childNodes name="uncontrollable_on_P_dummy_1" />
</childNodes>
</node>
<node nodeId="req_stab" name="requirements stability" nodeType="Ranked">
  <states>
    <state id="0" name="Extra Lower"/>
    <state id="1" name="Much Lower"/>
    <state id="2" name="Lower"/>
    <state id="3" name="The Same"/>
    <state id="4" name="Higher"/>
    <state id="5" name="Much Higher"/>
    <state id="6" name="Extra Higher"/>
  </states>
  <npt type="expression">
    <expression>TNormal(0.5,0.3,0.0,1.0)</expression>
  </npt>
  <parentNodes>
  </parentNodes>
  <childNodes>
    <childNodes name="req_q" />
  </childNodes>
</node>
<node nodeId="req_complet" name="requirements completeness"
nodeType="Ranked">
  <states>
    <state id="0" name="Extra Lower"/>
    <state id="1" name="Much Lower"/>
    <state id="2" name="Lower"/>
    <state id="3" name="The Same"/>
    <state id="4" name="Higher"/>
    <state id="5" name="Much Higher"/>
    <state id="6" name="Extra Higher"/>
  </states>
  <npt type="expression">
    <expression>TNormal(0.5,0.3,0.0,1.0)</expression>
  </npt>
  <parentNodes>
  </parentNodes>
  <childNodes>
    <childNodes name="req_q" />
  </childNodes>
</node>
<node nodeId="req_clear" name="requirements clearness" nodeType="Ranked">
  <states>
    <state id="0" name="Extra Lower"/>
    <state id="1" name="Much Lower"/>
    <state id="2" name="Lower"/>
    <state id="3" name="The Same"/>
    <state id="4" name="Higher"/>
    <state id="5" name="Much Higher"/>
    <state id="6" name="Extra Higher"/>
  </states>
  <npt type="expression">
    <expression>TNormal(0.5,0.3,0.0,1.0)</expression>
  </npt>
  <parentNodes>
  </parentNodes>
  <childNodes>
    <childNodes name="req_q" />
  </childNodes>
</node>
<node nodeId="req_q" name="requirements quality" nodeType="Ranked">
  <states>
    <state id="0" name="Extra Lower"/>
    <state id="1" name="Much Lower"/>

```

```

        <state id="2" name="Lower" />
        <state id="3" name="The Same" />
        <state id="4" name="Higher" />
        <state id="5" name="Much Higher" />
        <state id="6" name="Extra Higher" />
    </states>
    <npt type="expression">
        <expression>Arithmetic(wmean(1.5,req_stab,1.5,req_complet,1.0,
req_clear))</expression>
    </npt>
    <parentNodes>
        <parentNode name="req_stab" />
        <parentNode name="req_complet" />
        <parentNode name="req_clear" />
    </parentNodes>
    <childNodes>
        <childNode name="doc_q" />
    </childNodes>
</node>
<node nodeId="test_ppq" name="testing process and people quality"
nodeType="Ranked">
    <states>
        <state id="0" name="Extra Lower" />
        <state id="1" name="Much Lower" />
        <state id="2" name="Lower" />
        <state id="3" name="The Same" />
        <state id="4" name="Higher" />
        <state id="5" name="Much Higher" />
        <state id="6" name="Extra Higher" />
    </states>
    <npt type="expression">
        <expression>Arithmetic(1)</expression>
    </npt>
    <parentNodes>
    </parentNodes>
    <childNodes>
        <childNode name="test_eff" />
    </childNodes>
</node>
<node nodeId="cod_effort" name="change in effort on coding"
nodeType="Ranked">
    <states>
        <state id="0" name="Extra Lower" />
        <state id="1" name="Much Lower" />
        <state id="2" name="Lower" />
        <state id="3" name="The Same" />
        <state id="4" name="Higher" />
        <state id="5" name="Much Higher" />
        <state id="6" name="Extra Higher" />
    </states>
    <npt type="expression">
        <expression>TNormal(0.5,0.1,0.0,1.0)</expression>
    </npt>
    <parentNodes>
    </parentNodes>
    <childNodes>
        <childNode name="cod_eff" />
        <childNode name="tot_effort_dummy_1" />
        <childNode name="testing_effort_testing_eff" />
        <childNode name="spec_effort_spec_eff" />
        <childNode name="n_prod_effort_spec_eff" />
    </childNodes>
</node>
<node nodeId="cod_ppq" name="coding process and people quality"
nodeType="Ranked">
    <states>
        <state id="0" name="Extra Lower" />
        <state id="1" name="Much Lower" />

```

```

        <state id="2" name="Lower" />
        <state id="3" name="The Same" />
        <state id="4" name="Higher" />
        <state id="5" name="Much Higher" />
        <state id="6" name="Extra Higher" />
    </states>
    <npt type="expression">
        <expression>Arithmetic(0)</expression>
    </npt>
    <parentNodes>
    </parentNodes>
    <childNodes>
        <childNode name="cod_eff" />
    </childNodes>
</node>
<node nodeId="spec_ppq" name="specification process and people quality"
nodeType="Ranked">
    <states>
        <state id="0" name="Extra Lower" />
        <state id="1" name="Much Lower" />
        <state id="2" name="Lower" />
        <state id="3" name="The Same" />
        <state id="4" name="Higher" />
        <state id="5" name="Much Higher" />
        <state id="6" name="Extra Higher" />
    </states>
    <npt type="expression">
        <expression>Arithmetic(1)</expression>
    </npt>
    <parentNodes>
    </parentNodes>
    <childNodes>
        <childNode name="doc_q" />
    </childNodes>
</node>
<node nodeId="cod_eff" name="coding effectiveness" nodeType="Ranked">
    <states>
        <state id="0" name="Extra Lower" />
        <state id="1" name="Much Lower" />
        <state id="2" name="Lower" />
        <state id="3" name="The Same" />
        <state id="4" name="Higher" />
        <state id="5" name="Much Higher" />
        <state id="6" name="Extra Higher" />
    </states>
    <npt type="expression">
        <expression>Arithmetic(wmean(
6.8, cod_effort,
7.9, cod_ppq,
6.3, doc_q))</expression>
    </npt>
    <parentNodes>
        <parentNode name="cod_effort" />
        <parentNode name="doc_q" />
        <parentNode name="cod_ppq" />
    </parentNodes>
    <childNodes>
        <childNode name="impact_on_P" />
        <childNode name="controllable_on_D" />
        <childNode name="spec_cod_impact_test" />
    </childNodes>
</node>
<node nodeId="uncontrollable_on_P_dummy_2" name="uncontrollable on
productivity rate dummy 2" nodeType="Ranked">
    <states>
        <state id="0" name="Extra Lower" />
        <state id="1" name="Much Lower" />
        <state id="2" name="Lower" />

```

```

        <state id="3" name="The Same"/>
        <state id="4" name="Higher"/>
        <state id="5" name="Much Higher"/>
        <state id="6" name="Extra Higher"/>
    </states>
    <npt type="expression">
        <expression>Arithmetic(wmean(
5.9,proj_scale,
7.1,proj_compl,
8,proj_novel))</expression>
    </npt>
    <parentNodes>
        <parentNode name="proj_scale" />
        <parentNode name="proj_compl" />
        <parentNode name="proj_novel" />
    </parentNodes>
    <childNodes>
        <childNodes name="uncontrollable_on_P" />
    </childNodes>
</node>
<node nodeId="uncontrollable_on_D_dummy_1" name="uncontrollable on defect
rate dummy 1" nodeType="Ranked">
    <states>
        <state id="0" name="Extra Lower"/>
        <state id="1" name="Much Lower"/>
        <state id="2" name="Lower"/>
        <state id="3" name="The Same"/>
        <state id="4" name="Higher"/>
        <state id="5" name="Much Higher"/>
        <state id="6" name="Extra Higher"/>
    </states>
    <npt type="expression">
        <expression>Arithmetic(wmean(
6.3,q_input_doc,
7.2,pos_cust_inv,
5.5,neg_cust_inv))</expression>
    </npt>
    <parentNodes>
        <parentNode name="q_input_doc" />
        <parentNode name="pos_cust_inv" />
        <parentNode name="neg_cust_inv" />
    </parentNodes>
    <childNodes>
        <childNodes name="uncontrollable_on_D" />
    </childNodes>
</node>
<node nodeId="spec_cod_impact_test" name="spec & coding impact on
testing" nodeType="Ranked">
    <states>
        <state id="0" name="Extra Lower"/>
        <state id="1" name="Much Lower"/>
        <state id="2" name="Lower"/>
        <state id="3" name="The Same"/>
        <state id="4" name="Higher"/>
        <state id="5" name="Much Higher"/>
        <state id="6" name="Extra Higher"/>
    </states>
    <npt type="expression">
        <expression>Arithmetic(wmean(1, cod_eff,2, doc_q))</expression>
    </npt>
    <constants>
        <constant name="adjustment_factor" value="0.5"/>
    </constants>
    <parentNodes>
        <parentNode name="doc_q" />
        <parentNode name="cod_eff" />
    </parentNodes>
    <childNodes>

```

```

        <childNodes name="test_eff" />
    </childNodes>
</node>
<node nodeId="change_total_effort" name="change in total effort"
nodeType="Continuous Interval">
    <states>
        <state id="0" lowerBound="0.0" upperBound="1.0"/>
    </states>
    <npt type="expression">
        <expression>Arithmetic(wmean(1, tot_effort_dummy_1,
1, tot_effort_dummy_2))</expression>
    </npt>
    <parentNodes>
        <parentNode name="tot_effort_dummy_1" />
        <parentNode name="tot_effort_dummy_2" />
    </parentNodes>
    <childNodes>
        <childNodes name="effort_revised" />
        <childNodes name="p_revised" />
    </childNodes>
</node>
<node nodeId="tot_effort_dummy_2" name="total effort dummy 2"
nodeType="Ranked">
    <states>
        <state id="0" name="Extra Lower"/>
        <state id="1" name="Much Lower"/>
        <state id="2" name="Lower"/>
        <state id="3" name="The Same"/>
        <state id="4" name="Higher"/>
        <state id="5" name="Much Higher"/>
        <state id="6" name="Extra Higher"/>
    </states>
    <npt type="expression">
        <expression>Arithmetic(wmean(1, spec_effort, 1,
test_effort))</expression>
    </npt>
    <parentNodes>
        <parentNode name="test_effort" />
        <parentNode name="spec_effort" />
    </parentNodes>
    <childNodes>
        <childNodes name="change_total_effort" />
    </childNodes>
</node>
<node nodeId="tot_effort_dummy_1" name="total effort dummy 1"
nodeType="Ranked">
    <states>
        <state id="0" name="Extra Lower"/>
        <state id="1" name="Much Lower"/>
        <state id="2" name="Lower"/>
        <state id="3" name="The Same"/>
        <state id="4" name="Higher"/>
        <state id="5" name="Much Higher"/>
        <state id="6" name="Extra Higher"/>
    </states>
    <npt type="expression">
        <expression>Arithmetic(wmean(1, cod_effort, 1,
n_productive_effort))</expression>
    </npt>
    <parentNodes>
        <parentNode name="n_productive_effort" />
        <parentNode name="cod_effort" />
    </parentNodes>
    <childNodes>
        <childNodes name="change_total_effort" />
    </childNodes>
</node>

```

```

<node nodeId="n_productive_effort" name="change in non-productive effort"
nodeType="Ranked">
  <states>
    <state id="0" name="Extra Lower"/>
    <state id="1" name="Much Lower"/>
    <state id="2" name="Lower"/>
    <state id="3" name="The Same"/>
    <state id="4" name="Higher"/>
    <state id="5" name="Much Higher"/>
    <state id="6" name="Extra Higher"/>
  </states>
  <npt type="expression">
    <expression>TNormal(0.5,0.1,0.0,1.0)</expression>
  </npt>
  <constants>
    <constant name="impact_of_project_scale" value="0.5"/>
  </constants>
  <parentNodes>
  </parentNodes>
  <childNodes>
    <childNodes name="tot_effort_dummy_1" />
    <childNodes name="n_prod_effort_spec_eff" />
  </childNodes>
</node>
<node nodeId="dead_pres" name="deadline pressure" nodeType="Ranked">
  <states>
    <state id="0" name="Extra Lower"/>
    <state id="1" name="Much Lower"/>
    <state id="2" name="Lower"/>
    <state id="3" name="The Same"/>
    <state id="4" name="Higher"/>
    <state id="5" name="Much Higher"/>
    <state id="6" name="Extra Higher"/>
  </states>
  <npt type="expression">
    <expression>TNormal(0.5,0.1,0.0,1.0)</expression>
  </npt>
  <parentNodes>
  </parentNodes>
  <childNodes>
    <childNodes name="uncontrollable_on_P" />
    <childNodes name="uncontrollable_on_D" />
  </childNodes>
</node>
<node nodeId="impact_on_P" name="impact on productivity"
nodeType="Continuous Interval">
  <states>
    <state id="0" lowerBound="0.0" upperBound="1.0"/>
  </states>
  <npt type="expression">
    <expression>Arithmetic(wmin(1.581, uncontrollable_on_P,
2.422, cod_eff))</expression>
  </npt>
  <parentNodes>
    <parentNode name="uncontrollable_on_P" />
    <parentNode name="cod_eff" />
  </parentNodes>
  <childNodes>
    <childNodes name="p_revised_dummy" />
  </childNodes>
</node>
<node nodeId="impact_on_D" name="impact on defect rate"
nodeType="Continuous Interval">
  <states>
    <state id="0" lowerBound="0.0" upperBound="1.0"/>
  </states>
  <npt type="expression">

```

```

        <expression>Arithmetic(wmin(3.615, controllable_on_D, 1.667,
uncontrollable_on_D))</expression>
    </npt>
    <parentNodes>
        <parentNode name="controllable_on_D" />
        <parentNode name="uncontrollable_on_D" />
    </parentNodes>
    <childNodes>
        <childNode name="d_revised" />
    </childNodes>
</node>
<node nodeId="controllable_on_D" name="controllable on defect rate"
nodeType="Continuous Interval">
    <states>
        <state id="0" lowerBound="0.0" upperBound="1.0"/>
    </states>
    <npt type="expression">
        <expression>Arithmetic(wmean(15.53, spec_test_impact_on_D, 6.5,
cod_eff))</expression>
    </npt>
    <parentNodes>
        <parentNode name="cod_eff" />
        <parentNode name="spec_test_impact_on_D" />
    </parentNodes>
    <childNodes>
        <childNode name="impact_on_D" />
    </childNodes>
</node>
<node nodeId="spec_test_impact_on_D" name="spec&test impact on defect
rate" nodeType="Continuous Interval">
    <states>
        <state id="0" lowerBound="0.0" upperBound="1.0"/>
    </states>
    <npt type="expression">
<expression>Arithmetic(wmean(8.3, test_eff, 7.23, doc_q))</expression>
    </npt>
    <parentNodes>
        <parentNode name="test_eff" />
        <parentNode name="doc_q" />
    </parentNodes>
    <childNodes>
        <childNode name="controllable_on_D" />
    </childNodes>
</node>
<node nodeId="defect_thres" name="Defect threshold" nodeType="Boolean">
    <states>
        <state id="0" name="No"/>
        <state id="1" name="Yes"/>
    </states>
    <npt type="expression">
        <expression>Comparative(if ((num_defects >= 0) &&
(num_defects <= 40), "Yes",
"No"))</expression>
    </npt>
    <parentNodes>
        <parentNode name="num_defects" />
    </parentNodes>
    <childNodes>
    </childNodes>
</node>
<node nodeId="prior_effort" name="prior effort" nodeType="Continuous
Interval">
    <states>
        <state id="0" lowerBound="0.0" upperBound="10.0"/>
        <state id="1" lowerBound="10.0" upperBound="Infinity"/>
    </states>
    <npt type="partitioned expression">

```

```

        <expression>Log Normal(7.6671,1.4521)</expression>
        <expression>Log Normal(5.5877,1.4521)</expression>
        <expression>Log Normal(3.9783,1.4521)</expression>
        <expression>Log Normal(2.4742,1.4521)</expression>
        <expression>Normal(0,1.0E20)</expression>
    </npt>
    <parentNodes>
        <parentNode name="effort_uom" />
    </parentNodes>
    <childNodes>
        <childNode name="effort_revised" />
    </childNodes>
</node>
<node nodeId="testing_effort_testing_eff" name="impact of testing effort
on testing effectiveness" nodeType="Ranked">
    <states>
        <state id="0" name="Extra Lower"/>
        <state id="1" name="Much Lower"/>
        <state id="2" name="Lower"/>
        <state id="3" name="The Same"/>
        <state id="4" name="Higher"/>
        <state id="5" name="Much Higher"/>
        <state id="6" name="Extra Higher"/>
    </states>
    <npt type="expression">
        <expression>Arithmetic(max(0, min(1, (test_effort - cod_effort) /
2 + 0.5)))</expression>
    </npt>
    <parentNodes>
        <parentNode name="cod_effort" />
        <parentNode name="test_effort" />
    </parentNodes>
    <childNodes>
        <childNode name="test_eff" />
    </childNodes>
</node>
<node nodeId="spec_effort_spec_eff" name="impact of specification effort
on specification effectiveness" nodeType="Ranked">
    <states>
        <state id="0" name="Extra Lower"/>
        <state id="1" name="Much Lower"/>
        <state id="2" name="Lower"/>
        <state id="3" name="The Same"/>
        <state id="4" name="Higher"/>
        <state id="5" name="Much Higher"/>
        <state id="6" name="Extra Higher"/>
    </states>
    <npt type="expression">
        <expression>Arithmetic(max(0, min(1, (spec_effort - cod_effort) /
2 + 0.5)))</expression>
    </npt>
    <parentNodes>
        <parentNode name="cod_effort" />
        <parentNode name="spec_effort" />
    </parentNodes>
    <childNodes>
        <childNode name="doc_q" />
    </childNodes>
</node>
<node nodeId="n_prod_effort_spec_eff" name="impact of non productive
effort on project scale" nodeType="Ranked">
    <states>
        <state id="0" name="Extra Lower"/>
        <state id="1" name="Much Lower"/>
        <state id="2" name="Lower"/>
        <state id="3" name="The Same"/>
        <state id="4" name="Higher"/>
        <state id="5" name="Much Higher"/>

```

```

        <state id="6" name="Extra Higher"/>
    </states>
    <npt type="expression">
        <expression>Arithmetic( (n_productive_effort - cod_effort) / 2 +
0.5)</expression>
    </npt>
    <parentNodes>
        <parentNode name="cod_effort" />
        <parentNode name="n_productive_effort" />
    </parentNodes>
    <childNodes>
        <childNodes name="proj_scale" />
    </childNodes>
</node>
<node nodeId="p_revised_dummy" name="revised productivity rate dummy"
nodeType="Continuous Interval">
    <states>
        <state id="0" lowerBound="0.0" upperBound="1.0"/>
        <state id="1" lowerBound="1.0" upperBound="Infinity"/>
    </states>
    <npt type="expression">
        <expression>Arithmetic(prior_P * 2.3 ^ (4 * impact_on_P -
2))</expression>
    </npt>
    <constants>
        <constant name="adjustment_factor" value="1.0"/>
    </constants>
    <parentNodes>
        <parentNode name="prior_P" />
        <parentNode name="impact_on_P" />
    </parentNodes>
    <childNodes>
        <childNodes name="p_revised" />
    </childNodes>
</node>
<node nodeId="neg_cust_inv" name="negative customer involvement"
nodeType="Ranked">
    <states>
        <state id="0" name="Extra Lower"/>
        <state id="1" name="Much Lower"/>
        <state id="2" name="Lower"/>
        <state id="3" name="The Same"/>
        <state id="4" name="Higher"/>
        <state id="5" name="Much Higher"/>
        <state id="6" name="Extra Higher"/>
    </states>
    <npt type="expression">
        <expression>TNormal(0.5,0.1,0.0,1.0)</expression>
    </npt>
    <parentNodes>
    </parentNodes>
    <childNodes>
        <childNodes name="uncontrollable_on_D_dummy_1" />
        <childNodes name="uncontrollable_on_P_dummy_1" />
    </childNodes>
</node>
<node nodeId="uncontrollable_on_P_dummy_1" name="uncontrollable on
productivity rate dummy 1" nodeType="Ranked">
    <states>
        <state id="0" name="Extra Lower"/>
        <state id="1" name="Much Lower"/>
        <state id="2" name="Lower"/>
        <state id="3" name="The Same"/>
        <state id="4" name="Higher"/>
        <state id="5" name="Much Higher"/>
        <state id="6" name="Extra Higher"/>
    </states>
    <npt type="expression">

```

```

        <expression>Arithmetic(wmean(
6.1,q_input_doc,
5.4,pos_cust_inv,
4.8,1-neg_cust_inv))</expression>
    </npt>
    <parentNodes>
        <parentNode name="pos_cust_inv" />
        <parentNode name="q_input_doc" />
        <parentNode name="neg_cust_inv" />
    </parentNodes>
    <childNodes>
        <childNode name="uncontrollable_on_P" />
    </childNodes>
</node>
<node nodeId="uncontrollable_on_D_dummy_2" name="uncontrollable on defect
rate dummy 2" nodeType="Ranked">
    <states>
        <state id="0" name="Extra Lower"/>
        <state id="1" name="Much Lower"/>
        <state id="2" name="Lower"/>
        <state id="3" name="The Same"/>
        <state id="4" name="Higher"/>
        <state id="5" name="Much Higher"/>
        <state id="6" name="Extra Higher"/>
    </states>
    <npt type="expression">
        <expression>Arithmetic(wmean(
5.3,proj_scale,
7.4,proj_compl,
6.8,proj_novel))</expression>
    </npt>
    <parentNodes>
        <parentNode name="proj_scale" />
        <parentNode name="proj_compl" />
        <parentNode name="proj_novel" />
    </parentNodes>
    <childNodes>
        <childNode name="uncontrollable_on_D" />
    </childNodes>
</node>
</object>

<?xml version="1.0" encoding="iso-8859-2" ?>
<object name="PPQ" type="structure">
    <node nodeId="test_ppq" name="testing process and people quality"
nodeType="Ranked">
        <states>
            <state id="0" name="Extra Lower"/>
            <state id="1" name="Much Lower"/>
            <state id="2" name="Lower"/>
            <state id="3" name="The Same"/>
            <state id="4" name="Higher"/>
            <state id="5" name="Much Higher"/>
            <state id="6" name="Extra Higher"/>
        </states>
        <npt type="expression">
            <expression>TNormal(wmean(23.7,dummy_testing_peo,
20.8,dummy_test_proc_2,
14.2,dummy_test_proc_1),5.0E-4,0.0,1.0)</expression>
        </npt>
        <parentNodes>
            <parentNode name="dummy_testing_peo" />
            <parentNode name="dummy_test_proc_2" />
            <parentNode name="dummy_test_proc_1" />
        </parentNodes>
        <childNodes>
        </childNodes>
    </node>

```

