Defect Cost Flow Model – A Bayesian Network for Predicting Defect Correction Effort

Thomas Schulz
Robert Bosch GmbH
CC/ESR2, Postfach 16 61
71226 Leonberg, Germany
Tel. +49 711 811-47452
t.schulz@de.bosch.com

Łukasz Radliński
University of Szczecin
ul. Mickiewicza 64
71-101 Szczecin, Poland
Tel. +48 91 444-1911
lukrad@uoo.univ.szczecin.pl

Thomas Gorges
Robert Bosch GmbH
CC/ESR2, Postfach 16 61
71226 Leonberg, Germany
Tel. +49 711 811-47679
thomas.gorges@de.bosch.com

Wolfgang Rosenstiel
Eberhard Karls Universität Tübingen
Sand 13, B 207
72076 Tübingen, Germany
Tel. +49 7071 29-75482
rosenstiel@informatik.uni-tuebingen.de

ABSTRACT

Background. Software defect prediction has been one of the central topics of software engineering. Predicted defect counts have been used mainly to assess software quality and estimate the defect correction effort (DCE). However, in many cases these defect counts are not good indicators for DCE. Therefore, in this study DCE has been modeled from a different perspective. Defects originating from various development phases have different impact on the overall DCE, especially defects shifting from one phase to another. To reduce the DCE of a software product it is important to assess every development phase along with its specific characteristics and focus on the shift of defects over phases.

Aims. The aim of this paper is to build a model for effort prediction at different development stages. Our model is mainly focused on a dynamic DCE changing from one development phase to another. It reflects the increasing cost of correcting defects which are introduced in early, but found in later development phases.

Research Method. The modeling technique used in this study is a Bayesian network which, among many others, has three important capabilities: reflecting causal relationships, combining expert knowledge with empirical data and incorporating uncertainty. The procedure of model development contains a set of iterations including the following steps: problem analysis, data analysis, model enhancement with simulation runs and model validation.

Results. The developed Defect Cost Flow Model (DCFM) reflects the widely used V-model, an international standard for developing information technology systems. It has been pre-calibrated with empirical data from past projects developed at Robert Bosch GmbH. The analysis of evaluation scenarios confirms that DCFM correctly incorporates known qualitative and quantitative relationships. Because of its causal structure it can be used intuitively by end-users.

Conclusion. Typical cost benefit optimization strategies regarding the optimal effort spent on quality measures tend to optimize locally, e.g. every development phase is optimized separately in its own domain. In contrast to that, the DCFM demonstrates that even cost intensive quality measures pay off when the overall DCE of specific features is considered.

Categories and Subject Descriptors

D.2.9 [Software Engineering]: Management; G.3 [Probability and statistics] – Probabilistic algorithms (including Monte Carlo); H.4.2 [Information Systems Applications]: Types of Systems – Decision Support (e.g., MIS); I.6.0 [Simulation and Modeling]: General

General Terms

Experimentation, Management, Measurement, Performance

Keywords

Bayesian network, correction effort, decision support, defect flow, process modeling, software process

1. INTRODUCTION

For the next years, experts estimate the global readmission rate for cars to be over 70.000.000 per year [38]. At the same time, the European Commission requests a reduction of traffic deaths by 50% in the same period. 70% of future innovation will be based on software. Therefore, with the increasing number of cars, software becomes more and more important. Coming along with this, in recent years 50% of all vehicle recalls were software related.

One major challenge in the development of large scale software products for the automotive industry is to optimize *quality assurance* (QA) over product costs to develop high quality products at low in-field defect rates and still at low costs. Short development life cycles stand in contrast to long product life cycles, a fact which confirms the demand for failsafe innovative products.

In an earlier study we proposed the *Software Process Model* (SPM) [27] to describe the influence of changes on a software product regarding the potential *defect correction effort* (DCE) remaining after development. SPM defines a *defect cost factor* (DCF) representing the error-proneness for a specific feature of the software product. SPM's main purpose is the estimation of DCE based on multiple change characteristics of a specific software release with focus on changes since this is the only way of bringing defects into the software product.

We recognized a different impact on the overall DCE for defects originating from other development phases than they were detected in, compared to defects detected in the same development phases in which they were made. Therefore, to reduce the DCE of the software product, we assess every development phase along with its specific DCE characteristic and focus on the shift of defects through development phases.

We realized these ideas in the completely new *Defect Cost Flow Model* (DCFM) which is the main contribution of this paper. The main aim of the DCFM is the identification of process areas where optimization leads to lowest defect rates and lowest cost. The idea of a *defect flow model* (DFM) had already been established before DCFM but with a different focus. The DCFM enables to estimate the DCE based on *key performance indicators* (KPIs) representing product, process and project specifics. With the DCFM it is possible to assess effort spent on defect correction in comparison to effort spent on development throughout every phase of the development process. Besides this, the DCFM focuses on a single high level function of the software product.

Formally, the DCFM is a *Bayesian Network* (BN) which incorporates both process data as well as expert knowledge within a single model. The DCFM reflects a real engineering process for the development of embedded applications in the automotive industry. Historical and current process data as well as expert knowledge have been used for model calibration.

The following section gives *background* information on the DCFM, about the domain where it has been developed, about underlying concepts as well as the technologies used for its development. The background section is followed by the *research method*. It describes the procedure model we used for DCFM's development. After this, the main section *Defect Cost Flow Model* describes DCFM in detail. The simulation results of several scenarios are analyzed to demonstrate the capabilities of DCFM. Finally, the *conclusion* section summarizes results and describes what further steps could be in the domain of DCFM.

2. BACKGROUND

2.1 Automotive Software Engineering

The demand for software products in the automotive industry has been exponentially increasing over the last decades, making software engineering one of its critical success factors [16]. The intensive use of *electronic control units* (ECUs) in vehicles today has already led to an average of 20 ECUs in smaller and even more than 70 ECUs in upper class cars [38]. Complex features with short time to market and minimal engineering costs at very high quality are the challenges to be met.

Furthermore, embedded software is subject to domain specific characteristics, especially:

- *code efficiency* due to limitations of processor and memory resources,
- *code portability* to support the product line approach for variant handling,
- *code understandability* and *maintainability* to enable multiple, distributed engineering teams,
- *high availability* and *reliability* to support safety critical applications,
- *real-time operation*,
- *network operation* to support distributed systems.

