Defect Cost Flow Model with Bayesian Networks

Dissertation
der Mathematisch-Naturwissenschaftlichen Fakultät
der Eberhard Karls Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von
M.Sc. Thomas Schulz
aus Müllheim / Baden

Tübingen
2011

Tag der mündlichen Qualifikation:	30.09.2011
Dekan:	Prof. Dr. Wolfgang Rosenstiel
1. Berichterstatter:	Prof. Dr. Wolfgang Rosenstiel
2. Berichterstatter:	Prof. Dr. Norman Fenton

Zusammenfassung

Die Vorhersage von Software Fehlern ist eines der Hauptthemen der Software Entwicklung. Zur Bestimmung der Software Qualität bzw. des Aspekts der Fehleranfälligkeit dient weitläufig die Anzahl der Fehler als eine der wesentlichen Kenngrößen. Jedoch ist die alleinige Anzahl an Fehlern bei der Aufwandsschätzung von Fehlerkorrekturmaßnahmen ungeeignet, da der Fehlerkontext in diese Kenngröße nicht einfließt. Diese Arbeit stellt eine neue Möglichkeit der Schätzung des Fehlerkorrekturaufwandes vor.

In der Regel werden Fehler im Rahmen von Änderungen in das Software Produkt eingebracht. Ein Software Release umfasst potentiell hunderte dieser Änderungen. Dabei hat jede Änderung eine eigene Charakteristik hinsichtlich ihrer Fehlerwahrscheinlichkeit, die auf verschiedenen Einflussfaktoren basiert. In dieser Arbeit wird die Entwicklung des Software Prozess Modells (SPM) beschrieben, das die spezifischen Charakteristika jeder einzelnen Änderung im Kontext des Software Releases berücksichtigt.

Weiterhin haben Fehler der verschiedenen Entwicklungsphasen einen unterschiedlichen Einfluss auf den Fehlerkorrekturaufwand, abhängig von ihrem Ursprungs- und Entdeckungsort. Dabei steigt der Mehraufwand für die Fehlerkorrektur bei Fehlern, die über mehrere Entwicklungsphasen hinweg unentdeckt bleiben bzw. erst in einer Folgephase entdeckt werden. Um den Korrekturaufwand zu verringern, ist es wichtig, die einzelnen Entwicklungsphasen anhand ihrer spezifischen Merkmale zu bewerten und in diesem Kontext den Fokus von Qualitätsmaßnahmen zu definieren. Diese Arbeit beschreibt die Erstellung des Modells zum Fehlerkostenstrom (DCFM), das den Zusammenhang des Fehlerstroms über die Entwicklungsphasen zum Fehlerkorrekturaufwand herstellt.

Die Darstellungsform der Modelle in dieser Arbeit sind die Bayesschen Netze (BNs).

Sie zeichnen sich im Wesentlichen durch ihre Eigenschaft aus, probabilistische Kausalketten darzustellen und dieses unter der Einbeziehung von Expertenwissen in Kombination mit empirischen Daten. Die iterative Modellentwicklung berücksichtigt dabei die Problemanalyse, Datenanalyse, Modellerstellung, Simulation und Validation.

Die beiden entwickelten Modelle SPM und DCFM spiegeln das weitverbreitete Entwicklungsvorgehen anhand des V-Modells wider, ein internationaler Standard für die Entwicklung von Informationssystemen. Die Datengrundlage der Modelle basiert auf Projektdaten, die bei der Robert Bosch GmbH erhoben wurden. Die Analyse der verschiedenen Simulationsszenarien bestätigt, dass SPM und DCFM die realen Entwicklungsprozesse abbilden und auf dieser Basis Optimierungsstrategien erarbeitet werden können. Dabei ermöglicht die kausale Struktur der Modelle eine intuitive Anwendbarkeit.

Klassische Kostenoptimierungsstrategien in der Software Entwicklung tendieren dazu, die einzelnen Entwicklungsphasen separat in ihren eigenen Domänen zu betrachten. Im Gegensatz dazu zeigen die Ergebnisse dieser Arbeit, dass auch kostenintensive Qualitätsmaßnahmen einen Mehrwert darstellen, wenn man sie hinsichtlich des Fehlerkorrekturaufwandes im Kontext des gesamten Entwicklungsvorgehens betrachtet.

Abstract

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 thesis DCE has been modeled from a different perspective.