```

    <node nodeId="cod_ppq" name="coding process and people quality"
nodeType="Ranked">
    <states>
        <state id="0" name="Extra Lower"/>
        <state id="1" name="Much Lower"/>
        <state id="2" name="Lower"/>
        <state id="3" name="The Same"/>
        <state id="4" name="Higher"/>
        <state id="5" name="Much Higher"/>
        <state id="6" name="Extra Higher"/>
    </states>
    <npt type="expression">
        <expression>TNormal(wmean(23.7,dummy_coding_peo,
20.8,dummy_cod_proc_2,
14.2,dummy_cod_proc_1),5.0E-4,0.0,1.0)</expression>
    </npt>
    <parentNodes>
        <parentNode name="dummy_coding_peo" />
        <parentNode name="dummy_cod_proc_2" />
        <parentNode name="dummy_cod_proc_1" />
    </parentNodes>
    <childNodes>
    </childNodes>
</node>
    <node nodeId="spec_ppq" name="specification process and people quality"
nodeType="Ranked">
    <states>
        <state id="0" name="Extra Lower"/>
        <state id="1" name="Much Lower"/>
        <state id="2" name="Lower"/>
        <state id="3" name="The Same"/>
        <state id="4" name="Higher"/>
        <state id="5" name="Much Higher"/>
        <state id="6" name="Extra Higher"/>
    </states>
    <npt type="expression">
        <expression>TNormal(wmean(23.7,dummy_spec_peo,
20.8,dummy_spec_proc_2,
14.2,dummy_spec_proc_1),5.0E-4,0.0,1.0)</expression>
    </npt>
    <parentNodes>
        <parentNode name="dummy_spec_peo" />
        <parentNode name="dummy_spec_proc_1" />
        <parentNode name="dummy_spec_proc_2" />
    </parentNodes>
    <childNodes>
    </childNodes>
</node>
    <node nodeId="spec_staff_motiv" name="specification staff motivation"
nodeType="Ranked">
    <states>
        <state id="0" name="Extra Lower"/>
        <state id="1" name="Much Lower"/>
        <state id="2" name="Lower"/>
        <state id="3" name="The Same"/>
        <state id="4" name="Higher"/>
        <state id="5" name="Much Higher"/>
        <state id="6" name="Extra Higher"/>
    </states>
    <npt type="expression">
        <expression>TNormal(staff_motiv,0.01,0.0,1.0)</expression>
    </npt>
    <parentNodes>
        <parentNode name="staff_motiv" />
    </parentNodes>
    <childNodes>
        <childNode name="dummy_spec_peo" />
    </childNodes>

```

```

</node>
<node nodeId="dummy_spec_peo" name="dummy spec people" nodeType="Ranked">
  <states>
    <state id="0" name="Extra Lower"/>
    <state id="1" name="Much Lower"/>
    <state id="2" name="Lower"/>
    <state id="3" name="The Same"/>
    <state id="4" name="Higher"/>
    <state id="5" name="Much Higher"/>
    <state id="6" name="Extra Higher"/>
  </states>
  <npt type="expression">
    <expression>TNormal(wmean(8.7,spec_staff_motiv,8.3,
spec_staff_exper,6.7,spec_staff_educ),0.001,0.0,1.0)</expression>
  </npt>
  <parentNodes>
    <parentNode name="spec_staff_educ" />
    <parentNode name="spec_staff_exper" />
    <parentNode name="spec_staff_motiv" />
  </parentNodes>
  <childNodes>
    <childNode name="spec_ppq" />
  </childNodes>
</node>
<node nodeId="dummy_spec_proc_2" name="dummy spec proc 2"
nodeType="Ranked">
  <states>
    <state id="0" name="Extra Lower"/>
    <state id="1" name="Much Lower"/>
    <state id="2" name="Lower"/>
    <state id="3" name="The Same"/>
    <state id="4" name="Higher"/>
    <state id="5" name="Much Higher"/>
    <state id="6" name="Extra Higher"/>
  </states>
  <npt type="expression">
    <expression>TNormal(wmean(5.8,spec_def_proc,
7.2,spec_lead_q,
7.8,spec_team_org),0.0010,0.0,1.0)</expression>
  </npt>
  <parentNodes>
    <parentNode name="spec_def_proc" />
    <parentNode name="spec_lead_q" />
    <parentNode name="spec_team_org" />
  </parentNodes>
  <childNodes>
    <childNode name="spec_ppq" />
  </childNodes>
</node>
<node nodeId="spec_staff_educ" name="spec staff education"
nodeType="Ranked">
  <states>
    <state id="0" name="Extra Lower"/>
    <state id="1" name="Much Lower"/>
    <state id="2" name="Lower"/>
    <state id="3" name="The Same"/>
    <state id="4" name="Higher"/>
    <state id="5" name="Much Higher"/>
    <state id="6" name="Extra Higher"/>
  </states>
  <npt type="expression">
    <expression>TNormal(staff_educ,0.01,0.0,1.0)</expression>
  </npt>
  <parentNodes>
    <parentNode name="staff_educ" />
  </parentNodes>
  <childNodes>
    <childNode name="dummy_spec_peo" />
  </childNodes>

```

```

    </childNodes>
  </node>
  <node nodeId="spec_def_proc" name="defined process followed in
specification" nodeType="Ranked">
    <states>
      <state id="0" name="Extra Lower"/>
      <state id="1" name="Much Lower"/>
      <state id="2" name="Lower"/>
      <state id="3" name="The Same"/>
      <state id="4" name="Higher"/>
      <state id="5" name="Much Higher"/>
      <state id="6" name="Extra Higher"/>
    </states>
    <npt type="expression">
      <expression>TNormal(def_proc,0.01,0.0,1.0)</expression>
    </npt>
    <parentNodes>
      <parentNode name="def_proc" />
    </parentNodes>
    <childNodes>
      <childNodes name="dummy_spec_proc_2" />
    </childNodes>
  </node>
  <node nodeId="spec_staff_exper" name="specification staff experience"
nodeType="Ranked">
    <states>
      <state id="0" name="Extra Lower"/>
      <state id="1" name="Much Lower"/>
      <state id="2" name="Lower"/>
      <state id="3" name="The Same"/>
      <state id="4" name="Higher"/>
      <state id="5" name="Much Higher"/>
      <state id="6" name="Extra Higher"/>
    </states>
    <npt type="expression">
      <expression>TNormal(staff_exper,0.01,0.0,1.0)</expression>
    </npt>
    <parentNodes>
      <parentNode name="staff_exper" />
    </parentNodes>
    <childNodes>
      <childNodes name="dummy_spec_peo" />
    </childNodes>
  </node>
  <node nodeId="spec_team_org" name="team organization in specification"
nodeType="Ranked">
    <states>
      <state id="0" name="Extra Lower"/>
      <state id="1" name="Much Lower"/>
      <state id="2" name="Lower"/>
      <state id="3" name="The Same"/>
      <state id="4" name="Higher"/>
      <state id="5" name="Much Higher"/>
      <state id="6" name="Extra Higher"/>
    </states>
    <npt type="expression">
      <expression>TNormal(team_org,0.01,0.0,1.0)</expression>
    </npt>
    <parentNodes>
      <parentNode name="team_org" />
    </parentNodes>
    <childNodes>
      <childNodes name="dummy_spec_proc_2" />
    </childNodes>
  </node>
  <node nodeId="spec_lead_q" name="leadership quality in specification"
nodeType="Ranked">
    <states>

```

```

        <state id="0" name="Extra Lower"/>
        <state id="1" name="Much Lower"/>
        <state id="2" name="Lower"/>
        <state id="3" name="The Same"/>
        <state id="4" name="Higher"/>
        <state id="5" name="Much Higher"/>
        <state id="6" name="Extra Higher"/>
    </states>
    <npt type="expression">
        <expression>TNormal(lead_q,0.01,0.0,1.0)</expression>
    </npt>
    <parentNodes>
        <parentNode name="lead_q" />
    </parentNodes>
    <childNodes>
        <childNode name="dummy_spec_proc_2" />
    </childNodes>
</node>
<node nodeId="spec_dist_comm" name="level of distributed communication in
specification" nodeType="Ranked">
    <states>
        <state id="0" name="Extra Lower"/>
        <state id="1" name="Much Lower"/>
        <state id="2" name="Lower"/>
        <state id="3" name="The Same"/>
        <state id="4" name="Higher"/>
        <state id="5" name="Much Higher"/>
        <state id="6" name="Extra Higher"/>
    </states>
    <npt type="expression">
        <expression>TNormal(dist_comm,0.01,0.0,1.0)</expression>
    </npt>
    <parentNodes>
        <parentNode name="dist_comm" />
    </parentNodes>
    <childNodes>
        <childNode name="dummy_spec_proc_1" />
    </childNodes>
</node>
<node nodeId="spec_app_meth_tool" name="appropriateness of methods and
tools used in specification" nodeType="Ranked">
    <states>
        <state id="0" name="Extra Lower"/>
        <state id="1" name="Much Lower"/>
        <state id="2" name="Lower"/>
        <state id="3" name="The Same"/>
        <state id="4" name="Higher"/>
        <state id="5" name="Much Higher"/>
        <state id="6" name="Extra Higher"/>
    </states>
    <npt type="expression">
        <expression>TNormal(app_meth_tool,0.01,0.0,1.0)</expression>
    </npt>
    <parentNodes>
        <parentNode name="app_meth_tool" />
    </parentNodes>
    <childNodes>
        <childNode name="dummy_spec_proc_1" />
    </childNodes>
</node>
<node nodeId="cod_staff_motiv" name="coding staff motivation"
nodeType="Ranked">
    <states>
        <state id="0" name="Extra Lower"/>
        <state id="1" name="Much Lower"/>
        <state id="2" name="Lower"/>
        <state id="3" name="The Same"/>
        <state id="4" name="Higher"/>
    </states>

```

```

        <state id="5" name="Much Higher"/>
        <state id="6" name="Extra Higher"/>
    </states>
    <npt type="expression">
        <expression>TNormal(staff_motiv,0.01,0.0,1.0)</expression>
    </npt>
    <parentNodes>
        <parentNode name="staff_motiv" />
    </parentNodes>
    <childNodes>
        <childNode name="dummy_coding_peo" />
    </childNodes>
</node>
<node nodeId="dummy_coding_peo" name="dummy coding people"
nodeType="Ranked">
    <states>
        <state id="0" name="Extra Lower"/>
        <state id="1" name="Much Lower"/>
        <state id="2" name="Lower"/>
        <state id="3" name="The Same"/>
        <state id="4" name="Higher"/>
        <state id="5" name="Much Higher"/>
        <state id="6" name="Extra Higher"/>
    </states>
    <npt type="expression">
        <expression>TNormal(wmean(8.7,cod_staff_motiv,8.3,
cod_staff_exper,6.7,cod_staff_educ),0.001,0.0,1.0)</expression>
    </npt>
    <parentNodes>
        <parentNode name="cod_staff_motiv" />
        <parentNode name="cod_staff_exper" />
        <parentNode name="cod_staff_educ" />
    </parentNodes>
    <childNodes>
        <childNode name="cod_ppq" />
    </childNodes>
</node>
<node nodeId="cod_staff_educ" name="coding staff education"
nodeType="Ranked">
    <states>
        <state id="0" name="Extra Lower"/>
        <state id="1" name="Much Lower"/>
        <state id="2" name="Lower"/>
        <state id="3" name="The Same"/>
        <state id="4" name="Higher"/>
        <state id="5" name="Much Higher"/>
        <state id="6" name="Extra Higher"/>
    </states>
    <npt type="expression">
        <expression>TNormal(staff_educ,0.01,0.0,1.0)</expression>
    </npt>
    <parentNodes>
        <parentNode name="staff_educ" />
    </parentNodes>
    <childNodes>
        <childNode name="dummy_coding_peo" />
    </childNodes>
</node>
<node nodeId="cod_staff_exper" name="coding staff experience"
nodeType="Ranked">
    <states>
        <state id="0" name="Extra Lower"/>
        <state id="1" name="Much Lower"/>
        <state id="2" name="Lower"/>
        <state id="3" name="The Same"/>
        <state id="4" name="Higher"/>
        <state id="5" name="Much Higher"/>
        <state id="6" name="Extra Higher"/>
    </states>

```

```

</states>
<npt type="expression">
  <expression>TNormal(staff_exper,0.01,0.0,1.0)</expression>
</npt>
<parentNodes>
  <parentNode name="staff_exper" />
</parentNodes>
<childNodes>
  <childNode name="dummy_coding_peo" />
</childNodes>
</node>
<node nodeId="test_staff_motiv" name="testing staff motivation"
nodeType="Ranked">
  <states>
    <state id="0" name="Extra Lower"/>
    <state id="1" name="Much Lower"/>
    <state id="2" name="Lower"/>
    <state id="3" name="The Same"/>
    <state id="4" name="Higher"/>
    <state id="5" name="Much Higher"/>
    <state id="6" name="Extra Higher"/>
  </states>
  <npt type="expression">
    <expression>TNormal(staff_motiv,0.01,0.0,1.0)</expression>
  </npt>
  <parentNodes>
    <parentNode name="staff_motiv" />
  </parentNodes>
  <childNodes>
    <childNode name="dummy_testing_peo" />
  </childNodes>
</node>
<node nodeId="dummy_testing_peo" name="dummy testing people"
nodeType="Ranked">
  <states>
    <state id="0" name="Extra Lower"/>
    <state id="1" name="Much Lower"/>
    <state id="2" name="Lower"/>
    <state id="3" name="The Same"/>
    <state id="4" name="Higher"/>
    <state id="5" name="Much Higher"/>
    <state id="6" name="Extra Higher"/>
  </states>
  <npt type="expression">
    <expression>TNormal(wmean(8.7,test_staff_motiv,8.3,
test_staff_exper,6.7,test_staff_educ),0.001,0.0,1.0)</expression>
  </npt>
  <parentNodes>
    <parentNode name="test_staff_motiv" />
    <parentNode name="test_staff_exper" />
    <parentNode name="test_staff_educ" />
  </parentNodes>
  <childNodes>
    <childNode name="test_ppq" />
  </childNodes>
</node>
<node nodeId="test_staff_educ" name="testing staff education"
nodeType="Ranked">
  <states>
    <state id="0" name="Extra Lower"/>
    <state id="1" name="Much Lower"/>
    <state id="2" name="Lower"/>
    <state id="3" name="The Same"/>
    <state id="4" name="Higher"/>
    <state id="5" name="Much Higher"/>
    <state id="6" name="Extra Higher"/>
  </states>
  <npt type="expression">

```

```

        <expression>TNormal(staff_educ,0.01,0.0,1.0)</expression>
</npt>
<parentNodes>
  <parentNode name="staff_educ" />
</parentNodes>
<childNodes>
  <childNode name="dummy_testing_peo" />
</childNodes>
</node>
<node nodeId="test_staff_exper" name="testing staff experience"
nodeType="Ranked">
  <states>
    <state id="0" name="Extra Lower"/>
    <state id="1" name="Much Lower"/>
    <state id="2" name="Lower"/>
    <state id="3" name="The Same"/>
    <state id="4" name="Higher"/>
    <state id="5" name="Much Higher"/>
    <state id="6" name="Extra Higher"/>
  </states>
  <npt type="expression">
    <expression>TNormal(staff_exper,0.01,0.0,1.0)</expression>
  </npt>
  <parentNodes>
    <parentNode name="staff_exper" />
  </parentNodes>
  <childNodes>
    <childNode name="dummy_testing_peo" />
  </childNodes>
</node>
<node nodeId="staff_motiv" name="staff motivation" nodeType="Ranked">
  <states>
    <state id="0" name="Extra Lower"/>
    <state id="1" name="Much Lower"/>
    <state id="2" name="Lower"/>
    <state id="3" name="The Same"/>
    <state id="4" name="Higher"/>
    <state id="5" name="Much Higher"/>
    <state id="6" name="Extra Higher"/>
  </states>
  <npt type="expression">
    <expression>TNormal(0.5,0.1,0.0,1.0)</expression>
  </npt>
  <parentNodes>
  </parentNodes>
  <childNodes>
    <childNode name="spec_staff_motiv" />
    <childNode name="cod_staff_motiv" />
    <childNode name="test_staff_motiv" />
  </childNodes>
</node>
<node nodeId="staff_educ" name="staff education" nodeType="Ranked">
  <states>
    <state id="0" name="Extra Lower"/>
    <state id="1" name="Much Lower"/>
    <state id="2" name="Lower"/>
    <state id="3" name="The Same"/>
    <state id="4" name="Higher"/>
    <state id="5" name="Much Higher"/>
    <state id="6" name="Extra Higher"/>
  </states>
  <npt type="expression">
    <expression>TNormal(0.5,0.1,0.0,1.0)</expression>
  </npt>
  <parentNodes>
  </parentNodes>
  <childNodes>
    <childNode name="spec_staff_educ" />

```

```

        <childNodes name="cod_staff_educ" />
        <childNodes name="test_staff_educ" />
    </childNodes>
</node>
<node nodeId="staff_exper" name="staff experience" nodeType="Ranked">
    <states>
        <state id="0" name="Extra Lower"/>
        <state id="1" name="Much Lower"/>
        <state id="2" name="Lower"/>
        <state id="3" name="The Same"/>
        <state id="4" name="Higher"/>
        <state id="5" name="Much Higher"/>
        <state id="6" name="Extra Higher"/>
    </states>
    <npt type="expression">
        <expression>TNormal(0.5,0.1,0.0,1.0)</expression>
    </npt>
    <parentNodes>
    </parentNodes>
    <childNodes>
        <childNodes name="spec_staff_exper" />
        <childNodes name="cod_staff_exper" />
        <childNodes name="test_staff_exper" />
    </childNodes>
</node>
<node nodeId="dummy_spec_proc_1" name="dummy spec proc 1"
nodeType="Ranked">
    <states>
        <state id="0" name="Extra Lower"/>
        <state id="1" name="Much Lower"/>
        <state id="2" name="Lower"/>
        <state id="3" name="The Same"/>
        <state id="4" name="Higher"/>
        <state id="5" name="Much Higher"/>
        <state id="6" name="Extra Higher"/>
    </states>
    <npt type="expression">
        <expression>TNormal(wmean(7.4,spec_app_meth_tool,
6.8,spec_dist_comm),0.0010,0.0,1.0)</expression>
    </npt>
    <parentNodes>
        <parentNode name="spec_app_meth_tool" />
        <parentNode name="spec_dist_comm" />
    </parentNodes>
    <childNodes>
        <childNodes name="spec_ppq" />
    </childNodes>
</node>
<node nodeId="def_proc" name="defined process followed" nodeType="Ranked">
    <states>
        <state id="0" name="Extra Lower"/>
        <state id="1" name="Much Lower"/>
        <state id="2" name="Lower"/>
        <state id="3" name="The Same"/>
        <state id="4" name="Higher"/>
        <state id="5" name="Much Higher"/>
        <state id="6" name="Extra Higher"/>
    </states>
    <npt type="expression">
        <expression>TNormal(0.5,0.1,0.0,1.0)</expression>
    </npt>
    <parentNodes>
    </parentNodes>
    <childNodes>
        <childNodes name="test_def_proc" />
        <childNodes name="cod_def_proc" />
        <childNodes name="spec_def_proc" />
    </childNodes>

```



```

        <state id="0" name="Extra Lower"/>
        <state id="1" name="Much Lower"/>
        <state id="2" name="Lower"/>
        <state id="3" name="The Same"/>
        <state id="4" name="Higher"/>
        <state id="5" name="Much Higher"/>
        <state id="6" name="Extra Higher"/>
    </states>
    <npt type="expression">
        <expression>TNormal(0.5,0.1,0.0,1.0)</expression>
    </npt>
    <parentNodes>
    </parentNodes>
    <childNodes>
        <childNodes name="test_app_meth_tool" />
        <childNodes name="cod_app_meth_tool" />
        <childNodes name="spec_app_meth_tool" />
    </childNodes>
</node>
<node nodeId="cod_def_proc" name="defined process followed in coding"
nodeType="Ranked">
    <states>
        <state id="0" name="Extra Lower"/>
        <state id="1" name="Much Lower"/>
        <state id="2" name="Lower"/>
        <state id="3" name="The Same"/>
        <state id="4" name="Higher"/>
        <state id="5" name="Much Higher"/>
        <state id="6" name="Extra Higher"/>
    </states>
    <npt type="expression">
        <expression>TNormal(def_proc,0.01,0.0,1.0)</expression>
    </npt>
    <parentNodes>
        <parentNode name="def_proc" />
    </parentNodes>
    <childNodes>
        <childNodes name="dummy_cod_proc_2" />
    </childNodes>
</node>
<node nodeId="cod_team_org" name="team organization in coding"
nodeType="Ranked">
    <states>
        <state id="0" name="Extra Lower"/>
        <state id="1" name="Much Lower"/>
        <state id="2" name="Lower"/>
        <state id="3" name="The Same"/>
        <state id="4" name="Higher"/>
        <state id="5" name="Much Higher"/>
        <state id="6" name="Extra Higher"/>
    </states>
    <npt type="expression">
        <expression>TNormal(team_org,0.01,0.0,1.0)</expression>
    </npt>
    <parentNodes>
        <parentNode name="team_org" />
    </parentNodes>
    <childNodes>
        <childNodes name="dummy_cod_proc_2" />
    </childNodes>
</node>
<node nodeId="cod_lead_q" name="leadership quality in coding"
nodeType="Ranked">
    <states>
        <state id="0" name="Extra Lower"/>
        <state id="1" name="Much Lower"/>
        <state id="2" name="Lower"/>
        <state id="3" name="The Same"/>

```

```

        <state id="4" name="Higher" />
        <state id="5" name="Much Higher" />
        <state id="6" name="Extra Higher" />
    </states>
    <npt type="expression">
        <expression>TNormal(lead_q,0.01,0.0,1.0)</expression>
    </npt>
    <parentNodes>
        <parentNode name="lead_q" />
    </parentNodes>
    <childNodes>
        <childNode name="dummy_cod_proc_2" />
    </childNodes>
</node>
<node nodeId="cod_dist_comm" name="level of distributed communication in
coding" nodeType="Ranked">
    <states>
        <state id="0" name="Extra Lower" />
        <state id="1" name="Much Lower" />
        <state id="2" name="Lower" />
        <state id="3" name="The Same" />
        <state id="4" name="Higher" />
        <state id="5" name="Much Higher" />
        <state id="6" name="Extra Higher" />
    </states>
    <npt type="expression">
        <expression>TNormal(dist_comm,0.01,0.0,1.0)</expression>
    </npt>
    <parentNodes>
        <parentNode name="dist_comm" />
    </parentNodes>
    <childNodes>
        <childNode name="dummy_cod_proc_1" />
    </childNodes>
</node>
<node nodeId="cod_app_meth_tool" name="appropriateness of methods and
tools used in coding" nodeType="Ranked">
    <states>
        <state id="0" name="Extra Lower" />
        <state id="1" name="Much Lower" />
        <state id="2" name="Lower" />
        <state id="3" name="The Same" />
        <state id="4" name="Higher" />
        <state id="5" name="Much Higher" />
        <state id="6" name="Extra Higher" />
    </states>
    <npt type="expression">
        <expression>TNormal(app_meth_tool,0.01,0.0,1.0)</expression>
    </npt>
    <parentNodes>
        <parentNode name="app_meth_tool" />
    </parentNodes>
    <childNodes>
        <childNode name="dummy_cod_proc_1" />
    </childNodes>
</node>
<node nodeId="test_def_proc" name="defined process followed in testing"
nodeType="Ranked">
    <states>
        <state id="0" name="Extra Lower" />
        <state id="1" name="Much Lower" />
        <state id="2" name="Lower" />
        <state id="3" name="The Same" />
        <state id="4" name="Higher" />
        <state id="5" name="Much Higher" />
        <state id="6" name="Extra Higher" />
    </states>
    <npt type="expression">

```