Given these aspects, effort estimation including the effort needed for defect correction already plays a major role in the domain of automotive software engineering.

2.1.1 Process Model

Standardized engineering processes are crucial to continuously improve and adapt to the market's demands. The engineering process behind the model presented here is based on the *V-Model* [33], an international standard for developing information technology systems. Inside the automotive industry, the V-Model has been enhanced for decades to reflect the needs for development of high quality failsafe products. It describes different development phases in a software release life cycle focusing on high level system specification and testing, as well as on module implementation and testing. Figure 1 illustrates the V-Model.

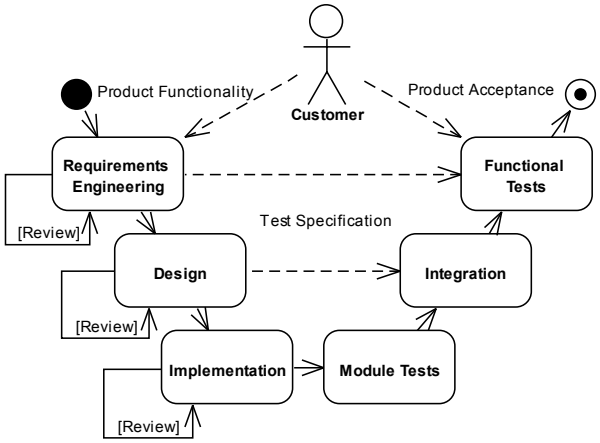

Figure 1 V-Model development phases, based on [33]

The customer's wishes for *product functionality* are the starting point of every software release life cycle. New features, changes or removal of existing functionality are handled in the phase *requirements engineering* (RE) where functions are classified according to specific feature categories, e.g. all configuration options a driver has with his ECU are bundled in a feature called *Human Machine Interface* (HMI).

The specification of these features is the responsibility of specific engineering teams, i.e. feature domain experts. In the phase *design* (DE), features are allocated to software components. These components contain software modules building the code base for the software product. The succeeding phase *implementation* (IM) is responsible for developing the code base with the help of artifacts from RE and DE. The software product we have been working on consists of approximately 50 features, 15 software components and 100 software modules. After every phase RE, DE and IM, all engineering artifacts (requirements, design specs or source code) are subject to phase specific QA measures or reviews respectively.

The V-Model's right hand side is related to QA measures only, mainly to *integration and testing* (I&T) based on *test specifications* from phases RE, DE and IM. *Module tests* are performed after implementation. These modules are integrated into software components whereas components are integrated into the overall software product in the so called *integration* phase. Next, the software is integrated into the ECU to perform high level *functional tests* forming the base for reviews and tests carried out by system engineers. Finally, the product has to pass *product acceptance* tests by the *customer* (CU).

2.1.2 Defect Cost Factor

Defect costs can be derived from the rework needed to complete a product. Rework is necessary where released features are corrected because they do not meet their requirements. The amount of rework often is expressed as a number of defects but this only is an indicator for the actual rework spent to fix product requirement deviations. As a single measure it is incapable of quantifying the amount of rework, e.g. a simple defect might be fixed in hours whereas another defect might take days of analyzing and fixing. Furthermore, defects detected in later development phases, e.g. in I&T, resulting in a change in early phases, e.g. RE or DE, are more cost intensive than defects detected in development phases where they are made.

The DCFM uses the potential *defect correction effort* as one indicator to describe a product's defect rate. A second indicator, the *development effort*, is needed as reference value for the DCE to indicate its impact, e.g. 10 hours of DCE for a feature with 1000 hours of development effort represents a low defect rate whereas 10 hours of DCE for a feature with only 5 hours of development indicates a very high defect rate. This leads to the definition of the *defect cost factor* (DCF) representing DCFM's indicator for defect rate:

$$DCF = \frac{defect\ correction\ effort}{development\ effort}$$

Where:

- *Defect correction effort* is the effort used to fix a defective implementation.
- *Development effort* represents the effort needed for implementation.

This implies, with increasing development effort the potential effort needed for later fixing increases proportionally. We define a DCF for every product feature because features are different in many aspects, e.g. concerning their requirement complexity and volatility, resulting code complexity and finally their testability. For instance, the probability of creating a defect is lower for implementing parameter changes than for a complex state machine used to realize HMI logic.

2.1.3 Defect Flow Model

Our DCFM is based on the DFM, a measurement system supporting the quantitative evaluation of QA measures in an engineering process [31]. DFM is derived from the orthogonal defect classification concept for process measurements [25]. Its main goal is to provide transparency on phase specific defect rates.

Defects are represented based on where they are created in relation to where they are found. Based on the DFM, analysis for every development phase can be assessed separately to identify phases where QA measures are to be focused on. The DFM uses the number of defects as the indicator for the performance of a development phase.

Figure 2 illustrates a DFM showing DCFM's phases RE, DE, IM and I&T. It depicts 55 defects made and 40 defects detected in the development phase RE resulting in 15 residual defects after this phase. It is possible that these defects will be detected in later phases, e.g. in phase DE. Finally, phase CU illustrates 33 product defects left, probably detected by the customer.

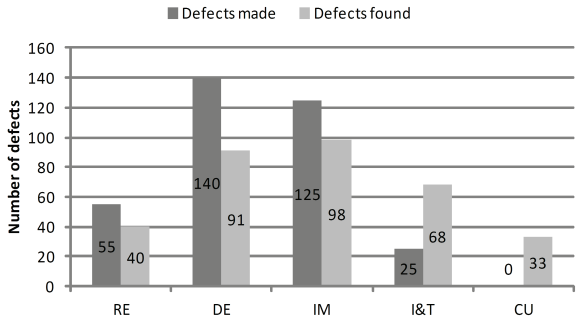

Figure 2 Defect Flow Model

According to [31] the DFM has proven its capability to monitor and improve quality processes in the domain of software development for automotive applications.

2.2 Bayesian Networks

It is not possible to build a predictive model reflecting the defect cost flow automatically from data because the datasets of required volume and diversity do not exist in practice. Thus, building such models involves combining domain expert knowledge and empirical data. While there are different modeling techniques satisfying this condition, we selected BNs for the following other important advantages: explicit incorporation of uncertainty, ability to reflect causal relationships, intuitiveness through a graphical representation, ability of both forward and backward reasoning, ability to run obtain predictions from incomplete data.