The most common way of inserting defects into a software product is by applying changes to it. In every software release, hundreds of changes are applied to the software product. Every change has its unique characteristics based on process and product factors. With regard to these factors, every change has its own probability of injecting a defect. In this thesis, the Software Process Model (SPM) is demonstrated taking into account the specific characteristics of multiple changes.

Furthermore, 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 it is important to assess every development phase along with its specific characteristics and focus on the shift of defects over phases. These ideas have been realized in the Defect Cost Flow Model (DCFM).

The modeling technique used in this thesis are Bayesian Networks (BNs) which, among many others, have 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.

The developed models SPM and and the Defect Cost Flow Model (DCFM) reflect 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 the Robert Bosch GmbH. The analysis of evaluation scenarios confirms that SPM and DCFM correctly incorporates known qualitative and quantitative relationships. Because of its causal structure it can be used intuitively by end-users.

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 SPM and DCFM demonstrate that even cost intensive quality measures pay off when the overall DCE of specific features is considered.

Acronyms

AI Artificial Intelligence. 20

ASIL Automotive Safety Integrity Level. 10

BN Bayesian Network. 5–7, 19, 20, 23, 24, 26–29, 33, 52, 58, 69, 70, 74, 80, 107, 115–117, 119

CMMI Capability Maturity Model Integrated. 7, 10, 12, 13, 116

COCOMO Constructive Cost Model. 108, 109, 119

CU Customer. 18, 104, 105

DBN Dynamic Bayesian Network. 27, 28, 51, 52, 69, 117

DCE Defect Correction Effort. 3, 4, 31, 40, 42, 43, 45–48, 51–54, 59, 62–64, 66, 69, 73–80, 82, 83, 85–88, 90–93, 97, 99, 104, 109, 110, 112–114, 116–118, 120

DCF Defect Cost Factor. 5, 39, 46–48, 60, 61, 86, 105, 109, 112, 113, 116–118, 120

DCFM Defect Cost Flow Model. 5–7, 71, 73–75, 83, 86, 87, 107, 109, 110, 113, 114, 116–120

DE Design. 16, 18, 46, 75–77, 80, 83, 84, 88–94, 97, 99, 101, 103, 105, 110–113

DFM Defect Flow Model. 7, 18, 74, 77, 117

DOE Design of Experiment. 5, 48, 62, 120

ECU Electronic Control Unit. 8, 21, 22

GQM Goal Question Metric. 31, 54

HMI Human Machine Interface. 46, 52, 60

I&T Integration & Test. 17, 18, 46, 75–77, 80, 84–86, 88, 90, 94, 97, 99, 101, 103, 104, 120

IEC International Electrotechnical Commission. 9, 10

IM Implementation. 17, 18, 75–77, 80, 83, 84, 88–90, 93, 97, 99, 101, 103, 110–113

ISO International Organization for Standardization. 10, 11

KPI Key Performance Indicator. 3–5, 32, 34, 40, 48, 51–54, 69, 73, 74, 80, 107–109, 116, 118

NPT Node Probability Table. 20, 24, 27, 82

PM Project Manager. 2–4, 19, 31, 115–118

QA Quality Assurance. 1, 3–5, 11, 18, 108–110, 113, 116, 118, 120, 121

RE Requirements Engineering. 16, 18, 46, 75–77, 80, 82–84, 86, 88–94, 97, 99, 101, 103, 105, 110–113

ROI Return of Invest. 6, 107

SPM Software Process Model. 5–7, 51–53, 65, 70, 107, 109, 115–118, 120

Contents

Abstract iii

Acronyms ix

1 Introduction 1

1.1 Motivation 1

1.2 Research Hypothesis 3

1.3 Outline 4

2 Background 7

2.1 Automotive Product Engineering 7

2.2 Defect Flow Model 18

2.3 Bayesian Networks 19

2.4 State of the Art 28

3 Development Methodology 31

3.1 Procedure Model 31

3.2 Statement of the Problem 33

3.3 Building and Simulating 35

3.4 Model Validation 36

3.5 Refinement and Generalisation 37

4 Model Data Chapter 39

4.1 Overview 39

4.2 Internal Data 40

4.3 Feature Classification 45

4.4 Defect Cost Factor 46

4.5 Experiment Data 48

5 Software Process Model 51

5.1 Overview 51
5.2 Problem Definition 52
5.3 Identifying KPIs and Relationships 54
5.4 Building and Simulating 56
5.5 Model Usage 65
5.6 Results 67
5.7 Model Complexity 69
5.8 Conclusion 70