```

        <expression>TNormal(def_proc,0.01,0.0,1.0)</expression>
    </npt>
    <parentNodes>
        <parentNode name="def_proc" />
    </parentNodes>
    <childNodes>
        <childNode name="dummy_test_proc_2" />
    </childNodes>
</node>
<node nodeId="test_team_org" name="team organization in testing"
nodeType="Ranked">
    <states>
        <state id="0" name="Extra Lower"/>
        <state id="1" name="Much Lower"/>
        <state id="2" name="Lower"/>
        <state id="3" name="The Same"/>
        <state id="4" name="Higher"/>
        <state id="5" name="Much Higher"/>
        <state id="6" name="Extra Higher"/>
    </states>
    <npt type="expression">
        <expression>TNormal(team_org,0.01,0.0,1.0)</expression>
    </npt>
    <parentNodes>
        <parentNode name="team_org" />
    </parentNodes>
    <childNodes>
        <childNode name="dummy_test_proc_2" />
    </childNodes>
</node>
<node nodeId="test_lead_q" name="leadership quality in testing"
nodeType="Ranked">
    <states>
        <state id="0" name="Extra Lower"/>
        <state id="1" name="Much Lower"/>
        <state id="2" name="Lower"/>
        <state id="3" name="The Same"/>
        <state id="4" name="Higher"/>
        <state id="5" name="Much Higher"/>
        <state id="6" name="Extra Higher"/>
    </states>
    <npt type="expression">
        <expression>TNormal(lead_q,0.01,0.0,1.0)</expression>
    </npt>
    <parentNodes>
        <parentNode name="lead_q" />
    </parentNodes>
    <childNodes>
        <childNode name="dummy_test_proc_2" />
    </childNodes>
</node>
<node nodeId="test_dist_comm" name="level of distributed communication in
testing" nodeType="Ranked">
    <states>
        <state id="0" name="Extra Lower"/>
        <state id="1" name="Much Lower"/>
        <state id="2" name="Lower"/>
        <state id="3" name="The Same"/>
        <state id="4" name="Higher"/>
        <state id="5" name="Much Higher"/>
        <state id="6" name="Extra Higher"/>
    </states>
    <npt type="expression">
        <expression>TNormal(dist_comm,0.01,0.0,1.0)</expression>
    </npt>
    <parentNodes>
        <parentNode name="dist_comm" />
    </parentNodes>

```

```

    <childNodes>
      <childNode name="dummy_test_proc_1" />
    </childNodes>
  </node>
  <node nodeId="test_app_meth_tool" name="appropriateness of methods and
tools used in testing" nodeType="Ranked">
    <states>
      <state id="0" name="Extra Lower"/>
      <state id="1" name="Much Lower"/>
      <state id="2" name="Lower"/>
      <state id="3" name="The Same"/>
      <state id="4" name="Higher"/>
      <state id="5" name="Much Higher"/>
      <state id="6" name="Extra Higher"/>
    </states>
    <npt type="expression">
      <expression>TNormal(app_meth_tool,0.01,0.0,1.0)</expression>
    </npt>
    <parentNodes>
      <parentNode name="app_meth_tool" />
    </parentNodes>
    <childNodes>
      <childNode name="dummy_test_proc_1" />
    </childNodes>
  </node>
  <node nodeId="dummy_cod_proc_2" name="dummy cod proc 2" nodeType="Ranked">
    <states>
      <state id="0" name="Extra Lower"/>
      <state id="1" name="Much Lower"/>
      <state id="2" name="Lower"/>
      <state id="3" name="The Same"/>
      <state id="4" name="Higher"/>
      <state id="5" name="Much Higher"/>
      <state id="6" name="Extra Higher"/>
    </states>
    <npt type="expression">
      <expression>TNormal(wmean(5.8,cod_def_proc,
7.2,cod_lead_q,
7.8,cod_team_org),0.001,0.0,1.0)</expression>
    </npt>
    <parentNodes>
      <parentNode name="cod_def_proc" />
      <parentNode name="cod_lead_q" />
      <parentNode name="cod_team_org" />
    </parentNodes>
    <childNodes>
      <childNode name="cod_ppq" />
    </childNodes>
  </node>
  <node nodeId="dummy_cod_proc_1" name="dummy cod proc 1" nodeType="Ranked">
    <states>
      <state id="0" name="Extra Lower"/>
      <state id="1" name="Much Lower"/>
      <state id="2" name="Lower"/>
      <state id="3" name="The Same"/>
      <state id="4" name="Higher"/>
      <state id="5" name="Much Higher"/>
      <state id="6" name="Extra Higher"/>
    </states>
    <npt type="expression">
      <expression>TNormal(wmean(7.4,cod_app_meth_tool,
6.8,cod_dist_comm),0.001,0.0,1.0)</expression>
    </npt>
    <parentNodes>
      <parentNode name="cod_app_meth_tool" />
      <parentNode name="cod_dist_comm" />
    </parentNodes>
    <childNodes>

```