The BN [5], [14], [22] is a probabilistic model, which has two perspectives: graphical and numeric. Graphically, it is a directed acyclic graph consisting of a set of nodes and arcs between pairs of nodes. Numerically, each node is defined as a conditional probability distribution given the states of its parents (immediate predecessors).

Thus in a BN consisting of variables *X1*, …, *Xn*, the joint probability distribution is defined as:

$$P(X_1,...,X_n) = \prod_{i=1}^{n} P(X_i \mid parents(X_i))$$

BN concepts (and the term itself) were introduced in the 1980's in pioneering work by Pearl [21], [22]. Since then it has now become a well established modeling technique, which has been applied in such diverse fields as: medicine, biology, chemistry, physics, law, management, computer science and other.

BNs in software engineering area have been used mainly for effort and quality prediction. Since very few of them cover similar issues as in our study, we only list most recent BN studies in Table 1. Fenton et al. [10] analyzed how the effectiveness of specification, coding and testing influences the defect potential, inserted and found, respectively. At the end, the model predicts the number of residual defects. However, it assumes that effort required to fix a defect is constant – i.e. that it does not depend on the defect itself. Such an assumption may work only when proportions of different types of defects are constant among multiple projects and components.

Bibi and Stamelos [2] proposed a model for development effort prediction in projects compatible with the Rational Unified Process. In their model effort is estimated at various project stages and for different activities, and then aggregated. This concept seems to be clear and intuitive. However, the authors have not performed any validation and only published a basic topology of the model. Thus, a detailed analysis of their model is difficult.

Table 1 Summary of recent BNs in software engineering

Source	Main problem investigated	Source	Main problem investigated
[38]	change coupling	[6]	maintenance delays
[3]	defect rate	[7]	need for requirement review
[8]	defects	[12]	project velocity
[10], [11]	defects	[9]	size, effort, quality
[2]	effort	[23], [24]	size, effort, quality
[27]	effort	[36]	testing process
[30]	effort, productivity	[33]	various aspects of software quality
[31]	maintainability	[15], [19]	web development effort

3. RESEARCH METHOD

3.1.1 Statement of the Problem

We created the DCFM based on the guidelines to designing expert systems [36]. Its main application is the support of project managers and process owners on complex decisions related to DCE estimation.

Figure 3 illustrates the procedure we used to build the model.

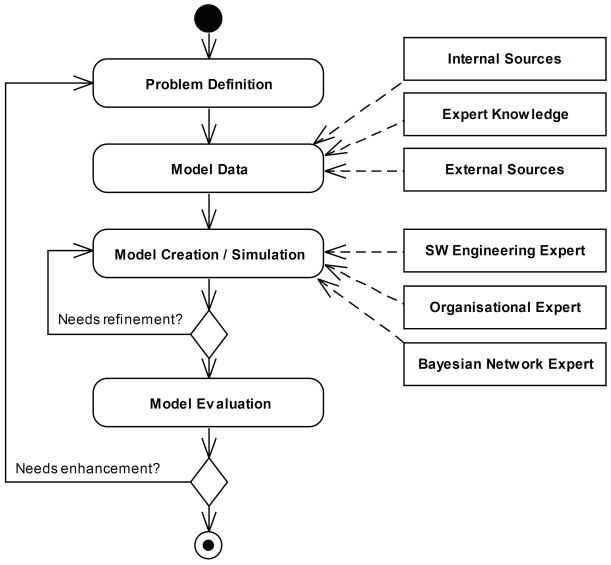

Figure 3 Procedure model

Essential to every expert system is the *problem definition* and KPIs related to it. It is derived from the initial goal of the model. We used the *Goal Question Metric* (GQM) approach [1] to systematically identify relevant KPIs of the model and their relation. According to GQM, the problem definition includes its purpose, object of interest, issue and user's point of view leading to the following definition:

For a specific engineering environment the DCFM shall identify the ideal distribution of QA effort to minimize rework at given time and costs from a project manager's point of view.

Based on the problem definition we identified the KPIs and their relations, and used them in a causal structure of the model.

3.1.2 Model Data

We obtained the data for model definition mostly from internal sources within the specific automotive engineering environment. We analyzed data from a completed project and extracted relevant metrics according to the data definition illustrated in Figure 4. Our main data source was the *change & defect management* (CM) system holding information about the development of artifacts needed to realize a specific part of a feature.

An entry in the CM has one of the following categories:

- C*hange* entry describes the development of a newly added, modified or removed requirement. It holds information about the *development effort* needed for this change.
- *Defect Analysis* entry describes a defect in an existing feature. It stores the information about the *analyzing effort*, where the defect has been detected (*origin phase*) and in what phase it has been corrected (*correction phase*).
- *Defect Correction* describes what changed to solve the defect. It stores information about the *correction effort*.

Every CM entry is related to a *feature* in the *requirements management* (RM) system. The RM system holds information about the complexity and volatility of a specific feature. The DCE and other model parameters are defined based on these data from CM and RM.

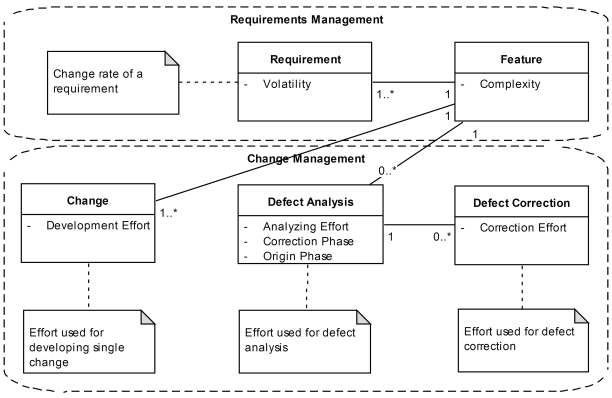

Figure 4 Project data definition

Further internal sources are process documentation as well as expert knowledge. Data used for model scenarios has been made anonymous due to their confidentiality.

3.1.3 Model Creation and Simulation

Model creation and simulation is a very challenging task because expertise is required in multiple fields. First, a deep understanding of the model's domain has to be established, software engineering in case of DCFM. Second, the problem under discussion has to be fully understood including its KPIs and how they are related to each other. For DCFM, it means being an expert in the specific automotive work environment able to map process and project data to the problem definition and its KPIs. Finally, statistical know-how is required e.g. to understand the consequences of combining data from various sources and how it affects the success of simulation results. We needed several iterations to design and calibrate the DCFM to fulfill all requirements from the fields above.