6 Defect Cost Flow Model 73

6.1 Overview 73
6.2 Problem Definition 74
6.3 Defect Cost Flow 75
6.4 Identifying KPIs and Relationships 79
6.5 Model Creation 80
6.6 Model Calibration 86
6.7 Scenario Definition 89
6.8 Simulation Results 90
6.9 Conclusion 104

7 Practical Benefits 107

7.1 Developing Knowledge 107
7.2 Process Changes 109

8 Summary and Outlook 115

8.1 Model Analyses 115
8.2 Novel Contributions 117
8.3 Hypothesis Confirmation 118
8.4 Research Activities 119
8.5 Process Optimization 120
8.6 Perspectives 121

List of Tables

2.1 Summary of recent BNs in software engineering 29

4.1 Development and defect correction effort 43
4.2 Type distribution error 45
4.3 Defect Cost Factor over feature 48
4.4 Number of defects over deadline pressure 49

5.1 Features 60
5.2 Type of change distribution 61
5.3 Deadline pressure 63
5.4 Effort spent on testing 64
5.5 Defect detection rates 65
5.6 Project scenarios 66
5.7 Prediction accuracy 67
5.8 Simulation results 69

6.1 Defect Cost Flow data 77
6.2 DCFs per development phase 86
6.3 Sufficiency of QA effort 87
6.4 Effort Reduction Rates 88
6.5 Development phase multipliers 88
6.6 Scenario overview 89
6.7 RE scenario result data 91
6.8 DE scenario result data 92
6.9 IM scenario result data 93
6.10 I&T scenario result data 94
6.11 S1 result overview 96

6.12 S1 result data 97
6.13 S2 result overview 98
6.14 S2 result data 99
6.15 S3 result overview 100
6.16 S3 result data 101
6.17 S4 result overview 102
6.18 S4 result data 103
6.19 Scenario overall results 104

7.1 DCFM overall results 110

List of Figures

2.1 V-Model development process 15
2.2 Defect Flow Model 18
2.3 Example Bayesian Network 23
2.4 Entering observations 25
2.5 Backward reasoning 25
2.6 Common Bayesian Network structures 26

3.1 Research procedure 32

4.1 Measurement concept 41
4.2 Overall effort 43
4.3 Number of changes 44
4.4 Effort distribution 44
4.5 Effort over feature 47

5.1 Effort estimation 53
5.2 Change model BN 57
5.3 Setup SPM 58
5.4 Change factors 60
5.5 Process factors 62
5.6 Product factors 63
5.7 Test activities 64
5.8 Development effort 68
5.9 Defect correction effort 68

6.1 DCFM concept 76
6.2 Defect Cost Flow 78
6.3 Bayesian Network DCFM 81

6.4 DCFM part RE 82
6.5 DCFM part DE 83
6.6 DCFM part IM 84
6.7 DCFM part I&T 85
6.8 RE scenario results 91
6.9 DE scenario results 92
6.10 IM scenario results 93
6.11 I&T scenario results 94
6.12 S1 DCFM chart 96
6.13 S2 DCFM chart 98
6.14 S3 DCFM chart 100
6.15 S4 DCFM chart 102

7.1 Effort 111
7.2 Occurrence 112
7.3 Review evaluation results 113

1 Introduction

1.1 Motivation

1.1.1 Automotive Industry

For the next years, experts estimate the global readmission rate for cars to be over 70.000.000 per year [Zimmermann and Hauser, 2004]. 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. A by- product of this, in recent years 50% of all vehicle recalls were software related.

The increasing part of embedded software and the resulting complexity does not only lead to improved new functions but also to an increasing defect rate. This affects quality and has a strong influence on success in the business. Especially for safety critical use, which is characteristic for the automotive sector, reliability has to be guaranteed by systematic defect avoidance already during development.

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.

The most common way of inserting defects into a software product is by applying changes to it. In every software release, hundreds of changes are applied to the software product. Every change has its unique characteristic based on process and product factors. With regard to these factors, every change has its own probability of injecting a defect. Furthermore, defects originating from other development phases than the one they were detected in, have a different impact on the overall product costs compared to defects detected in the same development phases in which they were applied.

1.1.2 Product Assessment

Decision making in software engineering is often based on various influencing variables from multiple domains which a single expert can not have a complete overview of. One of the central topics when developing innovative safety critical products is the assessment of a product's defect rate before it is released. For example, it is very difficult to estimate the defect rate of a specific software release when hundreds of changes have been applied to it. Major indicators are, e.g.