```

        <childNodes name="cod_ppq" />
      </childNodes>
    </node>
    <node nodeId="dummy_test_proc_2" name="dummy test proc 2"
nodeType="Ranked">
      <states>
        <state id="0" name="Extra Lower"/>
        <state id="1" name="Much Lower"/>
        <state id="2" name="Lower"/>
        <state id="3" name="The Same"/>
        <state id="4" name="Higher"/>
        <state id="5" name="Much Higher"/>
        <state id="6" name="Extra Higher"/>
      </states>
      <npt type="expression">
        <expression>TNormal(wmean(5.8,test_def_proc,
7.2,test_lead_q,
7.8,test_team_org),0.001,0.0,1.0)</expression>
      </npt>
      <parentNodes>
        <parentNode name="test_lead_q" />
        <parentNode name="test_team_org" />
        <parentNode name="test_def_proc" />
      </parentNodes>
      <childNodes>
        <childNodes name="test_ppq" />
      </childNodes>
    </node>
    <node nodeId="dummy_test_proc_1" name="dummy test proc 1"
nodeType="Ranked">
      <states>
        <state id="0" name="Extra Lower"/>
        <state id="1" name="Much Lower"/>
        <state id="2" name="Lower"/>
        <state id="3" name="The Same"/>
        <state id="4" name="Higher"/>
        <state id="5" name="Much Higher"/>
        <state id="6" name="Extra Higher"/>
      </states>
      <npt type="expression">
        <expression>TNormal(wmean(7.4,test_app_meth_tool,
6.8,test_dist_comm),0.001,0.0,1.0)</expression>
      </npt>
      <parentNodes>
        <parentNode name="test_app_meth_tool" />
        <parentNode name="test_dist_comm" />
      </parentNodes>
      <childNodes>
        <childNodes name="test_ppq" />
      </childNodes>
    </node>
  </object>

```

B.2 Questionnaire about relationships between software project factors

Questionnaire for analyzing dependencies between variables describing a project, software quality and project group productivity

Researcher: Łukasz Radliński, PhD Student, lukrad@dcs.qmul.ac.uk

Supervisor: Professor Norman Fenton

Queen Mary College, University of London

Mile End Road, London E1 4NS, UK

The questionnaire is a part of my research on developing models for software project risk assessment. The main aim of this analysis is getting familiar with opinions of experienced software developers (analysts, programmers, testers, managers etc.) about the impact of various factors on:

- **software quality** – the quality of developed software measured as number of defects per unit of project size, e.g. defects / KLOC, defects / function point etc.,
- **project group productivity** – meant as a size of software developed for an unit of effort, e.g.: KLOC / person-month, function point / person-month, etc. It does **not** mean the software productivity in a sense of potential benefits which the end users may get by using the software. “Project group” means all the staff working at the project apart from administrative staff not assigned to a particular project, like: booking, personnel department etc. The “productivity” takes into account people, process factors and uncontrollable project factors.

All questions assume that a project is feasible – there is no such mixture of factors causing the project to be infeasible (not possible to be developed).

If you need to enter some comments/notes to particular question or to the questionnaire overall put them in a table in question 10. Here you can also add information about the other factors which were not listed in earlier questions and which you believe are important.

Time to fill out the questionnaire is approximately 30-35 minutes.

Person filling questionnaire

| | | | |
|--------------------------|--------------------------|------------------|--|
| Student | <input type="checkbox"/> | Year | |
| Academic | <input type="checkbox"/> | Years experience | |
| Industry | <input type="checkbox"/> | Years experience | |
| Owner / general director | <input type="checkbox"/> | Years experience | |

1. There are many factors involved in software development. Some of them are **uncontrollable** (problem complexity, project novelty etc.). Some of them are **controllable**, like process and people factors (team organisation, following a defined process, staff experience etc.). For 12 different combinations of uncontrollable and controllable factors how would you define an impact of a particular combination (scenario) on **software quality** and **project group productivity**?

Both uncontrollable and controllable factors vary between ‘badly against you’ and ‘very favourable’. E.g. uncontrollable project factors which are badly against you might be the most complex and novel problem you can imagine. Controllable project factors which are badly against you might be a team with very little experience.

Please use a scale from “0” to “10”, where “0” means very negative impact, “10” means very positive impact and “5” means neutral (no impact).

For example, referring to questions **a.** and **b.** below: Imagine that in a new project the uncontrollable factors are badly against you while the controllable factors are in your favour. Then if you believe that the overall impact on quality is neutral but the overall impact on productivity is really bad you would put something like “5” in question **a.** and something like “1” in question **b.** Please treat all examples in the questionnaire only as illustration how the values entered by you will be treated by us in the analysis of results. The examples are not suggestions on the exact values which you should enter.

- a.** How would you rate the impact on **software quality** when the **uncontrollable factors** are ‘badly against you’ (as bad as you have seen on your projects in the past) and the **controllable factors** are ‘very favourable’ (as good as you have seen on your projects in the past), e.g. a very complex and novel project combined with very experienced and motivated staff?

| | | | | | | | | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Other |
| <input type="checkbox"/> |
| Very negative impact | | | No Impact | | | | Very positive impact | | | | |

- b.** How would you rate the impact on **project group productivity** when the **uncontrollable factors** are ‘badly against you’ (as bad as you have seen on your projects in the past) and the **controllable factors** are ‘very favourable’ (as good as you have seen on your projects in the past)?

| | | | | | | | | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Other |
| <input type="checkbox"/> |
| Very negative impact | | | No Impact | | | | Very positive impact | | | | |

- c.** How would you rate the impact on **software quality** when the **uncontrollable factors** are ‘very favourable’ (as good as you have seen on your projects in the past) and the **controllable factors** are ‘badly against you’ (as bad as you have seen on your projects in the past)?

| | | | | | | | | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Other |
| <input type="checkbox"/> |
| Very negative impact | | | No Impact | | | | Very positive impact | | | | |

- d. How would you rate the impact on **project group productivity** when the **uncontrollable factors** are 'very favourable' (as good as you have seen on your projects in the past) and the **controllable factors** are 'badly against you' (as bad as you have seen on your projects in the past)?

| | | | | | | | | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|-------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Other |
| <input type="checkbox"/> | |
| Very negative impact | | | No Impact | | | | Very positive impact | | | | |

- e. How would you rate the impact on **software quality** when the **uncontrollable factors** are 'badly against you' (as bad as you have seen on your projects in the past) and the **controllable factors** are 'average'?

| | | | | | | | | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|-------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Other |
| <input type="checkbox"/> | |
| Very negative impact | | | No Impact | | | | Very positive impact | | | | |

- f. How would you rate the impact on **project group productivity** when the **uncontrollable factors** are 'badly against you' (as bad as you have seen on your projects in the past) and the **controllable factors** are 'average'?

| | | | | | | | | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|-------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Other |
| <input type="checkbox"/> | |
| Very negative impact | | | No Impact | | | | Very positive impact | | | | |

- g. How would you rate the impact on **software quality** when the **uncontrollable factors** are 'average' and the **controllable factors** are 'badly against you' (as bad as you have seen on your projects in the past)?

| | | | | | | | | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|-------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Other |
| <input type="checkbox"/> | |
| Very negative impact | | | No Impact | | | | Very positive impact | | | | |

- h. How would you rate the impact on **project group productivity** when the **uncontrollable factors** are 'average' and the **controllable factors** are 'badly against you'?

| | | | | | | | | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|-------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Other |
| <input type="checkbox"/> | |
| Very negative impact | | | No Impact | | | | Very positive impact | | | | |

- i. How would you rate the impact on **software quality** when the **uncontrollable factors** are 'average' and the **controllable factors** are 'very favourable' (as good as you have seen on your projects in the past)?

| | | | | | | | | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|-------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Other |
| <input type="checkbox"/> | |
| Very negative impact | | | No Impact | | | | Very positive impact | | | | |

- j. How would you rate the impact on **project group productivity** when the **uncontrollable factors** are 'average' and the **controllable factors** are 'very favourable' (as good as you have seen on your projects in the past)?

| | | | | | | | | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|-------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Other |
| <input type="checkbox"/> | |
| Very negative impact | | | No Impact | | | | Very positive impact | | | | |

- k. How would you rate the impact on **software quality** when the **uncontrollable factors** are 'very favourable' (as good as you have seen on your projects in the past) and the **controllable factors** are 'average'?

| | | | | | | | | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|-------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Other |
| <input type="checkbox"/> | |
| Very negative impact | | | No Impact | | | | Very positive impact | | | | |

- l. How would you rate the impact on **project group productivity** when the **uncontrollable factors** are 'very favourable' (as good as you have seen on your projects in the past) and the **controllable factors** are 'average'?

| | | | | | | | | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|-------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Other |
| <input type="checkbox"/> | |
| Very negative impact | | | No Impact | | | | Very positive impact | | | | |

2. This set of questions is intended to determine your opinion of the relevant impact at which a particular **controllable** factor affects **software quality**.

a. How would you rate the degree at which the **effectiveness of analysis and documentation process** influences **software quality**?

Please use the scale from “0” to “10”, where “0” means no impact at all, “1” means very small impact and “10” means very high impact. If you would like to express even higher impact (e.g. comparing to factors asked about in other questions) you can enter your own value in the last column. Use the same explanation of scale for questions 2-7.

If you enter value “3” as an answer to this question and value “1” as an answer to question 2b., it would mean that you believe that the influence of ‘effectiveness of analysis and documentation process’ is 3 times more important than ‘effectiveness of coding process’ in influencing software quality.

| | | | | | | | | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Other |
| <input type="checkbox"/> |
| No impact | | | | | | | | Very high impact | | | |

b. How would you rate the degree at which the **effectiveness of coding process** influences the **software quality**?

| | | | | | | | | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Other |
| <input type="checkbox"/> |
| No impact | | | | | | | | Very high impact | | | |

c. How would you rate the degree at which the **effectiveness of testing process** influences the **software quality**?

| | | | | | | | | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Other |
| <input type="checkbox"/> |
| No impact | | | | | | | | Very high impact | | | |

3. This set of questions is intended to determine your opinion of the relevant weights in which a particular **uncontrollable** project factor affects **software quality** and **project group productivity**.

a. How would you rate the degree at which the **project complexity** (how complex is the task/problem – this is not complexity of source code) influences **software quality**?

| | | | | | | | | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|-------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Other |
| <input type="checkbox"/> | |
| No impact | | | | | | | | Very high impact | | | |

b. How would you rate the degree at which the **project complexity** (how complex is the task/problem – this is not complexity of source code) influences **project group productivity**?

| | | | | | | | | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|-------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Other |
| <input type="checkbox"/> | |
| No impact | | | | | | | | Very high impact | | | |

c. How would you rate the degree at which the **project novelty** (how novel is the task for your team) influences **software quality**?

| | | | | | | | | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|-------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Other |
| <input type="checkbox"/> | |
| No impact | | | | | | | | Very high impact | | | |

d. How would you rate the degree at which the **project novelty** (how novel is the task for your team) influences **project group productivity**?

| | | | | | | | | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|-------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Other |
| <input type="checkbox"/> | |
| No impact | | | | | | | | Very high impact | | | |

e. How would you rate the degree at which the **project scale** (how big is the project – in bigger projects you may need more percentage of effort on other activities than software development) influences **software quality**?

| | | | | | | | | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|-------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Other |
| <input type="checkbox"/> | |
| No impact | | | | | | | | Very high impact | | | |

f. How would you rate the degree at which the **project scale** (how big is the project – in bigger projects you may need more percentage of effort on other activities than software development) influences **project group productivity**?

| | | | | | | | | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|-------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Other |
| <input type="checkbox"/> | |
| No impact | | | | | | | | Very high impact | | | |

- g. How would you rate the degree at which the **deadline pressure** (the extent to which there is a pressure on delivering software faster than you normally would like to deliver such software) influences **software quality**?

The **deadline pressure** can have both negative and positive effect. Negative – on **software quality** (with high deadline pressure the team has usually less time on improving software quality than they would like to have). Positive – on **project group productivity** (the team has to be more productive – deliver the similar functionality in shorter time).

| | | | | | | | | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|-------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Other |
| <input type="checkbox"/> | |
| No impact | | | | | | | | Very high impact | | | |

- h. How would you rate the degree at which the **deadline pressure** (the extent to which there is a pressure on delivering software faster than you normally would like to deliver such software) influences **project group productivity**?

| | | | | | | | | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|-------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Other |
| <input type="checkbox"/> | |
| No impact | | | | | | | | Very high impact | | | |

- i. How would you rate the degree at which the **quality of input documentation** (documentation reused from past projects, not the documentation produced in the current project during analysis and documentation process) influences **software quality**?

| | | | | | | | | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|-------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Other |
| <input type="checkbox"/> | |
| No impact | | | | | | | | Very high impact | | | |

- j. How would you rate the degree at which the **quality of input documentation** (documentation reused from past projects, not the documentation produced in the current project during analysis and documentation process) influences **project group productivity**?

| | | | | | | | | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|-------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Other |
| <input type="checkbox"/> | |
| No impact | | | | | | | | Very high impact | | | |

- k. How would you rate the degree at which the **positive customer involvement** (the extent to which the customers are involved in the development) influences **software quality**?

The **positive customer involvement** means that the customers are involved in the development process when necessary and this leads to both improvement of the software quality and project group productivity because the customers may relax the development group from some activities, like part of testing. The **negative customer involvement** means the excessive customer involvement in the development process where the customers disturb the development group because they want to participate in most of activities – thus the development team cannot use their resources productively. However, both positive and negative customer involvement lead to the increase of software quality. Use this explanation in questions **3k–3n**.

| | | | | | | | | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|-------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Other |
| <input type="checkbox"/> | |
| No impact | | | | | | | | Very high impact | | | |

- l.** How would you rate the degree at which the **positive customer involvement** (the extent to which the customers are involved in the development) influences **project group productivity**?

| | | | | | | | | | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|-------|--|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Other | |
| <input type="checkbox"/> | | |
| No impact | | | | | | | | Very high impact | | | | |

- m.** How would you rate the degree at which the **negative customer involvement** (the extent to which the customers are involved in the development) influences **software quality**?

| | | | | | | | | | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|-------|--|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Other | |
| <input type="checkbox"/> | | |
| No impact | | | | | | | | Very high impact | | | | |

- n.** How would you rate the degree at which the **negative customer involvement** (the extent to which the customers are involved in the development) influences **project group productivity**?

| | | | | | | | | | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|-------|--|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Other | |
| <input type="checkbox"/> | | |
| No impact | | | | | | | | Very high impact | | | | |

4. This set of questions is intended to determine your opinion of the relevant weights in which a particular factor affects **overall process and people quality** (the aggregated measure of all process and people qualitative factors listed below as sub-questions).

a. How would you rate the degree at which the **level of staff experience** influences the **overall process and people quality**?

| | | | | | | | | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|-------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Other |
| <input type="checkbox"/> | |
| No impact | | | | | | | | Very high impact | | | |

b. How would you rate the degree at which the **level of staff motivation** influences the **overall process and people quality**?

| | | | | | | | | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|-------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Other |
| <input type="checkbox"/> | |
| No impact | | | | | | | | Very high impact | | | |

c. How would you rate the degree at which the **level of staff education** influences the **overall process and people quality**?

| | | | | | | | | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|-------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Other |
| <input type="checkbox"/> | |
| No impact | | | | | | | | Very high impact | | | |

d. How would you rate the degree at which the **team organisation** influences the **overall process and people quality**?

| | | | | | | | | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|-------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Other |
| <input type="checkbox"/> | |
| No impact | | | | | | | | Very high impact | | | |

e. How would you rate the degree at which the **appropriateness of methods and tools used** influences the **overall process and people quality**?

If the development methods (e.g. requirements elicitation, coding and testing methods, project lifecycle) and tools (CASE tools, development environments, testing tools) are selected properly the developers may work more efficiently and deliver software with fewer defects.

| | | | | | | | | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|-------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Other |
| <input type="checkbox"/> | |
| No impact | | | | | | | | Very high impact | | | |

- f. How would you rate the degree at which the **level of distributed communications** influences the **overall process and people quality**?

If the employees need to spend much time on communicating with other employees they may either have less time on delivering new artefacts (documentation, code, test cases etc.) or produce the same functionality faster but with poorer quality.

| | | | | | | | | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|-------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Other |
| <input type="checkbox"/> | |
| No impact | | | | | | | | Very high impact | | | |

- g. How would you rate the degree at which the **well defined process followed** influences the **overall process and people quality**?

If there is a set of development processes and procedures established the employees may know how to cope with certain situations and thus not loose time on finding what to do when the problem appears.

| | | | | | | | | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|-------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Other |
| <input type="checkbox"/> | |
| No impact | | | | | | | | Very high impact | | | |

- h. How would you rate the degree at which the **leadership quality** (both higher and lower level) influences the **overall process and people quality**?

Strong leadership with vision on the whole project and its parts may cause the improvements in managing employees and resources (time, equipment etc.) and increase the employees' productivity and delivered software quality.

| | | | | | | | | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|-------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Other |
| <input type="checkbox"/> | |
| No impact | | | | | | | | Very high impact | | | |

10. NotesA large, empty rectangular box with a black border, intended for the user to write notes. It occupies the majority of the page below the '10. Notes' header.

B.3 Raw results from questionnaire survey

| Person filling | No Type Years | 1 I 4 | 2 I 12 | 3 A 30 | 4 A 2 | 5 I 18 | 6 S I O 4 20 5 | 7 I 20 | 8 A 20 | 9 O 20 |
|--|---------------------|-------------|--------------|--------------|-------------|--------------|----------------------|--------------|--------------|--------------|
| | Weight | 5 | 3 | 3 | 1 | 3 | 4 | 3 | 3 | 5 |
| 1. Impact of uncontrollable (U) and controllable (C) on quality (Q) and productivity (P) | a-Q-UB-CF | 4 | 3 | 5 | 4 | 3 | 5 | 6 | 5 | 2 |
| | b-P-UB-CF | 4 | 9 | 3 | 4 | 2 | 5 | 6 | 3 | 2 |
| | c-Q-UF-CB | 2 | 2 | 5 | 5 | 2 | 2 | 2 | 2 | 4 |
| | d-P-UF-CB | 4 | 2 | 4 | 5 | 2 | 3 | 3 | 2 | 4 |
| | e-Q-UB-CA | 1 | 3 | 5 | 7 | 3 | 1 | 3 | 3 | 1 |
| | f-P-UB-CA | 3 | 5 | 5 | 5 | 3 | 3 | 3 | 3 | 1 |
| | g-Q-UA-CB | 1 | 2 | 5 | | 3 | 1 | 2 | 4 | 2 |
| | h-P-UA-CB | 3 | 1 | 5 | 6 | 3 | 1 | 1 | 3 | 2 |
| | i-Q-UA-CF | 7 | 10 | 5 | 7 | 6 | 9 | 8 | 8 | 8 |
| | j-P-UA-CF | 9 | 10 | 5 | 6 | 6 | 9 | 8 | 7 | 9 |
| | k-Q-UF-CA | 7 | 5 | 5 | 6 | 6 | 7 | 5 | 7 | 8 |
| l-Q-UF-CA | 9 | 7 | 5 | 7 | 6 | 6 | 5 | 7 | 7 | |
| 2. Impact of controllable on quality | a-eff_analysis | 9 | 0 | 8 | 9 | 8 | 8 | 9 | 7 | 7 |
| | b-eff_coding | 8 | 0 | 8 | 8 | 8 | 6 | 7 | 3 | 9 |
| | c-eff_testing | 10 | 0 | 10 | 10 | 9 | 10 | 6 | 8 | 10 |
| 3. Uncontrollable on quality and productivity | a-Q-compl | 7 | 2 | 9 | 7 | 9 | 7 | 7 | 10 | 8 |
| | b-P-compl | 5 | 2 | 9 | 9 | 5 | 8 | 6 | 10 | 10 |
| | c-Q-novel | 9 | 0 | 9 | 9 | 7 | 7 | 6 | 10 | 5 |
| | d-P-novel | 7 | 7 | 8 | 7 | 9 | 7 | 6 | 10 | 10 |
| | e-Q-scale | 3 | 0 | 10 | 9 | 5 | 2 | 8 | 8 | 7 |
| | f-P-scale | 3 | 0 | 10 | 3 | 7 | 3 | 7 | 8 | 10 |
| | g-Q-dead | 7 | 8 | 9 | 8 | 9 | 7 | 10 | 9 | 10 |
| | h-P-dead | 7 | 6 | 9 | 10 | 6 | 2 | 9 | 9 | 8 |
| | i-Q-qinput | 7 | 10 | 9 | 3 | 3 | 5 | 5 | 5 | 7 |
| | j-P-qinput | 7 | 10 | 9 | 1 | 2 | 5 | 4 | 4 | 8 |
| | k-Q-posinv | 8 | 7 | 10 | 7 | 5 | 3 | 9 | 10 | 7 |
| | l-P-posinv | 3 | 7 | 9 | 9 | 7 | 2 | 9 | 8 | 2 |
| | m-Q-neginv | 9 | 3 | 9 | 4 | 1 | 3 | 3 | 10 | 5 |
| | n-P-neginv | 4 | 3 | 9 | 9 | 3 | 2 | 2 | 10 | 5 |
| 4. PPQ factors | a-exper | 6 | 9 | 9 | 8 | 8 | 8 | 9 | 10 | 9 |
| | b-motiv | 8 | 8 | 9 | 9 | 9 | 9 | 9 | 10 | 8 |
| | c-educ | 5 | 6 | 9 | 4 | 8 | 5 | 9 | 5 | 8 |
| | d-teamorg | 8 | 8 | 9 | 8 | 9 | 7 | 9 | 3 | 9 |
| | e-appmeth | 6 | 8 | 9 | 9 | 10 | 7 | 9 | 6 | 6 |
| | f-distcomm | 8 | 9 | 5 | 9 | 7 | 3 | 5 | 8 | 8 |
| | g-defproc | 7 | 6 | 9 | 10 | 7 | 3 | 6 | 4 | 4 |
| | h-leadq | 7 | 9 | 7 | 10 | 7 | 7 | 5 | 5 | 9 |

| Person filling | No | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|--|-----------|------|-----|-----|-----|-----|------|------|-----|-----|
| 5. Effectiveness of analysis and doc | a-reqq | 9 | 8 | 8 | 6 | 10 | 7 | 9 | 9 | 9 |
| | b-ppq | 4 | 9 | 8 | 5 | 9 | 6 | 8 | 5 | 8 |
| | c-effort | 8 | 10 | 8 | 6 | 9 | 6 | 8 | 5 | 9 |
| 6. Effectiveness of coding | a-docq | 6 | 10 | 8 | 9 | 8 | 2 | 8 | 5 | 7 |
| | b-ppq | 8 | 10 | 8 | 6 | 8 | 7 | 8 | 5 | 9 |
| | c-effort | 4 | 7 | 8 | 7 | 7 | 8 | 8 | 5 | 8 |
| 7. Effectiveness of testing | a-docq | 9 | 10 | 8 | 6 | 9 | 7 | 8 | 5 | 9 |
| | b-ppq | 7 | 10 | 8 | 2 | 9 | 9 | 8 | 5 | 10 |
| | c-effort | 10 | 7 | 8 | 2 | 9 | 9 | 8 | 5 | 10 |
| 8. Code reuse | a-typical | 0.8 | 0.7 | 0.4 | 0.5 | 0.5 | 0.5 | 0.3 | 0.7 | 0.2 |
| | b-lowest | 0.5 | 0 | 0 | 0.3 | 0.3 | 0 | 0 | 0.2 | 0 |
| | c-highest | 0.99 | 0.8 | 0.3 | 0.7 | 0.6 | 0.9 | 0.4 | 0.9 | 0.8 |
| 9. Overall on quality and productivity | a-worst-Q | 0.01 | 0.3 | 0.5 | 0.2 | 0.3 | 0.05 | 0.03 | | 0.1 |
| | b-worst-P | 0.1 | 0.2 | 0.7 | 0.3 | 0.2 | 0.1 | 0.03 | 0.1 | 0.3 |
| | c-best-Q | 10 | 2 | 2.5 | | 6 | 10 | 4 | 10 | 3 |
| | d-best-P | 2 | 2 | 2.5 | 1.5 | 4 | 10 | 4 | 20 | 4 |

Appendix C Productivity and Defect Rates Model

C.1 Statistical analysis of productivity and defect rates

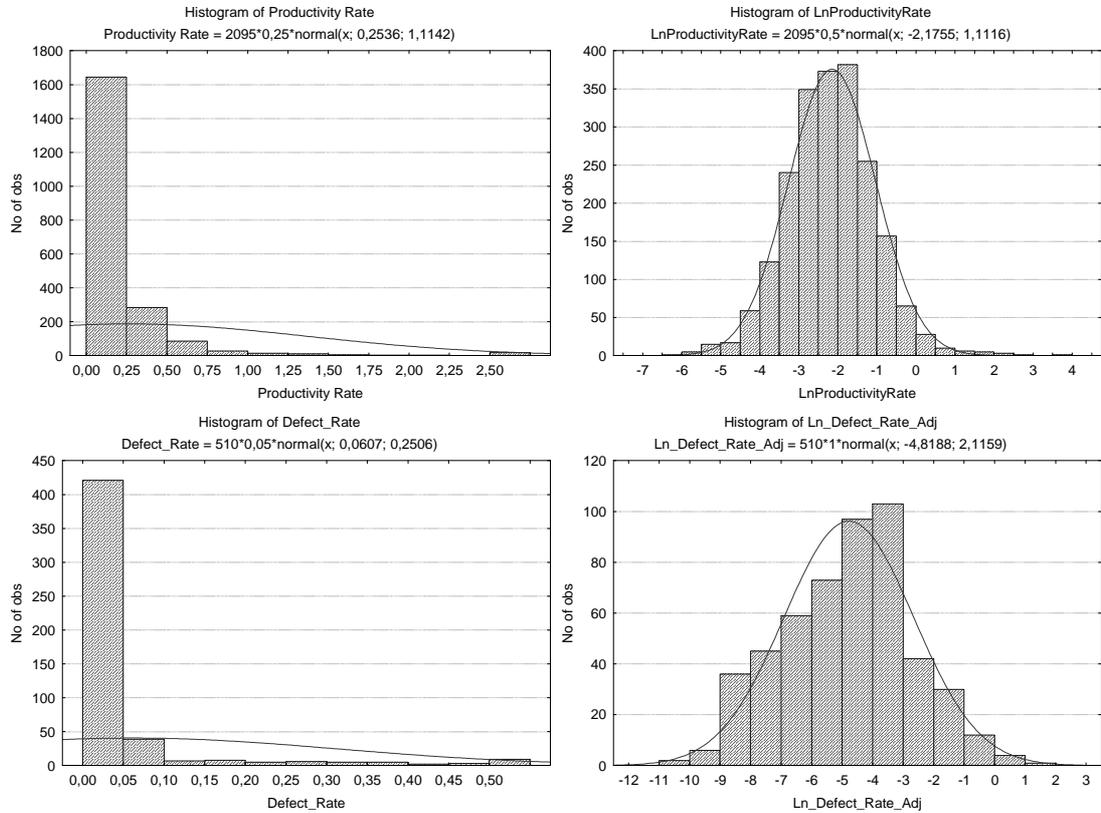


Figure C-1 Histograms for productivity and defect rates with and without Ln transformations

Table C-1 Spearman’s rank correlation coefficients (SRCC) between dependent variables and potential numeric predictors

| Predictor \ Dependent | Productivity Rate | Defect Rate |
|---|-------------------|--------------|
| Average Team Size | -0.42 | 0.01 |
| Project Elapsed Time | -0.21 | 0.11 |
| Functional Size | 0.30 | 0.08 |
| Summary Work Effort | -0.50 | 0.09 |
| Value Adjustment Factor | -0.16 | 0.00 |
| User Base – Business Units | -0.06 | 0.10 |
| User Base – Concurrent Users | -0.14 | -0.05 |
| User Base – Locations | -0.17 | -0.23 |
| values marked in bold are statistically significant at $p < 0.05$ | | |

Table C-2 Summary of KW-ANOVA H test statistic values

| Predictor \ Dependent | Productivity Rate | Defect Rate |
|------------------------------|----------------------|---------------------|
| Package Customisation | 2.88 (0.09) | 3.08 (0.08) |
| Development Platform | 104.91 (0.00) | 33.03 (0.00) |
| CASE Tool Used | 2.29 (0.13) | 11.02 (0.00) |
| Used Methodology | 45.50 (0.00) | 12.43 (0.00) |
| Development Type | 35.76 (0.00) | 4.61 (0.10) |
| Organisation Type | 245.62 (0.00) | 44.77 (0.00) |
| Business Area Type | 252.96 (0.00) | 48.95 (0.00) |
| Application Type | 131.63 (0.00) | 29.39 (0.00) |
| Architecture | 44.33 (0.00) | 33.52 (0.00) |
| Client-Server | 0.01 (0.94) | 10.86 (0.00) |
| Language Type | 51.81 (0.00) | 0.98 (0.61) |
| Primary Programming Language | 169.85 (0.00) | 38.79 (0.00) |
| Intended Market | 11.23 (0.01) | 22.40 (0.00) |

values marked in bold are statistically significant at p<0.05
p values in brackets

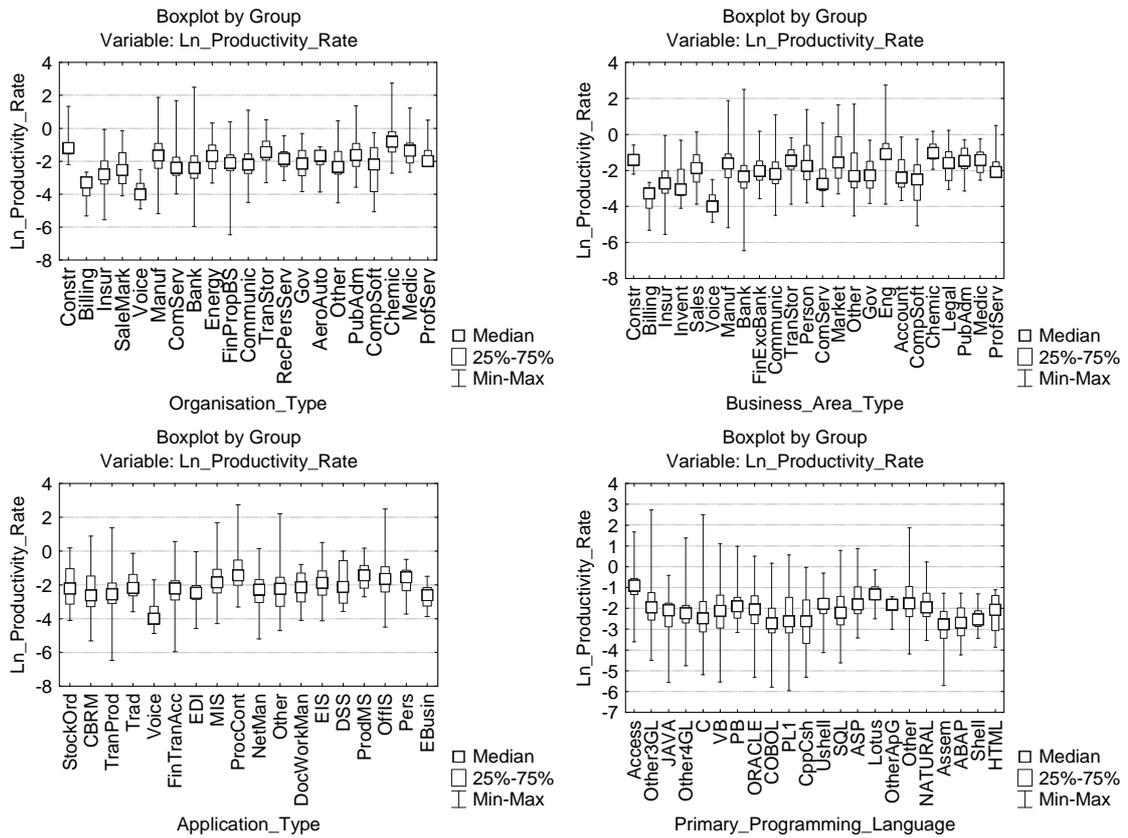


Figure C-2 Box-plots of 4 predictors with high number of states (for productivity rate)

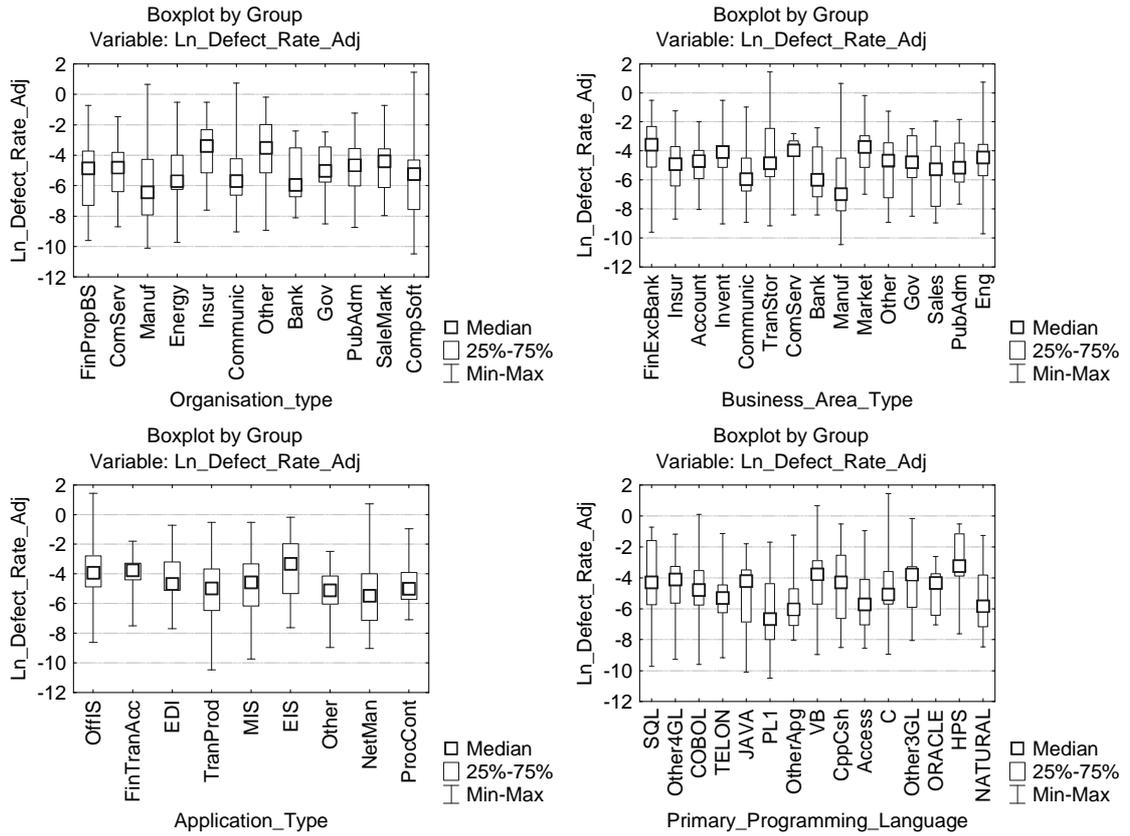


Figure C-3 Box-plots of 4 predictors with high number of states (for defect rate)

Table C-3 SRCCs between numeric predictor variables (productivity rate analysis)

| | Value Adjustment Factor | User Base – Concurrent Users | User Base – Locations |
|------------------------------|-------------------------|------------------------------|-----------------------|
| Value Adjustment Factor | - | 0.14 | 0.07 |
| User Base – Concurrent Users | 0.14 | - | 0.32 |
| User Base – Locations | 0.07 | 0.32 | - |

values marked in bold are statistically significant at $p < 0.05$

Table C-4 Associations between categorical predictors (*productivity rate analysis*)

| | Development Platform | Used Methodology | Development Type | Architecture | Language Type | Intended Market |
|---|---|----------------------|----------------------|---|----------------------|---|
| Development Platform | - | 0.23 0.23 0.23 | 0.23 0.16 0.23 | 1.06 0.61 0.73 | 0.25 0.15 0.25 | 0.71 0.41 0.58 |
| Used Methodology | 0.23 0.23 0.23 | - | 0.10 0.10 0.09 | 0.22 0.22 0.21 | 0.10 0.10 0.10 | 0.28 0.28 0.27 |
| Development Type | 0.23 0.16 0.23 | 0.10 0.10 0.09 | - | 0.22 0.15 0.21 | 0.10 0.07 0.10 | 0.32 0.23 0.30 |
| Architecture | 1.06 0.61 0.73 | 0.22 0.22 0.21 | 0.22 0.15 0.21 | - | 0.32 0.18 0.30 | 0.42 0.24 0.39 |
| Language Type | 0.25 0.15 0.25 | 0.10 0.10 0.10 | 0.10 0.07 0.10 | 0.32 0.18 0.30 | - | 0.29 0.17 0.28 |
| Intended Market | 0.71 0.41 0.58 | 0.28 0.28 0.27 | 0.32 0.23 0.30 | 0.42 0.24 0.39 | 0.29 0.17 0.28 | - |
| values in each cell are: Phi (top), Cramer's V (middle) and contingency coefficient (bottom) values above 0.5 marked in bold | | | | | | |

Table C-5 Summary of KW-ANOVA H test statistic values between numeric and categorical predictors (*productivity rate analysis*)

| | Value Adjustment Factor | User Base – Locations |
|---|-------------------------|-----------------------|
| Development Platform | 50.04 (0.00) | 1.33 (0.72) |
| Used Methodology | 4.44 (0.04) | 1.80 (0.18) |
| Development Type | 113.39 (0.00) | 12.08 (0.00) |
| Language Type | 18.56 (0.00) | 0.00 (1.00) |
| values marked in bold are statistically significant at p<0.05 p values in brackets | | |

Table C-6 SRCCs between numeric predictor variables (*defect rate analysis*)

| | Project Elapsed Time | Summary Work Effort | User Base – Locations |
|---|----------------------|---------------------|-----------------------|
| Project Elapsed Time | - | 0.74 | 0.00 |
| Summary Work Effort | 0.74 | - | 0.03 |
| User Base – Locations | 0.00 | 0.03 | - |
| values marked in bold are statistically significant at p<0.05 | | | |

Table C-7 Associations between categorical predictors (*defect rate analysis*)

| | Development Platform | CASE Tool Used | Used Methodology | Architecture | Client-Server | Intended Market |
|---|----------------------|----------------|------------------|--------------|---------------|-----------------|
| Development Platform | - | 0.11 | 0.25 | 1.04 | 0.50 | 0.35 |
| | | 0.11 | 0.25 | 0.60 | 0.50 | 0.20 |
| | | 0.11 | 0.25 | 0.72 | 0.45 | 0.33 |
| CASE Tool Used | 0.11 | - | -0.36 | 0.22 | -0.14 | 0.43 |
| | 0.11 | | 0.36 | 0.22 | 0.14 | 0.43 |
| | 0.11 | | 0.34 | 0.22 | 0.14 | 0.39 |
| Used Methodology | 0.25 | -0.36 | - | 0.22 | -0.01 | 0.44 |
| | 0.25 | 0.36 | | 0.22 | 0.01 | 0.44 |
| | 0.25 | 0.34 | | 0.21 | 0.01 | 0.40 |
| Architecture | 1.04 | 0.22 | 0.22 | | 1.00 | 0.39 |
| | 0.60 | 0.22 | 0.22 | - | 1.00 | 0.22 |
| | 0.72 | 0.22 | 0.21 | | 0.71 | 0.36 |
| Client-Server | 0.50 | -0.14 | -0.01 | 1.00 | | 0.25 |
| | 0.50 | 0.14 | 0.01 | 1.00 | - | 0.25 |
| | 0.45 | 0.14 | 0.01 | 0.71 | | 0.24 |
| Intended Market | 0.35 | 0.43 | 0.44 | 0.39 | 0.25 | |
| | 0.20 | 0.43 | 0.44 | 0.22 | 0.25 | - |
| | 0.33 | 0.39 | 0.40 | 0.36 | 0.24 | |
| values in each cell are: Phi (top), Cramer's V (middle) and contingency coefficient (bottom) values above 0.5 marked in bold | | | | | | |

Table C-8 Summary of KW-ANOVA H test statistic values between numeric and categorical predictors (*defect rate analysis*)

| | Summary Work Effort | User Base – Locations |
|---|---------------------|-----------------------|
| Development Platform | 36.01 (0.00) | 2.47 (0.48) |
| CASE Tool Used | 0.00 (0.98) | 11.90 (0.00) |
| Used Methodology | 3.00 (0.08) | 0.01 (0.91) |
| values marked in bold are statistically significant at $p < 0.05$ p values in brackets | | |

C.2 Definition of PDR model