Model structure and calibration is based on the automotive engineering process described before. Here, we collected historical and current data according to the project data definition. The DCFM was set up based on this data. Furthermore, domain experts calibrated those parts of the model where no project data was available. To evaluate the model performance, several scenarios were defined to reflect the different aspects of the problem definition. The final structure of the DCFM is based on an iterative refinement with the goal to optimize the performance for all scenarios.

3.1.4 Model Evaluation

The final step creating DCFM has been *model evaluation*. Due to the novelty of our approach, it was not possible to validate all simulation scenarios. Instead, domain experts analyzed and evaluated the model according to the definition of the problem.

The final BN model, including calibration and predefined scenarios, is available for public use from the PROMISE online repository [28].

4. DEFECT COST FLOW MODEL

4.1 Flow of defects

Based on the procedure model from the previous section, there first will be an introduction to the concept behind DCFM. It focuses on the problem definition of assessing every phase of a development process to identify the ideal amount of QA effort spent in every phase. Model data in form of the KPIs of the DCFM and their relations are described in form of a metric definition table where every metric is derived based on a set of questions describing the DCFM's principal aim, e.g.

- How much development effort is spent?
- How error-prone is the development?
- How much effort is spent on QA measures?
- How effective are these QA measures?
- What is the effort involved for defects shifting from one phase to another?

The DCFM is built as a BN based on these parameters and their relation among each other. Model assessment focuses on the overall amount of effort, development effort and rework as well as on QA effort for the features developed. For final comparison several scenarios are defined where KPIs are varied according to different alternatives regarding the distribution of QA effort over development phases, e.g. one scenario focuses on the elimination of all defects just before customer delivery whereas another scenario focuses the maximum amount of QA effort spent in early phases.

The *concept of flowing defects* is illustrated in Figure 5. There are four development phases built into DCFM according to the V-Model process definition: RE, DE, IM and I&T. Final phase CU is calculated based on results from its previous phase I&T. For every development phase, the DCE is determined separately based on process specific KPIs. The *DCE after QA* originating in RE flows to its succeeding phase after it is adjusted by its *phase multiplier*. This phase multiplier represents the increase of effort if defects are not corrected in the same phase where they are injected. The phase multiplier varies from company to company and depends on people involved over different phases. The more process activities involved, the higher the DCE for defects flowing from one phase to another.

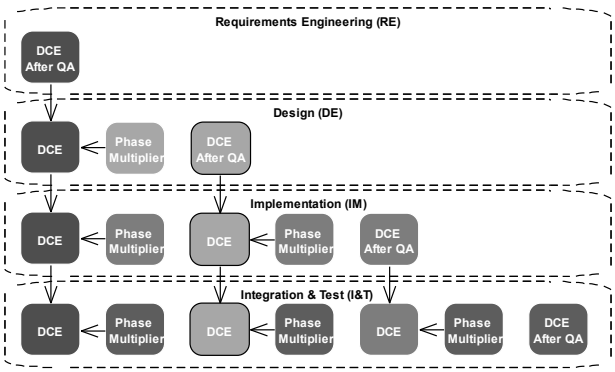

Figure 5 DCFM concept

Similar to the DCE injected in RE, it is handled in phases DE, IM and I&T. For every phase, there is a corresponding DCE flowing from one phase to another, processed based on phase specific KPIs. Details about how this concept is used to build DCFM are shown later in section 4.3.

4.2 Development of costs

The *development of costs* in DCFM is shown in Figure 6, focusing on the flow of DCE over development phases. Defects are injected in their corresponding phase and detected in later phases. In DCFM, defect correction costs are represented by the DCE. Positive axis illustrates effort spent on defect correction whereas negative axis depicts the reduced DCE after QA measures.

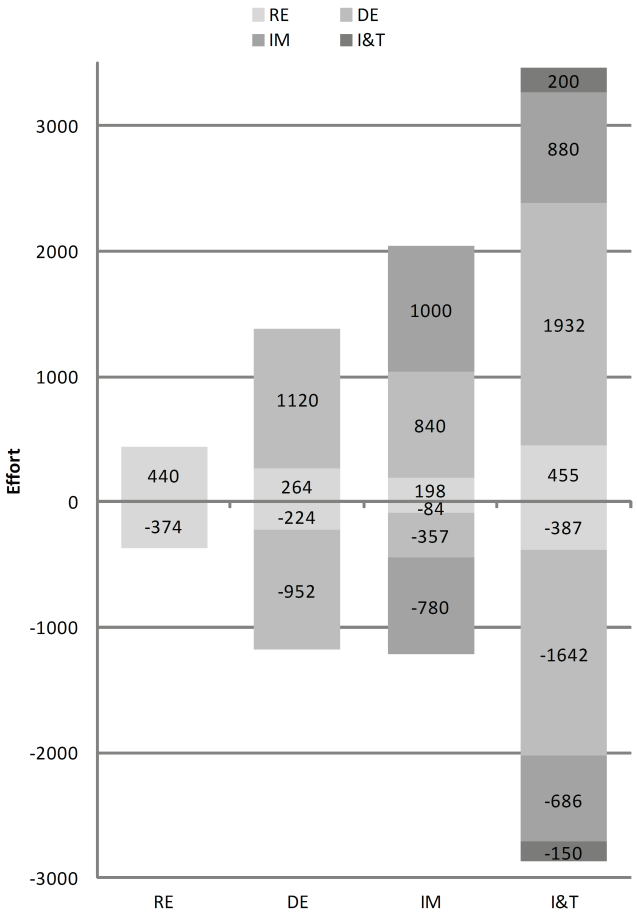

Figure 6 Defect Cost Flow

Focusing on defects with origin in phase RE, there are 440 hours of potential DCE residing in the software product. With QA measures and a very high detection rate of 85%, the DCE could be reduced by 374 hours leading to a total DCE of 66 hours for phase RE. These 66 hours stay undetected and flow from phase RE to DE.

In the DE phase they are adjusted by the phase multiplier for RE to DE where DCFM is calibrated with a value of 4. Following the flow of DCE injected in phase RE, 264 hours are detected by QA measures in phase DE and reduced by 224 to around 40 hours. Further adjusted by the phase multiplier (5 from DE to IM), 198 hours of DCE injected originally in phase RE flow from DE to IM. In phase IM it is much more difficult to detect defects from RE whereby only 43% of DCE are detected. Finally, the DCE flows to phase I&T (adjusted by a phase multiplier of 4) where 455 hours are reduced by 387 hours to a final DCE of 68 hours residing after the last phase and potentially detected by the customer.