- What parts of the software have been changed?
- How large were the changes to the software?
- How complex and difficult were these changes?
- Have all new functionalities been tested?

The defect rate of a software product has to be considered when estimating the effort needed to complete a project. Especially the rework needed to fix defects is one of the uncertain variables. In a project's context, the defect rate of the software product has further influencing factors, e.g. the project's deadline pressure, which further has to be taken into consideration when estimating the project effort.

For *Project Managers (PMs)* it is difficult to overview these variables and their impact on the product's defect rate. Their interpretation varies from one PM to another leading to different decisions among different PMs.

1.1.3 Process Improvement

One goal of the continuous improvement process is to identify areas where optimization leads to lower defect rates. Therefore, the efficiency of defect detection- and correction activities are maximized given certain boundary conditions. For example, if the reduction of the overall defect rate is considered, how is the effort for review and test activities distributed most effectively throughout the process phases? Before optimizing the development process, all process phases need to be analyzed concerning their effectiveness. This analysis needs to consider a variety of influencing variables. Especially the influence of every variable on the optimization criteria is of great interest. Based on this analysis, process changes can be introduced including the evaluation of their improvement. For PMs it is difficult to overview the impact of changing process variables to the overall engineering process. Thus, it is difficult to find the optimal process optimization measures for a specific development process.

1.2 Research Hypothesis

The goal of this thesis is derived from the problems addressed in the previous sections. These problems primarily focus on the assessment of software products and the supporting development processes regarding the effectiveness of QA measures and cost reduction. These problems can be summarized to the following statement:

- P1 Experts can not overview the impact of all *Key Performance Indicators (KPIs)* to product quality and costs.
- P2 Expert knowledge and process data are combined on a subjective basis leading to different decisions among different experts.

The overview on all main influencing variables, named KPIs throughout this thesis, is crucial to identifying optimization measures. Furthermore, it is important to understand the relationship among these KPIs to get an overall understanding of the development processes. Only with the understanding of how KPIs affect the defect rate of a software product it is possible to identify optimal measures leading to lower defect rates as well as cost reduction. Furthermore, with the understanding of the overall development process, it is possible to effectively distribute QA effort over process phases.

In this thesis, effort is used to describe costs. In addition to that, the *Defect Correction Effort (DCE)* is used to provide a measure for the defect rate of a software product.

Development effort as well as DCE form the estimation variables. Due to the common unit of these variables, it is possible to incorporate all KPIs within a consistent model. Moreover, the resulting model should be able to incorporate both, process data and expert knowledge when no data is available. Thus, the main goals of this thesis can be formulated as follows:

- G1 Enable PMs to identify QA measures where optimization leads to lower product costs.
- G2 Provide a method to identify product areas where optimization has an optimal cost-benefit ratio regarding a defect rate reduction.
- G3 Develop a model for a specific software development process to describe the KPIs for development effort and DCE.
- G4 Describe the KPIs in form of process data and expert knowledge when no data is available.

Based on these goals the following research hypothesis for this thesis are defined:

- H1 It is possible to develop estimation models for development effort and DCE that incorporate process data and expert knowledge in the absence of significant data.
- H2 It is possible to enable PMs to identify product areas where additional effort spent on defect rate reduction has an optimal cost-benefit ratio.
- H3 It is possible to incorporate the supporting development process to distribute effort for QA measures most effectively.

1.3 Outline

Chapter 1 introduces this thesis. Here, explicit research hypotheses are constructed based on the problems motivating this thesis.

Chapter 2 provides information about this thesis' problem domain. It describes the fields of automotive software engineering and gives insights into domain specifics, i.e. the engineering process model as well as QA topics are described. The section on automotive engineering is followed by an introduction to *Bayesian Networks (BNs)*, the method used to create the models of this thesis. It describes the motivation behind selecting BNs and introduces their main characteristics. The background chapter closes with a state-of-the-art section, describing similar activities to those presented in this thesis.

Chapter 3 describes the *methodology* followed to achieve the goal of this thesis. Every research stage, from defining the problem to validating the models is described along the research procedure.

Chapter 4 describes the *data* elicited for the models presented in this thesis. Process and project data as well as expert knowledge is collected from an industrial partner, one of the leading companies in the field of embedded applications in automotive industries. The data base provides insights to specific defect rates of a historical project as well as the development process. During the data analysis *feature classification* has been introduced to distinguish areas of of the software product with different defect rates. Furthermore, the *Defect Cost Factor (DCF)* is defined to quantify the impact of defects on the software development. In addition to that, this chapter describes the *Design of Experiment (DOE)* conducted to understand the impact of major influencing variables, i.e. the impact of deadline pressure on the defect rate of a software product.