```

<?xml version="1.0" encoding="iso-8859-2" ?>
<object name="Priors" type="structure">
  <node nodeId="LnP" name="Ln Productivity Rate" nodeType="Continuous
Interval">
    <states>
      <state id="0" lowerBound="-7.5" upperBound="-6.5"/>
      <state id="1" lowerBound="-6.5" upperBound="-5.5"/>
      <state id="2" lowerBound="-5.5" upperBound="-4.5"/>
      <state id="3" lowerBound="-4.5" upperBound="-3.5"/>
      <state id="4" lowerBound="-3.5" upperBound="-2.5"/>
      <state id="5" lowerBound="-2.5" upperBound="-1.5"/>
      <state id="6" lowerBound="-1.5" upperBound="-0.5"/>
      <state id="7" lowerBound="-0.5" upperBound="0.5"/>
      <state id="8" lowerBound="0.5" upperBound="1.5"/>
      <state id="9" lowerBound="1.5" upperBound="2.5"/>
      <state id="10" lowerBound="2.5" upperBound="3.5"/>
      <state id="11" lowerBound="3.5" upperBound="4.5"/>
    </states>
    <npt type="expression">
      <expression>Normal(-2.18,1.11 ^ 2)</expression>
    </npt>
    <parentNodes>
    </parentNodes>
    <childNodes>
      <childNodes name="locations_P" />
      <childNodes name="P" />
      <childNodes name="lang_type" />
      <childNodes name="dev_platform_p" />
      <childNodes name="dev_type" />
      <childNodes name="meth_P" />
    </childNodes>
  </node>
  <node nodeId="locations_P" name="User Base - Locations"
nodeType="Labelled">
    <states>
      <state id="0" name="1"/>
      <state id="1" name="2-5"/>
      <state id="2" name=">5"/>
    </states>
    <npt type="manual">
      <value>2.29336E-4</value>
      <value>0.001026981</value>
      <value>0.00458516</value>
      <value>0.020161577</value>
      <value>0.082478054</value>
      <value>0.26665395</value>
      <value>0.5735129</value>
      <value>0.83955175</value>
      <value>0.9608183</value>
      <value>0.99250984</value>
      <value>0.9985681</value>
      <value>0.9996989</value>
      <value>2.05169E-7</value>
      <value>9.27917E-6</value>
      <value>2.38406E-4</value>
      <value>0.00343725</value>
      <value>0.026269943</value>
      <value>0.09040909</value>
      <value>0.11793971</value>
      <value>0.059665933</value>
      <value>0.013445957</value>
      <value>0.001558354</value>
      <value>1.0023E-4</value>
    </npt>
  </node>
</object>

```

```

    <value>3.65502E-6</value>
    <value>0.99977046</value>
    <value>0.9989637</value>
    <value>0.99517643</value>
    <value>0.97640115</value>
    <value>0.891252</value>
    <value>0.64293694</value>
    <value>0.30854738</value>
    <value>0.10078233</value>
    <value>0.02573573</value>
    <value>0.005931825</value>
    <value>0.001331648</value>
    <value>2.97467E-4</value>
  </npt>
  <parentNodes>
    <parentNode name="LnP" />
  </parentNodes>
  <childNodes>
  </childNodes>
</node>
<node nodeId="LnD" name="Ln Defect Rate" nodeType="Continuous Interval">
  <states>
    <state id="0" lowerBound="-9.5" upperBound="-8.5"/>
    <state id="1" lowerBound="-8.5" upperBound="-7.5"/>
    <state id="2" lowerBound="-7.5" upperBound="-6.5"/>
    <state id="3" lowerBound="-6.5" upperBound="-5.5"/>
    <state id="4" lowerBound="-5.5" upperBound="-4.5"/>
    <state id="5" lowerBound="-4.5" upperBound="-3.5"/>
    <state id="6" lowerBound="-3.5" upperBound="-2.5"/>
    <state id="7" lowerBound="-2.5" upperBound="-1.5"/>
    <state id="8" lowerBound="-1.5" upperBound="-0.5"/>
    <state id="9" lowerBound="-0.5" upperBound="0.5"/>
    <state id="10" lowerBound="0.5" upperBound="1.5"/>
    <state id="11" lowerBound="1.5" upperBound="2.5"/>
    <state id="12" lowerBound="2.5" upperBound="3.5"/>
  </states>
  <npt type="expression">
    <expression>Normal(-3,2 ^ 2)</expression>
  </npt>
  <parentNodes>
  </parentNodes>
  <childNodes>
    <childNode name="func_size" />
    <childNode name="D" />
    <childNode name="dev_platform_d" />
    <childNode name="case" />
    <childNode name="meth_d" />
  </childNodes>
</node>
<node nodeId="func_size" name="Functional Size" nodeType="Integer Interval">
  <states>
    <state id="0" lowerBound="2.0" upperBound="5.0"/>
    <state id="1" lowerBound="5.0" upperBound="12.0"/>
    <state id="2" lowerBound="12.0" upperBound="33.0"/>
    <state id="3" lowerBound="33.0" upperBound="90.0"/>
    <state id="4" lowerBound="90.0" upperBound="245.0"/>
    <state id="5" lowerBound="245.0" upperBound="665.0"/>
    <state id="6" lowerBound="665.0" upperBound="1808.0"/>
    <state id="7" lowerBound="1808.0" upperBound="4915.0"/>
    <state id="8" lowerBound="4915.0" upperBound="13360.0"/>
    <state id="9" lowerBound="13360.0" upperBound="36317.0"/>
  </states>
  <npt type="manual">
    <value>0.03686562</value>
    <value>0.030468224</value>
    <value>0.024761898</value>
    <value>0.019776655</value>
  </npt>

```

<value>0.015512417</value>
<value>0.011942427</value>
<value>0.00901824</value>
<value>0.006675659</value>
<value>0.004840956</value>
<value>0.003436725</value>
<value>0.002386906</value>
<value>0.001620644</value>
<value>0.001074907</value>
<value>0.05222951</value>
<value>0.04597412</value>
<value>0.039944805</value>
<value>0.034235556</value>
<value>0.02892613</value>
<value>0.02407842</value>
<value>0.019734211</value>
<value>0.015914498</value>
<value>0.012620275</value>
<value>0.009834666</value>
<value>0.007526018</value>
<value>0.005651595</value>
<value>0.004161459</value>
<value>0.09832412</value>
<value>0.09050046</value>
<value>0.08252238</value>
<value>0.07449786</value>
<value>0.066541515</value>
<value>0.058768835</value>
<value>0.05129046</value>
<value>0.04420672</value>
<value>0.0376031</value>
<value>0.03154676</value>
<value>0.026084451</value>
<value>0.021241715</value>
<value>0.017023435</value>
<value>0.21635911</value>
<value>0.2090494</value>
<value>0.20057482</value>
<value>0.19097583</value>
<value>0.18033522</value>
<value>0.16877642</value>
<value>0.15645933</value>
<value>0.14357434</value>
<value>0.13033444</value>
<value>0.11696632</value>
<value>0.1037007</value>
<value>0.0907626</value>
<value>0.078361966</value>
<value>0.30252358</value>
<value>0.30635956</value>
<value>0.3087857</value>
<value>0.30956897</value>
<value>0.3085018</value>
<value>0.3054126</value>
<value>0.300175</value>
<value>0.29271686</value>
<value>0.283027</value>
<value>0.27116093</value>
<value>0.25724417</value>
<value>0.2414728</value>
<value>0.2241114</value>
<value>0.18611777</value>
<value>0.1972412</value>
<value>0.2085154</value>
<value>0.21975066</value>
<value>0.23072752</value>
<value>0.24119851</value>
<value>0.25089115</value>

```

<value>0.25951275</value>
<value>0.2667572</value>
<value>0.27231386</value>
<value>0.27587962</value>
<value>0.27717295</value>
<value>0.27595118</value>
<value>0.07869465</value>
<value>0.08714747</value>
<value>0.09648282</value>
<value>0.10672145</value>
<value>0.117865205</value>
<value>0.12989143</value>
<value>0.14274661</value>
<value>0.15633951</value>
<value>0.17053388</value>
<value>0.18514153</value>
<value>0.19991629</value>
<value>0.21455018</value>
<value>0.2286729</value>
<value>0.020265574</value>
<value>0.023089493</value>
<value>0.026356582</value>
<value>0.030123385</value>
<value>0.034449596</value>
<value>0.039396618</value>
<value>0.04502542</value>
<value>0.05139344</value>
<value>0.05855044</value>
<value>0.06653304</value>
<value>0.07535797</value>
<value>0.0850141</value>
<value>0.09545338</value>
<value>0.00623364</value>
<value>0.007300135</value>
<value>0.008583253</value>
<value>0.010125714</value>
<value>0.011977819</value>
<value>0.014198321</value>
<value>0.016855162</value>
<value>0.020025928</value>
<value>0.023797842</value>
<value>0.028267015</value>
<value>0.03353665</value>
<value>0.03971385</value>
<value>0.046904612</value>
<value>0.002386427</value>
<value>0.00286994</value>
<value>0.003472337</value>
<value>0.004223925</value>
<value>0.005162752</value>
<value>0.006336451</value>
<value>0.007804408</value>
<value>0.009640292</value>
<value>0.01193488</value>
<value>0.014799119</value>
<value>0.018367223</value>
<value>0.022799546</value>
<value>0.028284771</value>
</npt>
<parentNodes>
  <parentNode name="LnD" />
</parentNodes>
<childNodes>
</childNodes>
</node>
<node nodeId="P" name="Productivity Rate" nodeType="Continuous Interval">
  <states>
    <state id="0" lowerBound="0.0" upperBound="10.0"/>

```

```

        <state id="1" lowerBound="10.0" upperBound="Infinity"/>
    </states>
    <npt type="expression">
        <expression>Arithmetic(e ^ LnP)</expression>
    </npt>
    <parentNodes>
        <parentNode name="LnP" />
    </parentNodes>
    <childNodes>
    </childNodes>
</node>
<node nodeId="D" name="Defect Rate" nodeType="Continuous Interval">
    <states>
        <state id="0" lowerBound="0.0" upperBound="10.0"/>
        <state id="1" lowerBound="10.0" upperBound="Infinity"/>
    </states>
    <npt type="expression">
        <expression>Arithmetic(e ^ LnD)</expression>
    </npt>
    <parentNodes>
        <parentNode name="LnD" />
    </parentNodes>
    <childNodes>
    </childNodes>
</node>
<node nodeId="lang_type" name="Language Type" nodeType="Labelled">
    <states>
        <state id="0" name="2GL"/>
        <state id="1" name="3GL"/>
        <state id="2" name="4GL"/>
        <state id="3" name="ApG"/>
    </states>
    <npt type="manual">
        <value>0.024031976</value>
        <value>0.028511107</value>
        <value>0.027814467</value>
        <value>0.02194764</value>
        <value>0.013852079</value>
        <value>0.006890452</value>
        <value>0.002857604</value>
        <value>0.00117744</value>
        <value>4.64577E-4</value>
        <value>1.57687E-4</value>
        <value>4.634E-5</value>
        <value>1.15672E-5</value>
        <value>0.96795887</value>
        <value>0.94566005</value>
        <value>0.9036884</value>
        <value>0.83087593</value>
        <value>0.72683716</value>
        <value>0.5960973</value>
        <value>0.484833</value>
        <value>0.4660397</value>
        <value>0.5102811</value>
        <value>0.571728</value>
        <value>0.6597255</value>
        <value>0.7691727</value>
        <value>0.008009134</value>
        <value>0.025828825</value>
        <value>0.06849451</value>
        <value>0.14691542</value>
        <value>0.25205132</value>
        <value>0.34081307</value>
        <value>0.38420683</value>
        <value>0.43032464</value>
        <value>0.46154043</value>
        <value>0.4258369</value>
        <value>0.34017095</value>
    </npt>
</node>

```

```

    <value>0.23081528</value>
    <value>6.48028E-12</value>
    <value>7.69123E-9</value>
    <value>2.65099E-6</value>
    <value>2.61013E-4</value>
    <value>0.007259444</value>
    <value>0.056199208</value>
    <value>0.12810256</value>
    <value>0.102458246</value>
    <value>0.0277139</value>
    <value>0.002277443</value>
    <value>5.72265E-5</value>
    <value>4.31359E-7</value>
  </npt>
  <parentNodes>
    <parentNode name="LnP" />
  </parentNodes>
  <childNodes>
  </childNodes>
</node>
  <node nodeId="dev_platform_p" name="Development Platform"
nodeType="Labelled">
  <states>
    <state id="0" name="PC"/>
    <state id="1" name="Mid-Range"/>
    <state id="2" name="Mainframe"/>
    <state id="3" name="Multi"/>
  </states>
  <npt type="manual">
    <value>0.02321982</value>
    <value>0.04085116</value>
    <value>0.07543211</value>
    <value>0.1373862</value>
    <value>0.23453511</value>
    <value>0.3733123</value>
    <value>0.5566432</value>
    <value>0.75264436</value>
    <value>0.89640284</value>
    <value>0.9660743</value>
    <value>0.9905875</value>
    <value>0.99766254</value>
    <value>0.062192958</value>
    <value>0.1107442</value>
    <value>0.18137585</value>
    <value>0.25677058</value>
    <value>0.29858035</value>
    <value>0.28369343</value>
    <value>0.22128388</value>
    <value>0.13716109</value>
    <value>0.06562736</value>
    <value>0.024900332</value>
    <value>0.007877215</value>
    <value>0.002144969</value>
    <value>0.9125009</value>
    <value>0.83883446</value>
    <value>0.7092476</value>
    <value>0.5183544</value>
    <value>0.3111759</value>
    <value>0.15263607</value>
    <value>0.061464</value>
    <value>0.019668223</value>
    <value>0.004858277</value>
    <value>9.51625E-4</value>
    <value>1.55416E-4</value>
    <value>2.18478E-5</value>
    <value>0.00208629</value>
    <value>0.009570159</value>
    <value>0.03394445</value>
  </npt>

```

```

        <value>0.08748878</value>
        <value>0.15570863</value>
        <value>0.19035822</value>
        <value>0.16060892</value>
        <value>0.09052632</value>
        <value>0.03311155</value>
        <value>0.008073781</value>
        <value>0.001379902</value>
        <value>1.70658E-4</value>
    </npt>
    <parentNodes>
        <parentNode name="LnP" />
    </parentNodes>
    <childNodes>
    </childNodes>
</node>
<node nodeId="dev_type" name="Development Type" nodeType="Labelled">
    <states>
        <state id="0" name="New"/>
        <state id="1" name="Enhancement"/>
        <state id="2" name="Re-development"/>
    </states>
    <npt type="manual">
        <value>0.08010148</value>
        <value>0.11583285</value>
        <value>0.16462816</value>
        <value>0.2289541</value>
        <value>0.30977622</value>
        <value>0.40493855</value>
        <value>0.508282</value>
        <value>0.6109212</value>
        <value>0.70433104</value>
        <value>0.78301823</value>
        <value>0.8452166</value>
        <value>0.89198476</value>
        <value>0.9136128</value>
        <value>0.8739626</value>
        <value>0.82168376</value>
        <value>0.755942</value>
        <value>0.6765931</value>
        <value>0.5850704</value>
        <value>0.48580632</value>
        <value>0.386263</value>
        <value>0.29458746</value>
        <value>0.21664503</value>
        <value>0.15469787</value>
        <value>0.10799739</value>
        <value>0.00628571</value>
        <value>0.010204564</value>
        <value>0.013688075</value>
        <value>0.015103889</value>
        <value>0.013630667</value>
        <value>0.009991102</value>
        <value>0.005911679</value>
        <value>0.002815787</value>
        <value>0.0010815</value>
        <value>3.36731E-4</value>
        <value>8.5579E-5</value>
        <value>1.7876E-5</value>
    </npt>
    <parentNodes>
        <parentNode name="LnP" />
    </parentNodes>
    <childNodes>
    </childNodes>
</node>
<node nodeId="dev_platform_p" name="Development Platform"
nodeType="Labelled">

```

```

<states>
  <state id="0" name="PC"/>
  <state id="1" name="Mid-Range"/>
  <state id="2" name="Mainframe"/>
  <state id="3" name="Multi"/>
</states>
<npt type="manual">
  <value>0.02321982</value>
  <value>0.04085116</value>
  <value>0.07543211</value>
  <value>0.1373862</value>
  <value>0.23453511</value>
  <value>0.3733123</value>
  <value>0.5566432</value>
  <value>0.75264436</value>
  <value>0.89640284</value>
  <value>0.9660743</value>
  <value>0.9905875</value>
  <value>0.99766254</value>
  <value>0.062192958</value>
  <value>0.1107442</value>
  <value>0.18137585</value>
  <value>0.25677058</value>
  <value>0.29858035</value>
  <value>0.28369343</value>
  <value>0.22128388</value>
  <value>0.13716109</value>
  <value>0.06562736</value>
  <value>0.024900332</value>
  <value>0.007877215</value>
  <value>0.002144969</value>
  <value>0.9125009</value>
  <value>0.83883446</value>
  <value>0.7092476</value>
  <value>0.5183544</value>
  <value>0.3111759</value>
  <value>0.15263607</value>
  <value>0.061464</value>
  <value>0.019668223</value>
  <value>0.004858277</value>
  <value>9.51625E-4</value>
  <value>1.55416E-4</value>
  <value>2.18478E-5</value>
  <value>0.00208629</value>
  <value>0.009570159</value>
  <value>0.03394445</value>
  <value>0.08748878</value>
  <value>0.15570863</value>
  <value>0.19035822</value>
  <value>0.16060892</value>
  <value>0.09052632</value>
  <value>0.03311155</value>
  <value>0.008073781</value>
  <value>0.001379902</value>
  <value>1.70658E-4</value>
</npt>
<parentNodes>
  <parentNode name="LnP" />
</parentNodes>
<childNodes>
</childNodes>
</node>
<node nodeId="case" name="CASE Tool Used" nodeType="Boolean">
  <states>
    <state id="0" name="False"/>
    <state id="1" name="True"/>
  </states>
  <npt type="manual">

```

```

    <value>0.12267288</value>
    <value>0.11739238</value>
    <value>0.12160069</value>
    <value>0.13618332</value>
    <value>0.16419251</value>
    <value>0.21125808</value>
    <value>0.28549638</value>
    <value>0.39475724</value>
    <value>0.5380864</value>
    <value>0.69479674</value>
    <value>0.8295783</value>
    <value>0.91928273</value>
    <value>0.9668385</value>
    <value>0.87732714</value>
    <value>0.88260764</value>
    <value>0.8783993</value>
    <value>0.8638167</value>
    <value>0.8358075</value>
    <value>0.78874195</value>
    <value>0.7145036</value>
    <value>0.6052428</value>
    <value>0.4619136</value>
    <value>0.30520326</value>
    <value>0.17042173</value>
    <value>0.08071726</value>
    <value>0.033161517</value>
  </npt>
</parentNodes>
  <parentNode name="LnD" />
</parentNodes>
<childNodes>
  </childNodes>
</node>
<node nodeId="meth_d" name="Used Methodology" nodeType="Boolean">
  <states>
    <state id="0" name="False"/>
    <state id="1" name="True"/>
  </states>
  <npt type="manual">
    <value>0.005137622</value>
    <value>0.009425157</value>
    <value>0.016420292</value>
    <value>0.027136939</value>
    <value>0.04249669</value>
    <value>0.06301187</value>
    <value>0.08845228</value>
    <value>0.11764706</value>
    <value>0.14855064</value>
    <value>0.17857762</value>
    <value>0.20506994</value>
    <value>0.22571257</value>
    <value>0.23878588</value>
    <value>0.9948624</value>
    <value>0.99057484</value>
    <value>0.9835797</value>
    <value>0.9728631</value>
    <value>0.9575033</value>
    <value>0.9369881</value>
    <value>0.9115477</value>
    <value>0.88235295</value>
    <value>0.8514494</value>
    <value>0.8214224</value>
    <value>0.79493004</value>
    <value>0.7742874</value>
    <value>0.76121414</value>
  </npt>
</parentNodes>
  <parentNode name="LnD" />

```