Summing up for phase RE, there is an initial DCE of 440 versus an overall of 763 hours resulting from undetected RE defects. In every development phase there is a correspondent DCE to it, either reduced by specific QA measures or flowing to further phases. The DCFM shows that even with very high defect detection rates, for every undetected defect flowing from one phase to another, it causes a multiple of correction effort than either QA measures had prevented the defect from being made or at least these measures had detected it in the same phase where it was injected.

4.3 BN Model

The aim of our model is to identify the ideal amount of effort spent on QA measures per development phase to reduce the overall engineering effort. The identification of KPIs and their relations is the first step in creating such a model. According to our research method, the systematic approach to identify these KPIs is the prior definition of measurement categories. Every measurement category contains a set of specific metrics, actual measures to characterize a category. Measurement categories are defined with the help of questions you might ask to understand the statement of the problem. With the help of these questions it is possible to systematically identify relevant KPIs and characterize the goal of model. Table 2 illustrates the set of questions enabling the identification of all major components of the final model. For every question, a set of metrics is described to quantify it.

The DCFM as BN is depicted schematically in Figure 7. It represents a part of the actual DCFM. The complete BN model can be obtained from the PROMISE online repository [28]. It has four development phases implemented: RE, DE, IM and I&T whereas in the schematic there are only two phases: RE and DE. Figure 7 illustrates the BN as cause and effect chain where nodes represent events and arcs their relation. Calibration nodes are used for setting up the model.

In every phase, there is a specific development effort and defect cost factor resulting in the potential DCE as an indicator for a phase's error-proneness. These nodes take into account that for specific features, e.g. a simple parameter database, it might not be necessary to put the maximum amount of QA effort into defect detection because its initial defect rates are already very low. Furthermore, not every development phase has the same defect rate. Especially later phases have lower DCFs than e.g. RE or DE.

Table 2 Metric definition

Question	Metric	
	Name	**Description**
How are development costs expressed in the model?	development effort	Effort (in hours) spent on artifact development in RE, DE, IM and I&T excluding QA and correction effort.
How is the benefit / effort of QA determined?	amount of QA effort	Percentage of development effort used as QA effort.
	QA effort	Effort (in hours) spent on QA measures after development.
	defect correction effort reduced	Reduction of defect correction effort based on the amount of QA effort spent.
	defect correction effort after QA	Difference between defect correction effort and defect correction effort after QA.
How is the defect correction effort determined?	defect correction factor	Factor indicating a feature's error-proneness.
	defect correction effort	Potential effort (in hours) spent on correcting defects. Multiplication of defect correction factor and development effort.
How is the overall engineering effort determined?	engineering effort	Sum of development effort, QA effort and defect correction effort.
How is the increase of effort taken into account when defects shift from one phase to another?	phase multiplier	Company specific factor representing e.g. more people involved, additional process effort if defects need to be corrected over multiple development phases.

The DCFM is calibrated using phase specific DCFs illustrated in Table 3. These values are taken based on the assumption to develop a complex feature, e.g. a HMI. Other features with lower complexity have lower DCFs.

Table 3 DCFs per development phase

RE	DE	IM	I&T
1	0.8	0.7	0.01

In every development phase, both nodes *development effort* and *defect cost factor* are combined in a multiplication node to *defect correction effort.* The nodes *defect detection potential* and *amount of QA effort* represent the effectiveness of all QA activities for specific development phases.

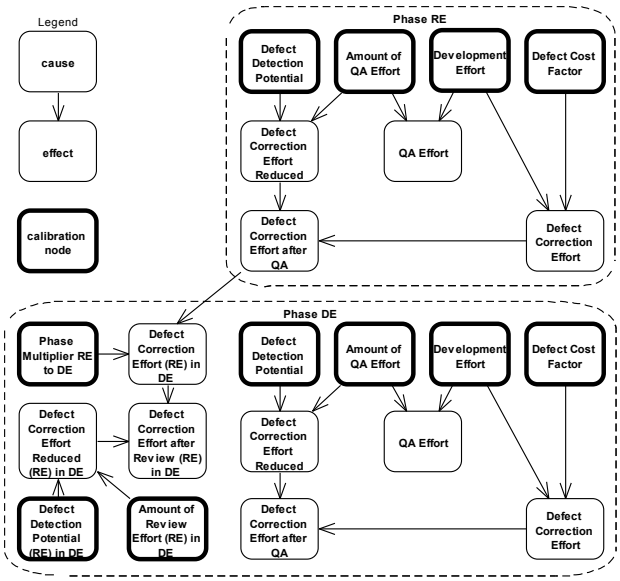

Figure 7 Bayesian Net DCFM

The defect detection potential enables to define detection rates of QA activities between 0% and 100%. The DCFM uses ranked nodes for their representation with the possibility to define the most relevant values, for a realistic scenario from a project manager's perspective. Four different ranks are defined:

- *Low* represents a worst case scenario.
- *Medium* is used in an average scenario.
- *High* represents ideal conditions for a scenario.
- *Very High* is used for the best case scenario.

Table 4 illustrates the review effort $E_R(E_D)$ for every *rank* from *low*, *medium*, *high* to *very high*. It is represented as percentage of the *development effort* E_D, e.g. if 1000 hours are planned for the development of a feature, an $E_R(E_D)$ of 10% represents 100 hours of additional review effort.

Table 4 Amount of QA effort

Rank	$E_R(E_D)$
Low	5%
Medium	10%
High	20%
Very High	40%

Node *defect detection potential* and *amount of QA effort* result in *defect correction effort reduced.* This node defines the possible reduction of DCE for every phase. The corresponding defect detection rates are shown in Table 5 and 6. In every development phase there is a specific detection rate dependant on the amount of QA effort and the defect type, e.g. in Table 5 column D(DE) represents detection rates for the DE phase for defects created in the DE phase itself and defects created in the previous phase RE. Node *defect correction effort after QA* is a subtraction node to calculate the difference between potential DCE and reduced DCE.

Table 5 Defect detection rates (RE, DE)

Re(R)	D(RE)	D(DE)	
		RE	DE
Low	10%	10%	10%
Medium	60%	60%	60%
High	75%	75%	75%
Very High	85%	85%	85%

Table 6 Defect detection rates (IM, I&T)

Re(R)	D(IM)			D(I&T)			
	RE	DE	IM	RE	DE	IM	I&T
Low	5%	5%	8%	5%	5%	8%	8%
Medium	30%	30%	53%	30%	30%	53%	53%
High	38%	38%	68%	38%	38%	68%	68%
Very High	43%	43%	78%	43%	43%	78%	78%

At this point the residing effort flows from phase RE to DE where it is adjusted by the *phase multiplier* to node *defect correction effort (RE) in DE*. It represents the increase of effort if defects are not corrected in the same phase where they are injected, e.g. if the design process has to be followed twice due to a defective requirement, it first has to be corrected (this effort cannot be saved) and in addition to that, the design might have to be redone. The phase multiplier varies from company to company and depends on the development process. The more development phases involved, the higher the overall effort needed for fixing these defects. There are different phase multipliers for different development phases dependant on their phase activities. The DCFM uses the phase multipliers illustrated in Table 7.

Table 7 Development phase multipliers

RE->DE	DE->IM	IM->I&T
4	5	4

For a DCE flowing from RE to DE, a phase multiplier of 4 is taken, leading to an effort multiplication by 4, e.g. if there is a residing DCE of 64 hours is left in phase RE, an effort of 256 hours is needed if it is detected in phase DE.. For DCE flowing from DE to IM it is 5 and from IM to the final phase I&T a factor of 4. This results in a worst case DCE multiplication of 80 for defects flowing through all development phases.

In phase DE, the nodes *amount of QA effort (RE) in DE* and *defect detection potential (RE) in DE* are different than in phase RE corresponding to the phase specifics, e.g. defect detection potential in phase IM is lower than in RE or DE for defects made in RE or DE because implementation is done according to requirements and design and it is not the task of a programmer to question the meaning of all his requirements nor the interface he uses.

This leads to different *defect correction effort reduced (RE) in DE* and *defect correction effort after QA (RE) in DE* in phase DE for defects originating in phase RE. The described pattern is repeated for every succeeding phase resulting in case of DCFM in a four phase model for phases RE, DE, IM and I&T.

4.4 Scenario Results

Following the goal to identify the ideal distribution of QA effort over all development phases (RE, DE, IM and I&T) of the development process under discussion, four scenarios have been defined based on the assumption to develop a complex feature with an estimated development effort of 1000 hours. These scenarios demonstrate the capabilities of the DCFM.

- *S1* is at *low* QA activities, the worst case scenario considering the development of DCE over all development phases.
- *S2* uses a *high* amount of QA effort typically used if you consider optimizing single development phase only. The definition of an additional scenario to demonstrate *medium* (average) QA activities has been left out because it performs similar to this one.
- *S3* has very high QA activities for RE and DE and a high amount for IM and I&T. It is expected to be too cost expensive if you consider every development phase only in its own context.
- *S4* uses very high amount of QA activities on all development phases.

Table 8 gives an overview on the amount of QA effort used for scenarios S1 to S4.

Table 8 Scenario overview

Amount of QA effort / phase	S1	S2	S3	S4
RE	Low	High	Very High	Very High
DE	Low	High	Very High	Very High
IM	Low	High	High	Very High
I&T	Low	High	High	Very High

The simulation results are illustrated throughout the following figures in form of a comparison of scenarios for every development phase. All values represent the calculated median of the predicted probability distribution for DCE.

For phase RE, Figure 8 illustrates a constant DCE of 200 hours in all scenarios. In S1, the DCE could only be reduced by 23 hours. S2 reduces the DCE by 150 hours whereas S3 and S4 have the highest DCE reduction of 170 hours due to very high QA activities.

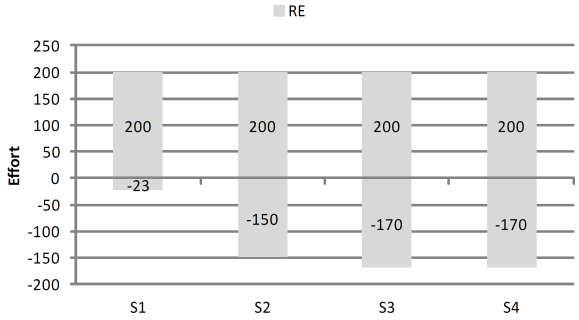

Figure 8 RE scenario results

The remaining DCE is shifted from phase RE to DE and adjusted by the phase specific multiplier. The development of DCE is illustrated in Figure 9. All scenarios have an additional DCE of 160 hours caused by defects originating in phase DE. The shifted DCE from RE has increased to 598 hours in S1. In S2, it is 199 hours whereas for S3 and S4 it is 119 hours of remaining DCE. With the corresponding QA effort, the DCE for defects which originated in phase DE could be reduced by 16 hours in S1, 120 hours in S2 and 136 hours in S3 and S4. DCFM assumes for defects originating in RE to have similar detection ratios as have DE defects. Thereby the DCE reduction for RE defects in phase DE is 17 hours in S1, 76 hours in S2 and 52 hours in S3 and S4.

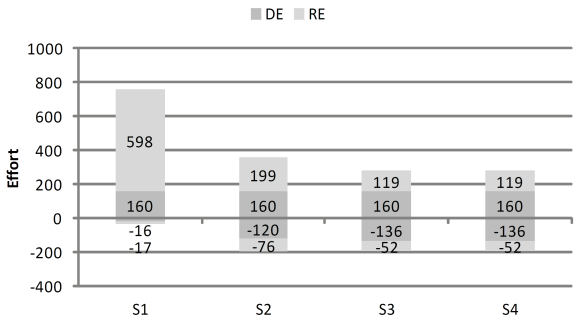

Figure 9 DE scenario results

For IM and I&T, result illustration and data have been divided into two charts due to the high level of detail. Table 9 and Figure 10 summarize IM results. Focusing only on defects with origin in RE, for Table 9 (column RE) the development of DCE has increased to 2529 hours in S1, in S2 the DCE is still 601 hours whereas S3 and S4 only have 332 hours. The reduction of DCE for defects with origin in RE is shown in column –RE. DCFM assumes low detection rates for RE and DE defects to be found in IM. Thus, the DCE is reduced by 36 hours in S1, 225 hours in S2 and 146 hours in S3 and S4.