```

    </parentNodes>
    <childNodes>
    </childNodes>
  </node>
  <node nodeId="meth_d" name="Used Methodology" nodeType="Boolean">
    <states>
      <state id="0" name="False"/>
      <state id="1" name="True"/>
    </states>
    <npt type="manual">
      <value>0.005137622</value>
      <value>0.009425157</value>
      <value>0.016420292</value>
      <value>0.027136939</value>
      <value>0.04249669</value>
      <value>0.06301187</value>
      <value>0.08845228</value>
      <value>0.11764706</value>
      <value>0.14855064</value>
      <value>0.17857762</value>
      <value>0.20506994</value>
      <value>0.22571257</value>
      <value>0.23878588</value>
      <value>0.9948624</value>
      <value>0.99057484</value>
      <value>0.9835797</value>
      <value>0.9728631</value>
      <value>0.9575033</value>
      <value>0.9369881</value>
      <value>0.9115477</value>
      <value>0.88235295</value>
      <value>0.8514494</value>
      <value>0.8214224</value>
      <value>0.79493004</value>
      <value>0.7742874</value>
      <value>0.76121414</value>
    </npt>
    <parentNodes>
      <parentNode name="LnD" />
    </parentNodes>
    <childNodes>
    </childNodes>
  </node>
</object>

```

Appendix D Defect Types Model

D.1 Statistical analysis of defect types

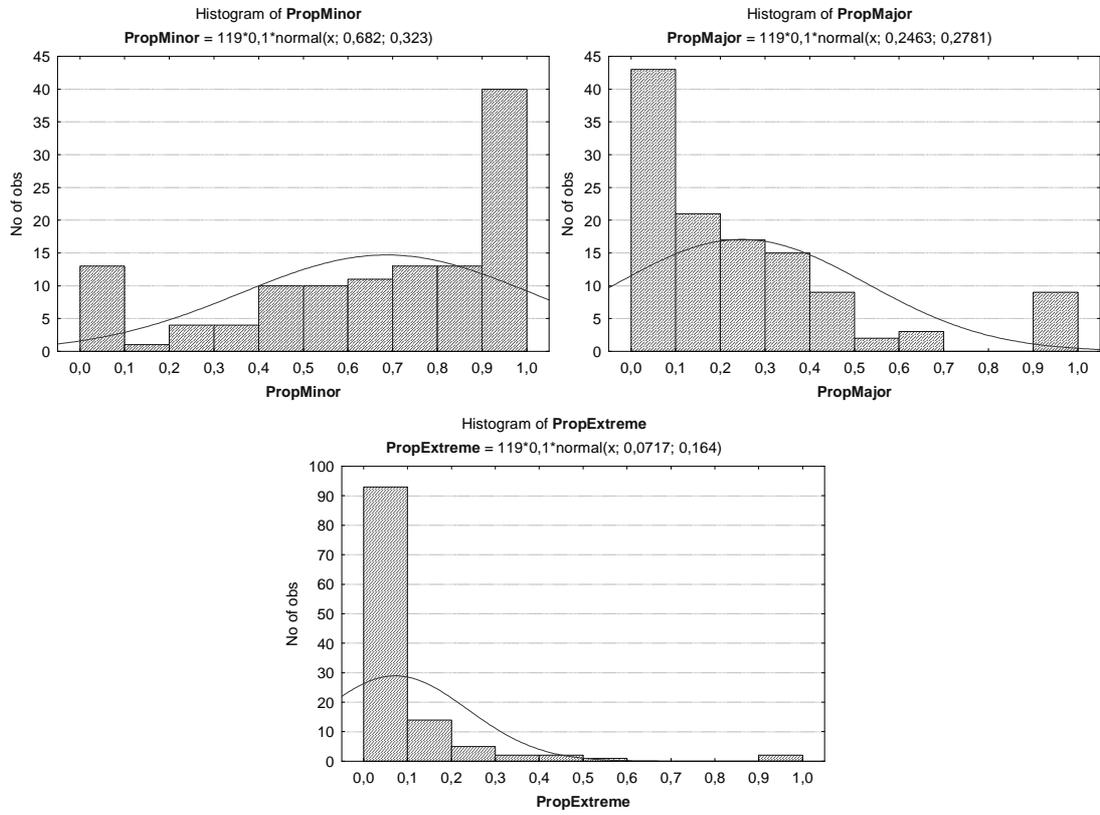


Figure D-1 Histograms for proportions of minor, major and extreme defects

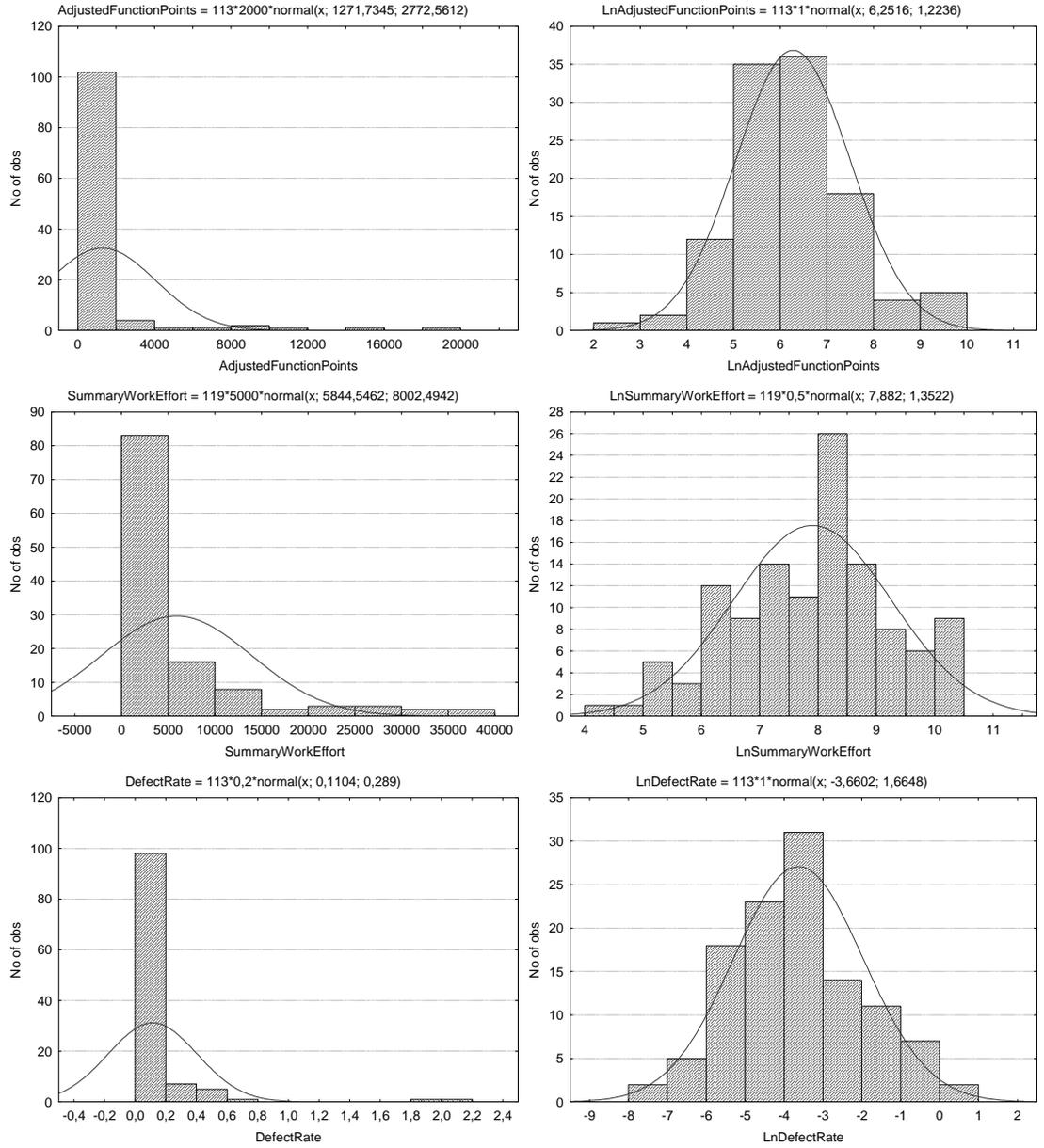


Figure D-2 Histograms for continuous predictors before and after transformation with natural logarithm

Table D-1 Spearman's rank correlation coefficients (SRCC) between dependent variables and numeric predictors

| Predictor \ Dependent | Prop Minor | Prop Major | Prop Extreme |
|---|--------------|--------------|--------------|
| Average Team Size | -0.10 | 0.09 | 0.13 |
| Defect Rate | -0.27 | 0.35 | 0.35 |
| Functional Size | 0.01 | 0.00 | 0.23 |
| Max Team Size | -0.17 | 0.17 | 0.24 |
| Productivity Rate | -0.14 | 0.08 | 0.29 |
| Project Elapsed Time | 0.10 | -0.09 | -0.04 |
| Summary Work Effort | 0.09 | -0.03 | -0.04 |
| User Base Business Units | 0.12 | -0.10 | 0.04 |
| User Base Concurrent Users | 0.05 | -0.01 | 0.01 |
| User Base Locations | 0.11 | -0.10 | -0.12 |
| Value Adjustment Factor | 0.12 | -0.03 | -0.09 |
| <i>Prop Minor</i> | - | -0.90 | -0.57 |
| <i>Prop Major</i> | -0.90 | - | 0.33 |
| <i>Prop Extreme</i> | -0.57 | 0.33 | - |
| values marked in bold are statistically significant at p<0.05 | | | |

Table D-2 Summary of KW-ANOVA H test statistic values

| | Prop Minor | Prop Major | Prop Extreme |
|---|---------------------|--------------------|---------------------|
| Activity Build | 0.17 (0.68) | 0.01 (0.92) | 0.67 (0.41) |
| Activity Design | 0.43 (0.51) | 0.08 (0.78) | 0.52 (0.47) |
| Activity Implementation | 6.01 (0.01) | 7.06 (0.01) | 9.33 (0.00) |
| Activity Planning | 0.31 (0.58) | 1.80 (0.18) | 1.34 (0.25) |
| Activity Specification | 1.70 (0.19) | 1.63 (0.20) | 0.84 (0.36) |
| Activity Test | 11.85 (0.00) | 9.55 (0.00) | 30.99 (0.00) |
| Application Type | 1.22 (0.87) | 1.63 (0.80) | 2.21 (0.70) |
| Architecture | 1.15 (0.56) | 1.18 (0.55) | 2.87 (0.24) |
| Business Area Type | 9.42 (0.58) | 8.54 (0.66) | 14.27 (0.22) |
| CASE Tool Used | 0.02 (0.89) | 0.48 (0.49) | 3.12 (0.08) |
| Client-Server | 0.49 (0.48) | 0.65 (0.42) | 3.15 (0.08) |
| Development Platform | 1.12 (0.57) | 0.61 (0.74) | 0.20 (0.90) |
| Development Type | 2.21 (0.14) | 1.45 (0.23) | 0.35 (0.55) |
| Intended Market | 6.76 (0.08) | 6.71 (0.08) | 25.65 (0.00) |
| Language Type | 2.89 (0.24) | 0.38 (0.83) | 4.23 (0.12) |
| Organisation Type | 4.86 (0.77) | 6.07 (0.64) | 7.38 (0.50) |
| Package Customisation | 0.96 (0.33) | 0.39 (0.53) | 10.65 (0.00) |
| Primary Programming Language | 9.75 (0.37) | 8.29 (0.51) | 19.44 (0.02) |
| Used Methodology | 1.28 (0.26) | 1.48 (0.22) | 7.97 (0.00) |
| values marked in bold are statistically significant at p<0.05 p values in brackets | | | |

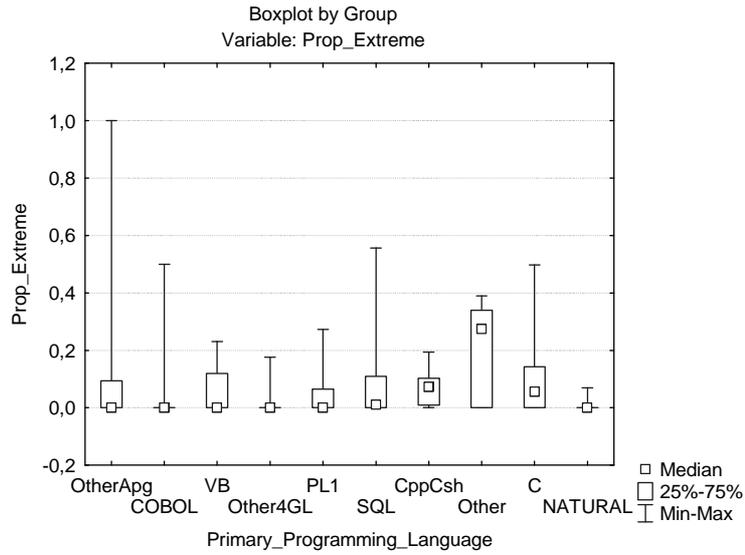


Figure D-3 Box-plot for proportion of extreme defects by *primary programming language*

Table D-3 SRCCs between numeric predictor variables

| | Productivity Rate | Defect Rate | Functional Size |
|-------------------|-------------------|-------------|-----------------|
| Productivity Rate | - | 0.01 | 0.25 |
| Defect Rate | 0.01 | - | -0.15 |
| Functional Size | 0.25 | -0.15 | - |

values marked in bold are statistically significant at $p < 0.05$

Table D-4 Associations between categorical predictors

| | Package Customisation | Used Methodology | Activity Test | Activity Implement | Intended Market |
|-----------------------|-----------------------|------------------|---------------|--------------------|-----------------|
| Package Customisation | - | 0.19 | 0.42 | 0.07 | 0.27 |
| Used Methodology | 0.19 | - | 0.81 | 0.61 | 0.55 |
| | 0.19 | | 0.81 | 0.61 | 0.55 |
| | 0.19 | | 0.63 | 0.52 | 0.48 |
| Activity Test | 0.42 | 0.81 | - | 0.67 | 0.73 |
| | 0.42 | 0.81 | | 0.67 | 0.73 |
| | 0.39 | 0.63 | | 0.56 | 0.59 |
| Activity Implement | 0.07 | 0.61 | 0.67 | - | 0.50 |
| | 0.07 | 0.61 | 0.67 | | 0.50 |
| | 0.07 | 0.52 | 0.56 | | 0.45 |
| Intended Market | 0.27 | 0.55 | 0.73 | 0.50 | - |
| | 0.27 | 0.55 | 0.73 | 0.50 | |
| | 0.26 | 0.48 | 0.59 | 0.45 | |

values in each cell are: Phi (top), Cramer's V (middle) and contingency coefficient (bottom)
values above 0.5 marked in bold

Table D-5 Summary of KW-ANOVA H test statistic values between numeric and categorical predictors

| | Defect Rate | Functional Size |
|---|---------------------|---------------------|
| Package Customisation | 0.70 (0.40) | 4.89 (0.03) |
| Activity Test | 17.23 (0.00) | 19.64 (0.00) |
| values marked in bold are statistically significant at $p < 0.05$ p values in brackets | | |

D.2 Definition of defect types model

```

<?xml version="1.0" encoding="iso-8859-2" ?>
<object name="Defect Types" type="structure">
  <node nodeId="prop_defects" name="Prop of Defects" nodeType="Labelled">
    <states>
      <state id="0" name="Prop. Minor"/>
      <state id="1" name="Prop. Major"/>
      <state id="2" name="Prop. Extreme"/>
    </states>
    <npt type="manual">
      <value>0.68</value>
      <value>0.25</value>
      <value>0.07</value>
    </npt>
    <parentNodes>
    </parentNodes>
    <childNodes>
      <childNodes name="pack_custom" />
      <childNodes name="act_test" />
      <childNodes name="func_size" />
      <childNodes name="task_complexity" />
      <childNodes name="act_build" />
      <childNodes name="deadline_pressure" />
      <childNodes name="act_spec" />
      <childNodes name="gui_usage" />
    </childNodes>
  </node>
  <node nodeId="act_spec" name="Activity specification" nodeType="Ranked">
    <states>
      <state id="0" name="Very Low"/>
      <state id="1" name="Low"/>
      <state id="2" name="Medium"/>
      <state id="3" name="High"/>
      <state id="4" name="Very High"/>
    </states>
    <npt type="manual">
      <value>0.0882353</value>
      <value>0.116</value>
      <value>0.15714286</value>
      <value>0.28455883</value>
      <value>0.324</value>
      <value>0.3642857</value>
      <value>0.35</value>
      <value>0.35</value>
      <value>0.35</value>
      <value>0.21764706</value>
      <value>0.176</value>
      <value>0.114285715</value>
      <value>0.059558824</value>
      <value>0.034</value>
      <value>0.014285714</value>
    </npt>
    <parentNodes>
      <parentNode name="prop_defects" />
    </parentNodes>
    <childNodes>
    </childNodes>
  </node>
  <node nodeId="task_complexity" name="Task complexity" nodeType="Ranked">
    <states>
      <state id="0" name="Very Low"/>
      <state id="1" name="Low"/>
      <state id="2" name="Medium"/>
      <state id="3" name="High"/>
    </states>
  </node>

```

```

    <state id="4" name="Very High"/>
  </states>
  <npt type="manual">
    <value>0.11029412</value>
    <value>0.08</value>
    <value>0.071428575</value>
    <value>0.21029411</value>
    <value>0.18</value>
    <value>0.17142858</value>
    <value>0.4</value>
    <value>0.4</value>
    <value>0.4</value>
    <value>0.18970588</value>
    <value>0.22</value>
    <value>0.22857143</value>
    <value>0.089705884</value>
    <value>0.12</value>
    <value>0.12857144</value>
  </npt>
  <parentNodes>
    <parentNode name="prop_defects" />
  </parentNodes>
  <childNodes>
  </childNodes>
</node>
<node nodeId="deadline_pressure" name="Deadline Pressure"
nodeType="Ranked">
  <states>
    <state id="0" name="Very Low"/>
    <state id="1" name="Low"/>
    <state id="2" name="Medium"/>
    <state id="3" name="High"/>
    <state id="4" name="Very High"/>
  </states>
  <npt type="manual">
    <value>0.08093591</value>
    <value>0.13974845</value>
    <value>0.14285715</value>
    <value>0.27900818</value>
    <value>0.34497905</value>
    <value>0.34285715</value>
    <value>0.35898757</value>
    <value>0.3326013</value>
    <value>0.325</value>
    <value>0.22220588</value>
    <value>0.15172689</value>
    <value>0.15714286</value>
    <value>0.05886248</value>
    <value>0.0309443</value>
    <value>0.032142855</value>
  </npt>
  <parentNodes>
    <parentNode name="prop_defects" />
  </parentNodes>
  <childNodes>
  </childNodes>
</node>
<node nodeId="func_size" name="Functional Size" nodeType="Integer
Interval">
  <states>
    <state id="0" lowerBound="2.0" upperBound="5.0"/>
    <state id="1" lowerBound="5.0" upperBound="12.0"/>
    <state id="2" lowerBound="12.0" upperBound="33.0"/>
    <state id="3" lowerBound="33.0" upperBound="90.0"/>
    <state id="4" lowerBound="90.0" upperBound="245.0"/>
    <state id="5" lowerBound="245.0" upperBound="665.0"/>
    <state id="6" lowerBound="665.0" upperBound="1808.0"/>
    <state id="7" lowerBound="1808.0" upperBound="4915.0"/>
  </states>

```

```

        <state id="8" lowerBound="4915.0" upperBound="13360.0"/>
        <state id="9" lowerBound="13360.0" upperBound="36317.0"/>
</states>
<npt type="manual">
  <value>0.001021799</value>
  <value>0.001041416</value>
  <value>6.41327E-4</value>
  <value>0.010188584</value>
  <value>0.010334054</value>
  <value>0.00698334</value>
  <value>0.03047754</value>
  <value>0.030761836</value>
  <value>0.022660226</value>
  <value>0.08092059</value>
  <value>0.081390694</value>
  <value>0.06612795</value>
  <value>0.22156128</value>
  <value>0.222062</value>
  <value>0.19752875</value>
  <value>0.31985685</value>
  <value>0.31943443</value>
  <value>0.30914816</value>
  <value>0.2189737</value>
  <value>0.21853721</value>
  <value>0.23515327</value>
  <value>0.07833301</value>
  <value>0.07818633</value>
  <value>0.10261234</value>
  <value>0.029110242</value>
  <value>0.028839221</value>
  <value>0.04275514</value>
  <value>0.009556392</value>
  <value>0.009412801</value>
  <value>0.01638947</value>
</npt>
<parentNodes>
  <parentNode name="prop_defects" />
</parentNodes>
<childNodes>
</childNodes>
</node>
<node nodeId="pack_custom" name="Package Customisation" nodeType="Ranked">
  <states>
    <state id="0" name="None"/>
    <state id="1" name="Low"/>
    <state id="2" name="Medium"/>
    <state id="3" name="High"/>
  </states>
  <npt type="manual">
    <value>0.51140547</value>
    <value>0.51</value>
    <value>0.35460994</value>
    <value>0.29580575</value>
    <value>0.3</value>
    <value>0.34042552</value>
    <value>0.14569536</value>
    <value>0.144</value>
    <value>0.21276596</value>
    <value>0.04709345</value>
    <value>0.046</value>
    <value>0.09219858</value>
  </npt>
  <parentNodes>
    <parentNode name="prop_defects" />
  </parentNodes>
  <childNodes>
  </childNodes>
</node>

```

```

<node nodeId="act_test" name="Activity Test" nodeType="Ranked">
  <states>
    <state id="0" name="Very Low"/>
    <state id="1" name="Low"/>
    <state id="2" name="Medium"/>
    <state id="3" name="High"/>
    <state id="4" name="Very High"/>
  </states>
  <npt type="manual">
    <value>0.07647621</value>
    <value>0.14</value>
    <value>0.18558173</value>
    <value>0.27354953</value>
    <value>0.348</value>
    <value>0.38543898</value>
    <value>0.35259944</value>
    <value>0.35</value>
    <value>0.32476804</value>
    <value>0.23119347</value>
    <value>0.144</value>
    <value>0.09707352</value>
    <value>0.06618134</value>
    <value>0.018</value>
    <value>0.007137759</value>
  </npt>
  <parentNodes>
    <parentNode name="prop_defects" />
  </parentNodes>
  <childNodes>
  </childNodes>
</node>
<node nodeId="act_build" name="Activity Build" nodeType="Ranked">
  <states>
    <state id="0" name="Very Low"/>
    <state id="1" name="Low"/>
    <state id="2" name="Medium"/>
    <state id="3" name="High"/>
    <state id="4" name="Very High"/>
  </states>
  <npt type="manual">
    <value>0.092578985</value>
    <value>0.112</value>
    <value>0.1294964</value>
    <value>0.28875828</value>
    <value>0.318</value>
    <value>0.34532374</value>
    <value>0.34974283</value>
    <value>0.35</value>
    <value>0.352518</value>
    <value>0.21307862</value>
    <value>0.18</value>
    <value>0.1438849</value>
    <value>0.055841293</value>
    <value>0.04</value>
    <value>0.028776977</value>
  </npt>
  <parentNodes>
    <parentNode name="prop_defects" />
  </parentNodes>
  <childNodes>
  </childNodes>
</node>
<node nodeId="gui_usage" name="GUI usage" nodeType="Ranked">
  <states>
    <state id="0" name="None"/>
    <state id="1" name="Low"/>
    <state id="2" name="Medium"/>
    <state id="3" name="High"/>
  </states>

```

```
</states>
<npt type="manual">
  <value>0.076425634</value>
  <value>0.14428857</value>
  <value>0.17118402</value>
  <value>0.17195767</value>
  <value>0.26452905</value>
  <value>0.2425107</value>
  <value>0.29012346</value>
  <value>0.31863728</value>
  <value>0.32952926</value>
  <value>0.46149325</value>
  <value>0.2725451</value>
  <value>0.25677603</value>
</npt>
<parentNodes>
  <parentNode name="prop_defects" />
</parentNodes>
<childNodes>
</childNodes>
</node>
</object>
```

Appendix E Learning Model for Testing and Fixing Defects

E.1 Definition of Learning Model for Testing and Fixing Defects

```

<?xml version="1.0" encoding="iso-8859-2" ?>
<object name="Priors" type="structure">
  <node nodeId="residual_defects" name="residual defects" nodeType="Integer
Interval">
    <states>
      <state id="0" lowerBound="0.0" upperBound="10.0"/>
      <state id="1" lowerBound="10.0" upperBound="Infinity"/>
    </states>
    <npt type="expression">
      <expression>Normal(400,100000)</expression>
    </npt>
    <parentNodes>
    </parentNodes>
    <childNodes>
    </childNodes>
  </node>
  <node nodeId="open_defects" name="open defects" nodeType="Integer
Interval">
    <states>
      <state id="0" lowerBound="0.0" upperBound="10.0"/>
      <state id="1" lowerBound="10.0" upperBound="Infinity"/>
    </states>
    <npt type="expression">
      <expression>Normal(0,1000000)</expression>
    </npt>
    <parentNodes>
    </parentNodes>
    <childNodes>
    </childNodes>
  </node>
  <node nodeId="testing_ppq_multiplier" name="testing ppq multiplier"
nodeType="Continuous Interval">
    <states>
      <state id="0" lowerBound="0.0" upperBound="1.0"/>
      <state id="1" lowerBound="1.0" upperBound="1.0E38"/>
    </states>
    <npt type="expression">
      <expression>Uniform(0,500)</expression>
    </npt>
    <parentNodes>
    </parentNodes>
    <childNodes>
    </childNodes>
  </node>
  <node nodeId="fixing_ppq_multiplier" name="fixing ppq multiplier"
nodeType="Continuous Interval">
    <states>
      <state id="0" lowerBound="0.0" upperBound="1.0"/>
      <state id="1" lowerBound="1.0" upperBound="Infinity"/>
    </states>
    <npt type="expression">
      <expression>Uniform(0,500)</expression>
    </npt>
    <parentNodes>
    </parentNodes>
    <childNodes>
    </childNodes>
  </node>

```

```

    </node>
    <node nodeId="testing_bias_multiplier" name="testing bias multiplier"
nodeType="Continuous Interval">
      <states>
        <state id="0" lowerBound="0.0" upperBound="1.0"/>
        <state id="1" lowerBound="1.0" upperBound="1.0E38"/>
      </states>
      <npt type="expression">
        <expression>Uniform(0,500)</expression>
      </npt>
      <parentNodes>
      </parentNodes>
      <childNodes>
      </childNodes>
    </node>
    <node nodeId="fixing_bias_multiplier" name="fixing bias multiplier"
nodeType="Continuous Interval">
      <states>
        <state id="0" lowerBound="0.0" upperBound="1.0"/>
        <state id="1" lowerBound="1.0" upperBound="1.0E38"/>
      </states>
      <npt type="expression">
        <expression>Uniform(0,500)</expression>
      </npt>
      <parentNodes>
      </parentNodes>
      <childNodes>
      </childNodes>
    </node>
    <node nodeId="testing_effort_multiplier" name="testing effort multiplier"
nodeType="Continuous Interval">
      <states>
        <state id="0" lowerBound="0.0" upperBound="1.0"/>
        <state id="1" lowerBound="1.0" upperBound="1.0E38"/>
      </states>
      <npt type="expression">
        <expression>Uniform(0,500)</expression>
      </npt>
      <parentNodes>
      </parentNodes>
      <childNodes>
      </childNodes>
    </node>
    <node nodeId="fixing_effort_multiplier" name="fixing effort multiplier"
nodeType="Continuous Interval">
      <states>
        <state id="0" lowerBound="0.0" upperBound="1.0"/>
        <state id="1" lowerBound="1.0" upperBound="1.0E38"/>
      </states>
      <npt type="expression">
        <expression>Uniform(0,500)</expression>
      </npt>
      <parentNodes>
      </parentNodes>
      <childNodes>
      </childNodes>
    </node>
  </object>