Table 9 IM scenario result data

Phase	S1	S2	S3	S4
RE	2529	601	332	332
DE	648	198	119	119
IM	140	140	140	140
-IM	-12	-95	-109	-109
-DE	-38	-74	-53	-53
-RE	-36	-225	-146	-146

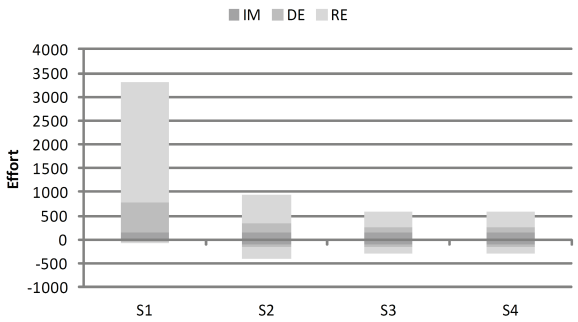

Figure 10 IM scenario results

Results for final phase I&T are presented in Table 10 and Figure 11. The DCE for RE defects is 8361 hours in S1, 1462 hours in S2 and 722 hours in S3 and S4. DCFM assumes higher detection rates for RE defects in the final phase I&T. Thus, the reduction of DCE is 851 hours851hours in S1, 1115 hours in S2, 375 hours in S3 and 614 hours in S4. Figure 11 also illustrates the final development of DCE for S1 to S4. It shows that S1 is still having 10 times more of DCE to be done than initially planned. This is when software projects run out of time or out of money and customers get disappointed. S2 has around 2000 hours of DCE left whereas S3 and S4 are at around 1000 hours, most of the rework needed on defects with origin in RE.

Table 10 I&T scenario result data

Phase	S1	S2	S3	S4
RE	8361	1462	722	722
DE	2048	482	265	265
IM	434	177	122	122
I&T	10	10	10	10
-I&T	-1	-6	-6	-7
-IM	-41	-121	-83	-96
-DE	-189	-361	-198	-225
-RE	-851	-1115	-375	-614

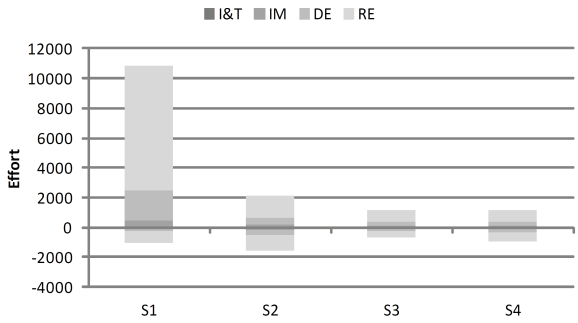

Figure 11 I&T scenario results

Table 11 summarizes the overall results. Initial development effort is 1000 hours for every scenario. Effort spent on QA activities is 50 hours in S1, 200 hours in S2, 320 hours S3 and 400 hours in S4. The overall DCE as part of QA activities is lowest for S1, 1224 hours. In S2 it is 2343 hours and S3 only has 1328 hours whereas S4 has 1608 hours. The DCE residing in the product after development is 9771 hours in S1, 528 hours in S2, 457 hours in S3 and 177 hours in S4. Finally, the overall simulation results expect S1 to need 12045 hours to develop a product initially planned at 1000 hours. S2 at 4071 hours of overall effort still consumes 3 times the amount of effort than planned. S3 has an estimated effort of 3105 hours close to S4 at 3185 hours.

Table 11 Scenario overall results

Result	S1	S2	S3	S4
Development effort	1000	1000	1000	1000
QA effort	50	200	320	400
DCE (part of QA)	1224	2343	1328	1608
Residual DCE	9771	528	457	177
Overall	12045	4071	3105	3185

Considering the overall amount of engineering effort it is still cheaper to invest in QA activities close to a maximum level than in optimizing effort spent for QA. Especially QA activities in early phases pay off because longer process iterations involve more process activities and therefore more effort.

5. CONCLUSION

For the engineering process under discussion the DCFM predicts the lowest DCE for a maximum of effort spent on QA measures in early development phases whereas later phases are predicted to have it benefit cost optimized. Typical cost benefit optimization strategies regarding the optimal effort spent on quality measures tend to optimize locally, e.g. every development phase is optimized separately in its own domain. In contrast to this we demonstrated that even cost intensive quality measures pay off when the overall DCE of specific features is considered.

The ideal amount of QA effort depends on DCE injected, whereas DCE itself is based on development effort and its corresponding DCF for the feature to be developed. Furthermore it depends on the people involved per development phase. The more people involved in the rework of an engineering artifact, the higher the overall DCE.

The overview on the influence of all KPIs of a software product is very complex. With DCFM project managers could not only monitor the current situation of a project but also estimate project behavior under given circumstances, e.g. project rescheduling or process optimization. Furthermore, DCFM could support higher process maturity levels, e.g. the Capability Maturity Model Integrated (CMMI) [29], a process improvement approach also used as reference for appraising a company's engineering processes.

The next step towards higher effort estimation performance and therefore better software projects in time, quality and costs is the establishment of our method as part of the continuous improvement process. One major part of it is the establishment of a long term measurement framework. Based on this data, we could further enhance our models and thereby provide decision support to process optimization and project estimation.

6. REFERENCES

[1] Basili V. R. and Rombach H. D, "The TAME project: Towards improvement-oriented software environments", IEEE Transactions on Software Engineering, 14(6), 758–773, 1988.

[2] Bibi S., Stamelos I., "Software Process Modeling with Bayesian Belief Networks", Proc. Int. Software Metrics Symposium, Chicago, 2004.

[3] Boehm, W. and Philip, M., "Understanding and Controlling Software Costs", IEEE Transactions on Software Engineering, 14(10), 1462-1477, 1988.

[4] Dabney J.B., Barber G., Ohi D., "Predicting Software Defect Function Point Ratios Using a Bayesian Belief Network", Proc. 2nd Int. Workshop on Predictor Models in Software Engineering, Philadelphia, 2006.

[5] Darwiche A., "Modeling and Reasoning with Bayesian Networks", Cambridge University Press, 2009.

[6] De Melo A.C.V., Sanchez A.J., "Software maintenance project delays prediction using Bayesian Networks", Expert Systems with Applications, 34, 908-919, 2008.