<?xml version="1.0" encoding="iso-8859-2" ?>
<object name="Main" type="structure">
  <node nodeId="pot_fixed" name="potentially fixed defects"
nodeType="Continuous Interval">
    <states>
      <state id="0" lowerBound="0.0" upperBound="1.0"/>
      <state id="1" lowerBound="1.0" upperBound="Infinity"/>
    </states>

```

```

    <npt type="expression">
      <expression>Normal(fixing_effort_adj * 2 ^
fixing_ppq_bias_dummy,fixing_effort_adj * 2 ^ fixing_ppq_bias_dummy /
10)</expression>
    </npt>
    <parentNodes>
      <parentNode name="fixing_ppq_bias_dummy" />
      <parentNode name="fixing_effort_adj" />
    </parentNodes>
    <childNodes>
      <childNode name="fixed" />
    </childNodes>
  </node>
  <node nodeId="residual_defects_pre" name="residual defects pre"
nodeType="Integer Interval">
    <states>
      <state id="0" lowerBound="0.0" upperBound="11.0"/>
      <state id="1" lowerBound="11.0" upperBound="Infinity"/>
    </states>
    <npt type="expression">
      <expression>Arithmetic(residual_defects_input)</expression>
    </npt>
    <parentNodes>
      <parentNode name="residual_defects_input" />
    </parentNodes>
    <childNodes>
      <childNode name="residual_found" />
      <childNode name="found" />
    </childNodes>
  </node>
  <node nodeId="open_defects_input" name="open defects input"
nodeType="Integer Interval">
    <states>
      <state id="0" lowerBound="0.0" upperBound="10.0"/>
      <state id="1" lowerBound="10.0" upperBound="Infinity"/>
    </states>
    <npt type="expression">
      <expression>Normal(0,1000000)</expression>
    </npt>
    <parentNodes>
    </parentNodes>
    <childNodes>
      <childNode name="total_open" />
    </childNodes>
  </node>
  <node nodeId="found" name="defects found" nodeType="Integer Interval">
    <states>
      <state id="0" lowerBound="0.0" upperBound="10.0"/>
      <state id="1" lowerBound="10.0" upperBound="Infinity"/>
    </states>
    <npt type="expression">
      <expression>Arithmetic(residual_defects_pre *
testing_eff)</expression>
    </npt>
    <parentNodes>
      <parentNode name="testing_eff" />
      <parentNode name="residual_defects_pre" />
    </parentNodes>
    <childNodes>
      <childNode name="total_open" />
      <childNode name="residual_found" />
    </childNodes>
  </node>
  <node nodeId="total_open" name="total open defects" nodeType="Integer
Interval">
    <states>
      <state id="0" lowerBound="0.0" upperBound="10.0"/>
      <state id="1" lowerBound="10.0" upperBound="Infinity"/>

```

```

</states>
<npt type="expression">
  <expression>Arithmetic(open_defects_input + found)</expression>
</npt>
<parentNodes>
  <parentNode name="found" />
  <parentNode name="open_defects_input" />
</parentNodes>
<childNodes>
  <childNodes name="open_defects" />
  <childNodes name="fixed" />
</childNodes>
</node>
<node nodeId="open_defects" name="open defects" nodeType="Integer
Interval">
  <states>
    <state id="0" lowerBound="0.0" upperBound="10.0"/>
    <state id="1" lowerBound="10.0" upperBound="Infinity"/>
  </states>
  <npt type="expression">
    <expression>Arithmetic(max(0, total_open - fixed))</expression>
  </npt>
  <parentNodes>
    <parentNode name="total_open" />
    <parentNode name="fixed" />
  </parentNodes>
  <childNodes>
  </childNodes>
</node>
<node nodeId="inserted" name="defects inserted" nodeType="Integer
Interval">
  <states>
    <state id="0" lowerBound="0.0" upperBound="10.0"/>
    <state id="1" lowerBound="10.0" upperBound="Infinity"/>
  </states>
  <npt type="expression">
    <expression>Arithmetic(fixed * (3.5 ^ (-fixing_ppq_bias_dummy) *
0.1))</expression>
  </npt>
  <parentNodes>
    <parentNode name="fixed" />
    <parentNode name="fixing_ppq_bias_dummy" />
  </parentNodes>
  <childNodes>
    <childNodes name="residual_defects" />
  </childNodes>
</node>
<node nodeId="fixed" name="defects fixed" nodeType="Integer Interval">
  <states>
    <state id="0" lowerBound="0.0" upperBound="10.0"/>
    <state id="1" lowerBound="10.0" upperBound="Infinity"/>
  </states>
  <npt type="expression">
    <expression>Arithmetic(min(total_open, pot_fixed))</expression>
  </npt>
  <parentNodes>
    <parentNode name="total_open" />
    <parentNode name="pot_fixed" />
  </parentNodes>
  <childNodes>
    <childNodes name="open_defects" />
    <childNodes name="inserted" />
  </childNodes>
</node>
<node nodeId="testing_eff" name="testing effectiveness"
nodeType="Continuous Interval">
  <states>
    <state id="0" lowerBound="0.0" upperBound="1.0"/>

```

```

    </states>
    <npt type="expression">
      <expression>TNormal(testing_effort_adj * 2 ^
testing_ppq_bias_dummy,0.01,0.0,1.0)</expression>
    </npt>
    <parentNodes>
      <parentNode name="testing_ppq_bias_dummy" />
      <parentNode name="testing_effort_adj" />
    </parentNodes>
    <childNodes>
      <childNode name="found" />
    </childNodes>
  </node>
  <node nodeId="testing_ppq" name="testing process and people quality"
nodeType="Ranked">
    <states>
      <state id="0" name="Lowest" />
      <state id="1" name="Very Low" />
      <state id="2" name="Low" />
      <state id="3" name="Medium" />
      <state id="4" name="High" />
      <state id="5" name="Very High" />
      <state id="6" name="Highest" />
    </states>
    <npt type="expression">
      <expression>TNormal(0.5,0.01,0.0,1.0)</expression>
    </npt>
    <parentNodes>
    </parentNodes>
    <childNodes>
      <childNode name="testing_ppq_adj" />
    </childNodes>
  </node>
  <node nodeId="testing_oth_f" name="testing other factors"
nodeType="Ranked">
    <states>
      <state id="0" name="Lowest" />
      <state id="1" name="Very Low" />
      <state id="2" name="Low" />
      <state id="3" name="Medium" />
      <state id="4" name="High" />
      <state id="5" name="Very High" />
      <state id="6" name="Highest" />
    </states>
    <npt type="expression">
      <expression>TNormal(0.5,0.01,0.0,1.0)</expression>
    </npt>
    <parentNodes>
    </parentNodes>
    <childNodes>
      <childNode name="testing_oth_f_adj" />
    </childNodes>
  </node>
  <node nodeId="testing_effort" name="testing effort" nodeType="Continuous
Interval">
    <states>
      <state id="0" lowerBound="0.0" upperBound="1.0" />
      <state id="1" lowerBound="1.0" upperBound="Infinity" />
    </states>
    <npt type="expression">
      <expression>Normal(0,1000000)</expression>
    </npt>
    <parentNodes>
    </parentNodes>
    <childNodes>
      <childNode name="testing_effort_adj" />
    </childNodes>
  </node>

```

```

    <node nodeId="testing_ppq_bias_dummy" name="testing ppq + bias dummy"
nodeType="Continuous Interval">
    <states>
        <state id="0" lowerBound="-Infinity" upperBound="0.0"/>
        <state id="1" lowerBound="0.0" upperBound="10.0"/>
        <state id="2" lowerBound="10.0" upperBound="Infinity"/>
    </states>
    <npt type="expression">
        <expression>Arithmetic(wmean(1, testing_ppq_adj, 1,
testing_oth_f_adj))</expression>
    </npt>
    <parentNodes>
        <parentNode name="testing_oth_f_adj" />
        <parentNode name="testing_ppq_adj" />
    </parentNodes>
    <childNodes>
        <childNodes name="testing_eff" />
    </childNodes>
</node>
    <node nodeId="residual_found" name="residual - found" nodeType="Integer
Interval">
    <states>
        <state id="0" lowerBound="0.0" upperBound="10.0"/>
        <state id="1" lowerBound="10.0" upperBound="Infinity"/>
    </states>
    <npt type="expression">
        <expression>Arithmetic(max(0, residual_defects_pre -
found))</expression>
    </npt>
    <parentNodes>
        <parentNode name="residual_defects_pre" />
        <parentNode name="found" />
    </parentNodes>
    <childNodes>
        <childNodes name="residual_defects" />
    </childNodes>
</node>
    <node nodeId="residual_defects" name="residual defects" nodeType="Integer
Interval">
    <states>
        <state id="0" lowerBound="0.0" upperBound="10.0"/>
        <state id="1" lowerBound="10.0" upperBound="Infinity"/>
    </states>
    <npt type="expression">
        <expression>Arithmetic(max(0, residual_found +
inserted))</expression>
    </npt>
    <parentNodes>
        <parentNode name="residual_found" />
        <parentNode name="inserted" />
    </parentNodes>
    <childNodes>
    </childNodes>
</node>
    <node nodeId="testing_ppq_multiplier" name="testing process and people
quality multiplier" nodeType="Continuous Interval">
    <states>
        <state id="0" lowerBound="0.0" upperBound="1.0"/>
        <state id="1" lowerBound="1.0" upperBound="Infinity"/>
    </states>
    <npt type="expression">
        <expression>Arithmetic(testing_ppq_multiplier_input)</expression>
    </npt>
    <parentNodes>
        <parentNode name="testing_ppq_multiplier_input" />
    </parentNodes>
    <childNodes>
        <childNodes name="testing_ppq_adj" />
    </childNodes>

```

```

    </childNodes>
  </node>
  <node nodeId="testing_oth_f_multiplier" name="testing other factors
multiplier" nodeType="Continuous Interval">
    <states>
      <state id="0" lowerBound="0.0" upperBound="1.0"/>
      <state id="1" lowerBound="1.0" upperBound="Infinity"/>
    </states>
    <npt type="expression">

<expression>Arithmetic(testing_oth_f_multiplier_input)</expression>
    </npt>
    <parentNodes>
      <parentNode name="testing_oth_f_multiplier_input" />
    </parentNodes>
    <childNodes>
      <childNode name="testing_oth_f_adj" />
    </childNodes>
  </node>
  <node nodeId="testing_effort_multiplier" name="testing effort multiplier"
nodeType="Continuous Interval">
    <states>
      <state id="0" lowerBound="0.0" upperBound="10.0"/>
    </states>
    <npt type="expression">

<expression>Arithmetic(testing_effort_multiplier_input)</expression>
    </npt>
    <parentNodes>
      <parentNode name="testing_effort_multiplier_input" />
    </parentNodes>
    <childNodes>
      <childNode name="testing_effort_adj" />
    </childNodes>
  </node>
  <node nodeId="testing_ppq_adj" name="testing process and people quality
adjusted" nodeType="Continuous Interval">
    <states>
      <state id="0" lowerBound="-Infinity" upperBound="0.0"/>
      <state id="1" lowerBound="0.0" upperBound="1.0"/>
      <state id="2" lowerBound="1.0" upperBound="Infinity"/>
    </states>
    <npt type="expression">
      <expression>Arithmetic((testing_ppq - 0.5) *
testing_ppq_multiplier)</expression>
    </npt>
    <parentNodes>
      <parentNode name="testing_ppq_multiplier" />
      <parentNode name="testing_ppq" />
    </parentNodes>
    <childNodes>
      <childNode name="testing_ppq_bias_dummy" />
    </childNodes>
  </node>
  <node nodeId="testing_oth_f_adj" name="testing other factors adjusted"
nodeType="Continuous Interval">
    <states>
      <state id="0" lowerBound="-Infinity" upperBound="0.0"/>
      <state id="1" lowerBound="0.0" upperBound="1.0"/>
      <state id="2" lowerBound="1.0" upperBound="Infinity"/>
    </states>
    <npt type="expression">
      <expression>Arithmetic((testing_oth_f - 0.5) *
testing_oth_f_multiplier)</expression>
    </npt>
    <parentNodes>
      <parentNode name="testing_oth_f_multiplier" />
      <parentNode name="testing_oth_f" />

```

```

    </parentNodes>
    <childNodes>
      <childNode name="testing_ppq_bias_dummy" />
    </childNodes>
  </node>
  <node nodeId="testing_effort_adj" name="testing effort adjusted"
nodeType="Continuous Interval">
    <states>
      <state id="0" lowerBound="0.0" upperBound="1.0"/>
      <state id="1" lowerBound="1.0" upperBound="Infinity"/>
    </states>
    <npt type="expression">
      <expression>Arithmetic(testing_effort/100 *
testing_effort_multiplier)</expression>
    </npt>
    <parentNodes>
      <parentNode name="testing_effort_multiplier" />
      <parentNode name="testing_effort" />
    </parentNodes>
    <childNodes>
      <childNode name="testing_eff" />
    </childNodes>
  </node>
  <node nodeId="testing_ppq_multiplier_input" name="testing process and
people quality multiplier input" nodeType="Continuous Interval">
    <states>
      <state id="0" lowerBound="0.0" upperBound="1.0"/>
      <state id="1" lowerBound="1.0" upperBound="Infinity"/>
    </states>
    <npt type="expression">
      <expression>Normal(1,0.01)</expression>
    </npt>
    <parentNodes>
    </parentNodes>
    <childNodes>
      <childNode name="testing_ppq_multiplier" />
    </childNodes>
  </node>
  <node nodeId="testing_oth_f_multiplier_input" name="testing other factors
multiplier input" nodeType="Continuous Interval">
    <states>
      <state id="0" lowerBound="0.0" upperBound="1.0"/>
      <state id="1" lowerBound="1.0" upperBound="Infinity"/>
    </states>
    <npt type="expression">
      <expression>Normal(1,0.01)</expression>
    </npt>
    <parentNodes>
    </parentNodes>
    <childNodes>
      <childNode name="testing_oth_f_multiplier" />
    </childNodes>
  </node>
  <node nodeId="testing_effort_multiplier_input" name="testing effort
multiplier input" nodeType="Continuous Interval">
    <states>
      <state id="0" lowerBound="0.0" upperBound="1.0"/>
      <state id="1" lowerBound="1.0" upperBound="Infinity"/>
    </states>
    <npt type="expression">
      <expression>Normal(1,0.01)</expression>
    </npt>
    <parentNodes>
    </parentNodes>
    <childNodes>
      <childNode name="testing_effort_multiplier" />
    </childNodes>
  </node>

```

```

<node nodeId="residual_defects_input" name="residual defects input"
nodeType="Integer Interval">
  <states>
    <state id="0" lowerBound="0.0" upperBound="11.0"/>
    <state id="1" lowerBound="11.0" upperBound="Infinity"/>
  </states>
  <npt type="expression">
    <expression>Normal(0,1000000)</expression>
  </npt>
  <parentNodes>
  </parentNodes>
  <childNodes>
    <childNode name="residual_defects_pre" />
  </childNodes>
</node>
<node nodeId="fixing_ppq" name="fixing process and people quality"
nodeType="Ranked">
  <states>
    <state id="0" name="Lowest"/>
    <state id="1" name="Very Low"/>
    <state id="2" name="Low"/>
    <state id="3" name="Medium"/>
    <state id="4" name="High"/>
    <state id="5" name="Very High"/>
    <state id="6" name="Highest"/>
  </states>
  <npt type="expression">
    <expression>TNormal(0.5,0.01,0.0,1.0)</expression>
  </npt>
  <parentNodes>
  </parentNodes>
  <childNodes>
    <childNode name="fixing_ppq_adj" />
  </childNodes>
</node>
<node nodeId="fixing_ppq_adj" name="fixing process and people quality
adjusted" nodeType="Continuous Interval">
  <states>
    <state id="0" lowerBound="-Infinity" upperBound="0.0"/>
    <state id="1" lowerBound="0.0" upperBound="1.0"/>
    <state id="2" lowerBound="1.0" upperBound="Infinity"/>
  </states>
  <npt type="expression">
    <expression>Arithmetic((fixing_ppq - 0.5) *
fixing_ppq_multiplier)</expression>
  </npt>
  <parentNodes>
    <parentNode name="fixing_ppq" />
    <parentNode name="fixing_ppq_multiplier" />
  </parentNodes>
  <childNodes>
    <childNode name="fixing_ppq_bias_dummy" />
  </childNodes>
</node>
<node nodeId="fixing_ppq_multiplier" name="fixing process and people
quality multiplier" nodeType="Continuous Interval">
  <states>
    <state id="0" lowerBound="0.0" upperBound="1.0"/>
    <state id="1" lowerBound="1.0" upperBound="Infinity"/>
  </states>
  <npt type="expression">
    <expression>Arithmetic(fixing_ppq_multiplier_input)</expression>
  </npt>
  <parentNodes>
    <parentNode name="fixing_ppq_multiplier_input" />
  </parentNodes>
  <childNodes>
    <childNode name="fixing_ppq_adj" />
  </childNodes>
</node>

```

```

    </childNodes>
  </node>
  <node nodeId="fixing_ppq_multiplier_input" name="fixing process and people
quality multiplier input" nodeType="Continuous Interval">
  <states>
    <state id="0" lowerBound="0.0" upperBound="1.0"/>
    <state id="1" lowerBound="1.0" upperBound="Infinity"/>
  </states>
  <npt type="expression">
    <expression>Normal(1,0.01)</expression>
  </npt>
  <parentNodes>
  </parentNodes>
  <childNodes>
    <childNodes name="fixing_ppq_multiplier" />
  </childNodes>
</node>
<node nodeId="fixing_oth_f" name="fixing other factors" nodeType="Ranked">
  <states>
    <state id="0" name="Lowest"/>
    <state id="1" name="Very Low"/>
    <state id="2" name="Low"/>
    <state id="3" name="Medium"/>
    <state id="4" name="High"/>
    <state id="5" name="Very High"/>
    <state id="6" name="Highest"/>
  </states>
  <npt type="expression">
    <expression>TNormal(0.5,0.01,0.0,1.0)</expression>
  </npt>
  <parentNodes>
  </parentNodes>
  <childNodes>
    <childNodes name="fixing_oth_f_adj" />
  </childNodes>
</node>
<node nodeId="fixing_oth_f_adj" name="fixing other factors adjusted"
nodeType="Continuous Interval">
  <states>
    <state id="0" lowerBound="-Infinity" upperBound="0.0"/>
    <state id="1" lowerBound="0.0" upperBound="1.0"/>
    <state id="2" lowerBound="1.0" upperBound="Infinity"/>
  </states>
  <npt type="expression">
    <expression>Arithmetic((fixing_oth_f - 0.5) *
fixing_oth_f_multiplier)</expression>
  </npt>
  <parentNodes>
    <parentNode name="fixing_oth_f" />
    <parentNode name="fixing_oth_f_multiplier" />
  </parentNodes>
  <childNodes>
    <childNodes name="fixing_ppq_bias_dummy" />
  </childNodes>
</node>
<node nodeId="fixing_oth_f_multiplier" name="fixing other factors
multiplier" nodeType="Continuous Interval">
  <states>
    <state id="0" lowerBound="0.0" upperBound="1.0"/>
    <state id="1" lowerBound="1.0" upperBound="Infinity"/>
  </states>
  <npt type="expression">
    <expression>Arithmetic(fixing_oth_f_multiplier_input)</expression>
  </npt>
  <parentNodes>
    <parentNode name="fixing_oth_f_multiplier_input" />
  </parentNodes>
  <childNodes>

```

```

        <childNodes name="fixing_oth_f_adj" />
    </childNodes>
</node>
<node nodeId="fixing_oth_f_multiplier_input" name="fixing other factors
multiplier input" nodeType="Continuous Interval">
    <states>
        <state id="0" lowerBound="0.0" upperBound="1.0"/>
        <state id="1" lowerBound="1.0" upperBound="Infinity"/>
    </states>
    <npt type="expression">
        <expression>Normal(1,0.01)</expression>
    </npt>
    <parentNodes>
    </parentNodes>
    <childNodes>
        <childNodes name="fixing_oth_f_multiplier" />
    </childNodes>
</node>
<node nodeId="fixing_effort" name="fixing effort" nodeType="Continuous
Interval">
    <states>
        <state id="0" lowerBound="0.0" upperBound="1.0"/>
        <state id="1" lowerBound="1.0" upperBound="Infinity"/>
    </states>
    <npt type="expression">
        <expression>Normal(0,1000000)</expression>
    </npt>
    <parentNodes>
    </parentNodes>
    <childNodes>
        <childNodes name="fixing_effort_adj" />
    </childNodes>
</node>
<node nodeId="fixing_effort_adj" name="fixing effort adjusted"
nodeType="Continuous Interval">
    <states>
        <state id="0" lowerBound="0.0" upperBound="1.0"/>
        <state id="1" lowerBound="1.0" upperBound="Infinity"/>
    </states>
    <npt type="expression">
        <expression>Arithmetic(fixing_effort /100*
fixing_effort_multiplier)</expression>
    </npt>
    <parentNodes>
        <parentNode name="fixing_effort_multiplier" />
        <parentNode name="fixing_effort" />
    </parentNodes>
    <childNodes>
        <childNodes name="pot_fixed" />
    </childNodes>
</node>
<node nodeId="fixing_effort_multiplier" name="fixing effort multiplier"
nodeType="Continuous Interval">
    <states>
        <state id="0" lowerBound="0.0" upperBound="10.0"/>
    </states>
    <npt type="expression">
        <expression>Arithmetic(fixing_effort_multiplier_input)</expression>
    </npt>
    <parentNodes>
        <parentNode name="fixing_effort_multiplier_input" />
    </parentNodes>
    <childNodes>
        <childNodes name="fixing_effort_adj" />
    </childNodes>
</node>

```

```

<node nodeId="fixing_effort_multiplier_input" name="fixing effort
multiplier input" nodeType="Continuous Interval">
  <states>
    <state id="0" lowerBound="0.0" upperBound="1.0"/>
    <state id="1" lowerBound="1.0" upperBound="Infinity"/>
  </states>
  <npt type="expression">
    <expression>Normal(1,0.01)</expression>
  </npt>
  <parentNodes>
  </parentNodes>
  <childNodes>
    <childNode name="fixing_effort_multiplier" />
  </childNodes>
</node>
<node nodeId="fixing_ppq_bias_dummy" name="fixing ppq + bias dummy"
nodeType="Continuous Interval">
  <states>
    <state id="0" lowerBound="-Infinity" upperBound="0.0"/>
    <state id="1" lowerBound="0.0" upperBound="10.0"/>
    <state id="2" lowerBound="10.0" upperBound="Infinity"/>
  </states>
  <npt type="expression">
    <expression>Arithmetic(wmean(1, fixing_ppq_adj, 1,
fixing_oth_f_adj))</expression>
  </npt>
  <parentNodes>
    <parentNode name="fixing_oth_f_adj" />
    <parentNode name="fixing_ppq_adj" />
  </parentNodes>
  <childNodes>
    <childNode name="pot_fixed" />
    <childNode name="inserted" />
  </childNodes>
</node>
</object>

```

E.2 Defect Removal Model Manager

Tool description

Defect Removal Model Manager is a software tool which enables easier creation of the iterative models together with exporting the prediction results. It uses AgenaRisk API. The tool supports 3 types of iterative models:

1. multiple BN objects – each testing iteration is replicated as a separate BNO linked to the next BNO using the input/output nodes,
2. single BN object (sequential) – testing iterations are replicated in the same BNO, iterations are linked sequentially according to the input/output nodes defined in BNO ‘Main’ but these input nodes are removed during generating iterations,
3. single BN object (parameter learning) – testing iterations are replicated in the same BNO, certain nodes must contain the constants reflecting the number of iteration; no sequential links are created – rather the links between the parameter nodes (about to be learnt) and other nodes are created (according to definition in ‘Main’ BNO).

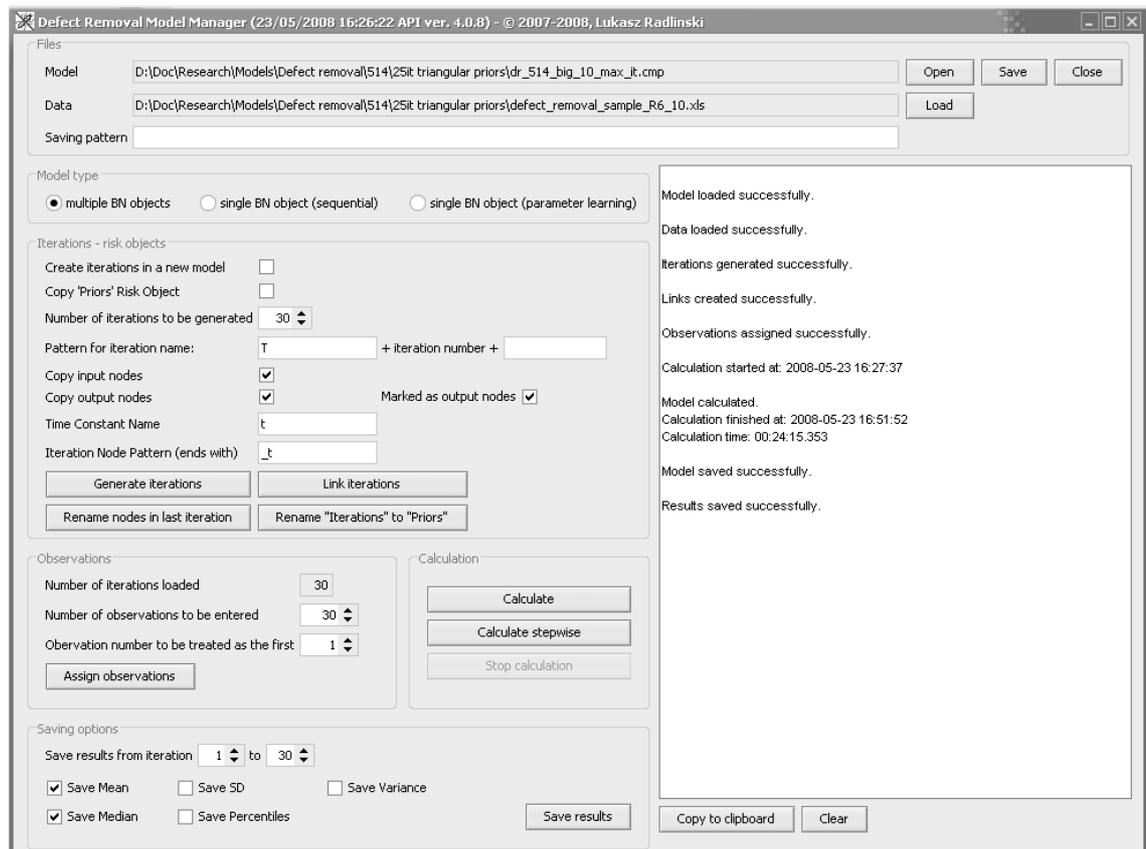


Figure E-1 Screenshot from the Defect Removal Model Manager

The following functions are performed after clicking specific button:

- ‘Open’ – opens AgenaRisk file which should contain 2 BNOs: ‘Priors’ – containing initialized values/probabilities for nodes which are linked between subsequent testing iterations, ‘Main’ – the main part of the model (single testing iteration).
- ‘Save’ – saves the current model.
- ‘Close’ – closes the current model.
- ‘Load’ – loads the dataset with observations which will later be assigned to specific nodes from MS Excel file.
- ‘Generate iterations’ – replicates the single instance of testing iterations number of times specified in relevant place according to the method selected in ‘Model type’.
- ‘Link iterations’ – creates links between testing instances BNOs according to the input/output nodes defined earlier in BNO ‘Main’ (only when model type is multiple BN objects).
- ‘Rename nodes in last iteration’ – renames nodes in the last testing iteration by removing the number of iteration from the node name; this enables creating more complex structure of model like: first n iterations as a single BNO (e.g. for learning) and next m iterations as multiple BNOs (e.g. for prediction); used jointly with button ‘Rename “Iterations” to “Priors”’.
- ‘Rename “Iterations” to “Priors”’ – renames BNO “Iterations” to “Priors” (only when model type initially is a single BNO) which contains nodes which can be treated as initialized nodes (already learned) for the next testing iterations.
- ‘Assign observations’ – assigns values earlier from MS Excel file to appropriate nodes as hard evidence.
- ‘Calculate’ – starts the model calculation process.
- ‘Calculate stepwise’ – starts the stepwise model calculation process; stepwise means that for certain nodes (defined earlier in MS Excel data file) the model performs prediction, reads the median predicted value and enters this value to this node as hard evidence, and then runs the calculation for the same testing iteration again (only when model type is multiple BN objects).
- ‘Stop calculation’ – stops the model calculation process (not yet implemented).

- ‘Save results’ – saves the selected statistics to a result MS Excel file for variables defined in the input MS Excel data file.
- ‘Copy to clipboard’ – copies the contents of the tool output console to the clipboard.
- ‘Clear’ – clears the output console.

Typical examples of using the tool in different scenarios:

Example 1. Building a multiple BNO model (DBN) containing 30 testing iterations together with saving results and the calculated model:

- button ‘Open’ – loading appropriate pattern of the model
- button ‘Load’ – loading appropriate dataset with observations
- ‘Number of iterations to be generated’: 30
- button ‘Generate iterations’
- button ‘Link iterations’
- button ‘Assign observations’
- button ‘Save’ – to save a model not yet calculated but generated according to the task and with observations assigned (optional stage)
- button ‘Calculate’
- button ‘Save results’
- button ‘Save’

Example 2. Building a model in which the first 5 iterations are linked together as a single BNO (for learning) and next 15 iterations are linked together as separate BNOs (with saving results and the model):

- button ‘Open’ – loading appropriate pattern of the model
- button ‘Load’ – loading appropriate dataset with observations
- option ‘single BN object (sequential)’
- ‘Number of iterations to be generated’: 5
- button ‘Generate iterations’
- ‘Number of observations to be entered’: 5
- button ‘Assign observations’
- button ‘Calculate’
- button ‘Save results’ – this saves the results only for the first 5 testing iterations

- button 'Rename nodes in last iteration'
- button 'Rename "Iterations" to "Priors"'
- option 'multiple BN objects'
- 'Number of iterations to be generated': 15
- button 'Generate iterations'
- button 'Link iterations'
- 'Number of observations to be entered': 5
- 'Observation number to be treated as first': 6 (5 used for learning + 1)
- button 'Assign observations'
- button 'Calculate'
- button 'Save results' – this saves the results only for iterations 6-20
- button 'Save'

E.3 Datasets used to validate the model

Table E-1 NASA MDP datasets for iterative testing (without process factors)

| Month | JM1 | | KC1 | | PC1 | | PC3 | | PC4 | |
|-------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|
| | Defects found | Defects fixed |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 3 | 0 | 1 | 0 |
| 2 | 3 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 3 | 35 | 0 | 0 | 0 | 2 | 0 | 1 | 0 | 25 | 0 |
| 4 | 29 | 0 | 1 | 0 | 3 | 0 | 8 | 0 | 10 | 2 |
| 5 | 39 | 0 | 3 | 0 | 8 | 0 | 5 | 0 | 1 | 0 |
| 6 | 82 | 0 | 1 | 0 | 0 | 0 | 7 | 0 | 0 | 0 |
| 7 | 45 | 0 | 4 | 0 | 0 | 0 | 13 | 1 | 2 | 2 |
| 8 | 62 | 0 | 3 | 0 | 1 | 0 | 19 | 0 | 1 | 0 |
| 9 | 34 | 0 | 4 | 2 | 1 | 0 | 6 | 0 | 0 | 0 |
| 10 | 36 | 0 | 0 | 0 | 0 | 11 | 6 | 5 | 1 | 0 |
| 11 | 26 | 0 | 0 | 0 | 9 | 0 | 14 | 0 | 32 | 0 |
| 12 | 40 | 0 | 1 | 0 | 0 | 0 | 20 | 0 | 1 | 25 |
| 13 | 36 | 0 | 0 | 0 | 0 | 0 | 45 | 19 | 16 | 1 |
| 14 | 39 | 0 | 7 | 0 | 18 | 0 | 89 | 4 | 13 | 8 |
| 15 | 43 | 0 | 6 | 0 | 2 | 0 | 86 | 59 | 23 | 2 |
| 16 | 29 | 0 | 5 | 0 | 6 | 2 | 67 | 4 | 39 | 4 |
| 17 | 28 | 0 | 2 | 0 | 13 | 0 | 64 | 23 | 34 | 0 |
| 18 | 41 | 0 | 10 | 0 | 51 | 0 | 40 | 28 | 42 | 5 |
| 19 | 27 | 0 | 6 | 9 | 23 | 9 | 37 | 49 | 41 | 14 |
| 20 | 29 | 0 | 24 | 1 | 29 | 25 | 36 | 17 | 42 | 7 |
| 21 | 27 | 0 | 22 | 1 | 20 | 73 | 42 | 33 | 44 | 40 |
| 22 | 11 | 0 | 55 | 5 | 33 | 21 | 69 | 43 | 95 | 12 |
| 23 | 22 | 0 | 24 | 54 | 37 | 18 | 25 | 27 | 46 | 41 |
| 24 | 18 | 0 | 29 | 12 | 29 | 33 | 39 | 125 | 21 | 30 |
| 25 | 14 | 0 | 22 | 7 | 10 | 0 | 19 | 51 | 40 | 151 |
| 26 | 17 | 0 | 24 | 73 | 22 | 26 | 32 | 41 | 59 | 82 |
| 27 | 16 | 0 | 27 | 27 | 17 | 15 | 12 | 51 | 45 | 93 |
| 28 | 22 | 0 | 10 | 20 | 8 | 53 | 27 | 35 | 37 | 22 |
| 29 | 16 | 0 | 22 | 11 | 10 | 9 | 18 | 27 | 30 | 124 |
| 30 | 35 | 0 | 26 | 7 | 26 | 20 | 15 | 11 | 23 | 34 |
| 31 | 15 | 0 | 12 | 16 | 5 | 0 | 23 | 4 | 32 | 36 |
| 32 | 23 | 0 | 33 | 14 | 26 | 31 | 29 | 50 | 23 | 47 |
| 33 | 11 | 0 | 16 | 11 | 9 | 0 | 11 | 5 | 36 | 1 |
| 34 | 3 | 0 | 17 | 17 | 16 | 0 | 6 | 2 | 30 | 40 |
| 35 | 14 | 0 | 22 | 12 | 25 | 21 | 11 | 9 | 6 | 1 |
| 36 | 31 | 0 | 10 | 37 | 10 | 42 | 18 | 25 | 12 | 1 |
| 37 | 11 | 0 | 12 | 8 | 13 | 18 | 17 | 2 | 17 | 38 |
| 38 | 13 | 0 | 14 | 13 | 6 | 0 | 19 | 37 | 7 | 4 |
| 39 | 18 | 0 | 10 | 6 | 8 | 0 | 33 | 9 | 5 | 0 |
| 40 | 8 | 0 | 13 | 17 | 12 | 0 | 14 | 9 | 3 | 0 |

| Month | JM1 | | KC1 | | PC1 | | PC3 | | PC4 | |
|-------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|
| | Defects found | Defects fixed |
| 41 | 8 | 0 | 9 | 29 | 10 | 11 | 11 | 19 | 6 | 20 |
| 42 | 24 | 0 | 6 | 16 | 23 | 55 | 14 | 32 | 1 | 31 |
| 43 | 17 | 0 | 5 | 6 | 6 | 0 | 13 | 0 | 5 | 15 |
| 44 | 25 | 0 | 8 | 2 | 8 | 0 | 21 | 25 | 5 | 1 |
| 45 | 36 | 1 | 6 | 6 | 5 | 22 | 22 | 13 | 0 | 4 |
| 46 | 19 | 0 | 22 | 13 | 5 | 0 | 28 | 7 | 4 | 0 |
| 47 | 15 | 0 | 11 | 15 | 0 | 0 | 24 | 12 | 2 | 0 |
| 48 | 15 | 0 | 8 | 8 | 0 | 0 | 13 | 31 | 6 | 0 |
| 49 | 15 | 1 | 24 | 15 | 3 | 3 | 10 | 15 | 6 | 6 |
| 50 | 19 | 0 | 23 | 13 | 0 | 0 | 24 | 10 | 10 | 4 |
| 51 | 23 | 1 | 28 | 10 | 5 | 0 | 19 | 57 | 1 | 0 |
| 52 | 9 | 0 | 30 | 3 | 0 | 0 | 6 | 61 | 0 | 3 |
| 53 | 13 | 1 | 30 | 34 | 0 | 0 | 6 | 32 | 0 | 0 |
| 54 | 7 | 0 | 11 | 10 | 0 | 0 | 2 | 32 | 0 | 0 |
| 55 | 7 | 113 | 18 | 4 | 0 | 0 | 1 | 15 | 2 | 0 |
| 56 | 12 | 1 | 14 | 35 | 0 | 0 | 3 | 10 | 0 | 0 |
| 57 | 8 | 0 | 13 | 33 | 0 | 0 | 2 | 4 | 0 | 0 |
| 58 | 7 | 1 | 23 | 24 | 0 | 0 | 2 | 1 | 0 | 0 |
| 59 | 12 | 0 | 18 | 33 | 1 | 10 | 4 | 7 | 1 | 0 |
| 60 | 8 | 73 | 14 | 20 | | | 2 | 5 | 0 | 0 |
| 61 | 22 | 1 | 7 | 6 | | | | | 1 | 0 |
| 62 | 28 | 1 | 17 | 10 | | | | | 0 | 0 |
| 63 | 22 | 97 | 16 | 4 | | | | | 0 | 0 |
| 64 | 21 | 0 | 10 | 21 | | | | | 0 | 0 |
| 65 | 11 | 2 | 16 | 20 | | | | | 0 | 0 |
| 66 | 13 | 3 | 8 | 11 | | | | | 0 | 0 |
| 67 | 13 | 2 | 5 | 24 | | | | | 1 | 0 |
| 68 | 10 | 0 | 12 | 12 | | | | | | |
| 69 | 7 | 2 | 7 | 16 | | | | | | |
| 70 | 12 | 39 | 9 | 23 | | | | | | |
| 71 | 4 | 0 | 13 | 11 | | | | | | |
| 72 | 11 | 0 | 4 | 20 | | | | | | |
| 73 | 16 | 1 | 8 | 8 | | | | | | |
| 74 | 9 | 0 | 8 | 9 | | | | | | |
| 75 | 11 | 126 | 17 | 21 | | | | | | |
| 76 | 14 | 1 | 7 | 8 | | | | | | |
| 77 | 14 | 0 | 9 | 7 | | | | | | |
| 78 | 6 | 10 | 5 | 12 | | | | | | |
| 79 | 6 | 25 | 1 | 6 | | | | | | |
| 80 | 6 | 0 | 1 | 28 | | | | | | |
| 81 | 8 | 0 | 0 | 3 | | | | | | |
| 82 | 6 | 0 | 2 | 2 | | | | | | |
| 83 | 11 | 0 | 0 | 5 | | | | | | |
| 84 | 5 | 0 | 0 | 0 | | | | | | |

| Month | JM1 | | KC1 | | PC1 | | PC3 | | PC4 | |
|-------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|
| | Defects found | Defects fixed |
| 85 | 5 | 0 | 0 | 0 | | | | | | |
| 86 | 3 | 0 | 0 | 0 | | | | | | |
| 87 | 4 | 0 | 0 | 1 | | | | | | |
| 88 | 3 | 0 | | | | | | | | |
| 89 | 2 | 0 | | | | | | | | |
| 90 | 1 | 0 | | | | | | | | |
| 91 | 2 | 46 | | | | | | | | |
| 92 | 1 | 0 | | | | | | | | |
| 93 | 4 | 15 | | | | | | | | |
| 94 | 0 | 0 | | | | | | | | |
| 95 | 0 | 0 | | | | | | | | |
| 96 | 0 | 0 | | | | | | | | |
| 97 | 0 | 2 | | | | | | | | |
| 98 | 0 | 17 | | | | | | | | |

Table E-2 Semi-randomly generated dataset for iterative testing only (with process factors)

| Iteration number | Testing effort | Testing process and people quality | Testing bias | Defects found | Iteration number | Testing effort | Testing process and people quality | Testing bias | Defects found |
|------------------|----------------|------------------------------------|--------------|---------------|------------------|----------------|------------------------------------|--------------|---------------|
| 1 | 1.00 | 1.00 | 1.00 | 60 | 26 | 0.79 | 1.20 | 1.18 | 84 |
| 2 | 0.87 | 1.30 | 1.02 | 50 | 27 | 1.09 | 0.93 | 1.00 | 81 |
| 3 | 1.22 | 0.84 | 0.79 | 71 | 28 | 0.83 | 0.95 | 1.11 | 57 |
| 4 | 0.77 | 0.85 | 1.05 | 56 | 29 | 1.05 | 0.69 | 0.91 | 60 |
| 5 | 1.24 | 1.20 | 1.22 | 55 | 30 | 0.87 | 1.36 | 0.99 | 47 |
| 6 | 1.18 | 1.07 | 0.76 | 82 | 31 | 0.98 | 1.28 | 1.07 | 40 |
| 7 | 1.00 | 0.98 | 1.05 | 76 | 32 | 0.84 | 1.02 | 1.00 | 32 |
| 8 | 1.12 | 1.27 | 0.84 | 98 | 33 | 0.88 | 1.10 | 0.87 | 31 |
| 9 | 1.05 | 1.27 | 1.06 | 94 | 34 | 0.80 | 1.03 | 0.94 | 26 |
| 10 | 0.85 | 1.38 | 1.06 | 71 | 35 | 0.86 | 0.82 | 1.03 | 22 |
| 11 | 1.13 | 1.03 | 1.19 | 64 | 36 | 0.85 | 1.26 | 0.96 | 18 |
| 12 | 1.01 | 0.80 | 0.82 | 76 | 37 | 0.96 | 1.33 | 1.09 | 15 |
| 13 | 0.92 | 1.22 | 0.92 | 74 | 38 | 1.34 | 1.00 | 1.31 | 14 |
| 14 | 1.08 | 1.29 | 0.96 | 80 | 39 | 0.79 | 1.01 | 0.99 | 12 |
| 15 | 1.07 | 0.80 | 0.90 | 90 | 40 | 1.08 | 1.30 | 1.18 | 10 |
| 16 | 0.90 | 1.31 | 1.19 | 65 | 41 | 0.82 | 1.20 | 1.00 | 8 |
| 17 | 1.09 | 0.87 | 0.92 | 72 | 42 | 0.92 | 0.84 | 1.03 | 7 |
| 18 | 0.75 | 1.30 | 1.04 | 49 | 43 | 1.14 | 1.38 | 1.17 | 7 |
| 19 | 1.34 | 1.08 | 0.71 | 87 | 44 | 1.00 | 0.79 | 1.02 | 7 |
| 20 | 1.15 | 1.00 | 1.15 | 81 | 45 | 1.03 | 1.07 | 1.19 | 6 |
| 21 | 1.18 | 1.13 | 0.73 | 122 | 46 | 0.91 | 0.94 | 1.19 | 4 |
| 22 | 0.87 | 1.04 | 0.82 | 122 | 47 | 1.00 | 1.27 | 0.92 | 5 |
| 23 | 0.90 | 0.70 | 0.72 | 147 | 48 | 1.07 | 1.26 | 1.01 | 5 |
| 24 | 0.95 | 1.03 | 1.10 | 113 | 49 | 1.00 | 1.09 | 0.92 | 5 |
| 25 | 1.35 | 1.03 | 0.92 | 143 | 50 | 0.81 | 0.76 | 0.72 | 6 |

Table E-3 Semi-randomly generated dataset for testing and fixing defects (with process factors)

| Iteration number | Testing effort | Testing process and people quality | Testing other factors | Fixing effort | Fixing process and people quality | Fixing other factors | Defects found | Defects fixed | Defects inserted |
|------------------|----------------|------------------------------------|-----------------------|---------------|-----------------------------------|----------------------|---------------|---------------|------------------|
| 1 | 73 | Very Low | Low | 65 | Very Low | Low | 16 | 9 | 3 |
| 2 | 70 | Lowest | Medium | 55 | Low | Very Low | 11 | 9 | 2 |
| 3 | 80 | Low | High | 77 | Low | Low | 43 | 14 | 3 |
| 4 | 47 | Very Low | Low | 62 | Medium | Low | 8 | 14 | 2 |
| 5 | 64 | Medium | Low | 51 | Medium | High | 29 | 14 | 1 |
| 6 | 77 | Low | Low | 57 | High | Very Low | 19 | 15 | 1 |
| 7 | 73 | High | High | 42 | Low | Medium | 71 | 8 | 1 |
| 8 | 48 | Medium | Medium | 83 | High | Medium | 14 | 27 | 2 |
| 9 | 64 | High | High | 73 | Low | Low | 35 | 13 | 2 |
| 10 | 66 | Medium | Very High | 59 | Low | High | 19 | 13 | 2 |
| 11 | 55 | Low | Medium | 51 | Medium | Low | 5 | 12 | 1 |
| 12 | 60 | Medium | Very Low | 54 | Medium | Medium | 5 | 14 | 1 |
| 13 | 73 | Medium | High | 41 | High | Very High | 11 | 16 | 1 |
| 14 | 56 | High | Medium | 47 | High | Low | 8 | 14 | 1 |
| 15 | 45 | Medium | Medium | 59 | Medium | High | 3 | 16 | 1 |
| 16 | 55 | Very High | Medium | 61 | Medium | Medium | 10 | 15 | 2 |
| 17 | 68 | Medium | Low | 59 | Medium | High | 3 | 16 | 1 |
| 18 | 66 | High | Very High | 63 | High | Low | 8 | 18 | 1 |
| 19 | 68 | Medium | High | 57 | Low | High | 2 | 12 | 2 |
| 20 | 61 | High | Very Low | 59 | High | Medium | 2 | 19 | 1 |
| 21 | 48 | Medium | Medium | 51 | Highest | Low | 1 | 24 | 1 |
| 22 | 56 | Low | Low | 61 | Medium | Medium | 1 | 12 | 1 |
| 23 | 61 | Medium | Medium | 55 | Very High | Medium | 1 | 1 | 0 |
| 24 | 76 | Highest | Low | 63 | Medium | Low | 5 | 5 | 1 |
| 25 | 42 | Medium | Low | 55 | Very High | High | 0 | 0 | 0 |
| 26 | 69 | High | Low | 44 | Medium | Medium | 1 | 1 | 0 |
| 27 | 58 | Highest | High | 49 | Highest | Lowest | 2 | 2 | 0 |
| 28 | 64 | High | Lowest | 67 | Medium | High | 0 | 0 | 0 |
| 29 | 58 | High | Very High | 60 | Medium | High | 0 | 0 | 0 |
| 30 | 70 | Highest | Medium | 63 | Highest | Medium | 1 | 1 | 0 |