[7] Del Salgado Martinez J., del Aguina Cano I.M., "A Bayesian Network for Predicting the Need for a Requirements Review", in: Meziane F., Vadera S. (eds.), "Artificial Intelligence Applications for Improved Software Engineering Development: New Prospects", Information Science Reference, 106-128, New York, 2008.

[8] Fenton N., Hearty P., Neil M., Radliński Ł., "Software Project and Quality Modelling Using Bayesian Networks", in: Meziane F., Vadera S. (eds.), "Artificial Intelligence Applications for Improved Software Engineering Development: New Prospects", Information Science Reference, 1-25, New York, 2008.

[9] Fenton N., Marsh W., Neil M., Cates P., Forey S., Tailor M., "Making Resource Decisions for Software Projects", Proc. 26th Int. Conference on Software Engineering, 397–406, Washington DC, 2004.

[10] Fenton N., Neil M., Marsh W., Hearty P., Radliński Ł., Krause P., "On the effectiveness of early life cycle defect prediction with Bayesian Nets", Empir. Software Eng., 13, 499–537, 2008.

[11] Fenton N.E., Neil M.., Marsh W., Krause P., Mishra R., "Predicting Software Defects in Varying Development Lifecycles using Bayesian Nets", Inf. and Software Techn., 43(1), 32–43, 2007.

[12] Freimut, Denger, Ketterer, "An Industrial Case Study of Implementing and Validating Defect Classification for Process Improvement and Quality Management", Proc. 11th. Int. Symposium on Software Metrics, 2005.

[13] Hearty P., Fenton N., Marquez D., Neil M., "Predicting Project Velocity in XP using a Learning Dynamic Bayesian Network Model", IEEE Trans. Software Eng., 37(1), 124–137, 2009.

[14] Jensen F.V., "An Introduction to Bayesian Networks", UCL Press, London, 1996.

[15] Kruchten P., "The Rational Unified Process: An Introduction", Addison-Wesley, 1998.

[16] Lee, I., Leung, J.Y.T., Son, S.H., "Handbook of Real-Time and Embedded Systems", Chapman & Hall/CRC, 2007.

[17] Mendes and N. Mosley, "Bayesian Network Models for Web Effort Prediction: A Comparative Study", IEEE Trans. Software Eng., 34, 723-737, 2008.

[18] Mendes E., "Using Bayesian Networks for Web Effort Estimation", in: Meziane F., Vadera S. (eds.), "Artificial Intelligence Applications for Improved Software Engineering Development: New Prospects", Information Science Reference, 26-44, New York, 2008.

[19] Mendes E., "The Use of Bayesian Networks for Web Effort Estimation: Further Investigation", Proc. 8th Int. Conf on Web Engineering, 203-216, Yorktown Heights, 2008.

[20] Musilek P., Pedrycz W., Nan Sun, Succi G., "On the Sensitivity of COCOMO II Software Cost Model", Proc 8th IEEE Symposium on Software Metrics, 13–20, 2002.

[21] Pearl J., "Bayesian Networks: A Model of Self-Activated Memory for Evidential Reasoning", Proc. 7th Conf. of the Cognitive Science Society, 329–334, University of California, 1985.

[22] Pearl J., "Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference", Morgan Kaufmann, San Francisco, 1988.

[23] 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, 13–19, Cambridge, 2007.

[24] Radlinski L., "Improved Software Project Risk Assessment Using Bayesian Nets", Ph.D. Thesis, Queen Mary University, London, unpublished, 2008.

[25] Ram Chillarege, Inderpal S. Bhandari, Jarir K. Chaar, Michael J. Halliday, Diane S. Moebus, Bonnie K. Ray and Man-Yuen Wong, "Orthogonal Defect Classification-A Concept for In-Process Measurements", IEEE Transactions on Software Engineering, 18(11), 943-956, 1992.

[26] Russell S., Norvig P., "Artificial Intelligence. A Modern Approach", Second Edition, Pearson Education, Inc., 2003.

[27] Schulz, T., Radliński L, Gorges T. and Rosenstiel W., "Software Process Model using Dynamic Bayesian Networks" in Ramachandran M., (Ed.), "Knowledge Engineering for Software Development Life Cycles: Support Technologies and Applications", IGI Global, in press, 2010.

[28] Schulz, T., Radliński L, Gorges T. and Rosenstiel W., "PROMISE Repository of empirical software engineering data", http://promisedata.org/?p=224, West Virginia University, Department of Computer Science, 2010.

[29] Software Engineering Institute (SEI), "Capability Maturity Model Integrated", http://www.sei.cmu.edu/cmmi, 2010.

[30] Stewart, B., "Predicting project delivery rates using the Naive–Bayes classifier", J. Software Maint. Evol.: Res. Pract., 14, 161–179, 2002.

[31] Stolz W. and Wagner P., "Introduction of a quantitative Quality Management for the ECU Software development at Gasoline Systems", presented at BOCSE, 2005.

[32] Van Koten C., Gray A.R., "An application of Bayesian network for predicting object-oriented software maintainability, Information and Software Technology", 48, 59-67, 2006.

[33] V-Model, "Clarus Concept of Operations", FHWA-JPO-05-072, Federal Highway Administration (FHWA), 2005.

[34] Wagner S., "A Bayesian network approach to assess and predict software quality using activity-based quality models", Proc. 5th Int. Conf. on Predictor Models in Software Engineering, ACM Pres, 2009.

[35] Wagner S., "Global Sensitivity Analysis of Predictor Models in Software Engineering", Proc. 3rd Int. Workshop on Predictor Models in Software Engineering, Int. Conference on Software Engineering, pp. 3, Washington DC, 2007.

[36] Weiss, S. M. and Kulikowsky, C.A, "A Practical Guide to Designing Expert Systems", Yourdon Press, 1984.

[37] Wooff D.A., Goldstein M., Coolen F.P.A., "Bayesian Graphical Models for Software Testing", IEEE Trans. Software Eng., 28(5), 510–525, 2002.

[38] Z. K. W. and H.-M. Hauser, "The growing importance of embedded software: Managing hybrid hardware-software business", The Boston Consulting Group Inc., Tech. Rep., 2004.

[39] Zhou Y., Würsch M., Giger E., Gall H.C., Lü J., "A Bayesian Network Based Approach for Change Coupling Prediction", Proc. 15th Working Conference on Reverse Engineering, 27–36, Washington DC, 2008.