Chapter 5 presents a model to estimate the amount of defects for a specific software product. It focuses on the estimation of defects based on the changes introduced in a specific software release cycle. The resulting model is called the *Software Process Model (SPM)*. After the *Overview* section, the main goal of SPM is defined. Based on this, SPM's main variables are defined. After this, the section on *Building and Simulating* describes the actual building of the model. It is followed by a section on the *model usage*. Scenarios are defined, analyzed and presented as a part of the *results* section. Finally, the *Conclusion* section summarizes the results.

Chapter 6 describes the creation of the *Defect Cost Flow Model (DCFM)*. The aim of the DCFM is the identification of development phases in which an optimization of QA effort will lead to lowest costs. The first sections provide background information on the DCFM. Based on the *problem definition*, the KPIs and their relations can be defined. The following sections describe the DCFM in detail including the simulation of several scenarios demonstrating its capabilities. The simulation results are discussed after that. Finally, the conclusion section summarizes the simulation results and describes further steps in the domain of DCFM.

Chapter 7 presents the practical benefits of SPM and DCFM. Because this thesis has been carried out in an industrial environment, several usage areas of such models could be identified. It starts with the gain of knowledge in different areas, e.g. in modelling techniques like BNs, accompanying the work of SPM and DCFM. But, most importantly, the *Return of Invest (ROI)* is described based on a change in the development process within the industrial environment. For this, an additional evaluation has been carried out resulting from SPM and DCFM.

The final chapter 8 *Summary and Outlook* concludes this thesis.

2 Background

This chapter gives background information about the main aspects of this thesis, i.e. the *Software Process Model (SPM)* and the *Defect Cost Flow Model (DCFM)*. Section 2.1 describes the domain of *embedded software engineering* along with the challenges to be met. It describes the engineering process on which the hypothesis evaluation of this thesis is based on. The following section on *embedded software* engineering introduces the standards and norms used for the development of automotive software as well as other safety critical embedded software. These sections motivate the need for fail-safe products and therefore methods describing how to reduce software defects. The section about the *Defect Flow Model (DFM)* introduces a concept to describe defects as part of the development process. It is followed by the principles of *Bayesian Networks (BNs)*, the probabilistic theory used to develop the models for this thesis. The final section of this chapter is a review of the *state-of-the-art* in the field of predictive software engineering showing this thesis' relationship to current research.

2.1 Automotive Product Engineering

This section describes the field of automotive product engineering depicting the boundary conditions for the development of fail-safe automotive products, one major motivation for this thesis. The *introduction* is followed by a section on *standards and norms* defining the requirements for the development of safety critical embedded systems. The following sections describe general purpose process models fulfilling these standards and norms, i.e. *quality management*, the *Capability Maturity Model Integrated (CMMI)* and the *V-Model development process*, an international standard for developing software products. The description of these general purpose process models is followed by an introduction to the development process implementing the described standards. It is used within the automotive industry to develop high quality products. The concepts and models developed as part of this thesis have been developed in this specific engineering environment.

2.1.1 Introduction

The demand for software products in the automotive industry has been increasing exponentially over the last decades, making software engineering one of its critical success factors [Lee et al., 2007]. The extensive use of *Electronic Control Units (ECUs)* in vehicles today has already lead to an average of 20 ECUs in smaller and even more than 70 ECUs in upper class cars [Zimmermann and Hauser, 2004]. Complex features with short time to market and minimal engineering costs at very high quality are the challenges to be met. Besides functional complexity, embedded software is the subject to domain specific characteristics, especially:

- code efficiency due to limitations of processor and memory resources.
- code portability to ensure further development and variant handling.
- high availability and reliability to support safety critical applications.
- real-time operation.
- network operation.

With the software part continuously increasing, the automobile has turned into a complex technical product. It has to fulfill increasing requirements by the customer and legal specifications. Basically, these are marked by high requirements for reliability, availability and safety. As safety relevant vehicle functions increase, acting without the active interference of the driver, the analyses of the function safety and the specification of suitable concepts have great importance in the development. The requirements and the resulting cross-company collaboration lead to an intensified tendency towards distributed systems.

Additionally, the function development needs to follow a defined domain specific process, specifically adjusted to the character of embedded systems and the development context. The development of high-quality products in such an environment is a great challenge as they underlie the frame conditions of limited costs, short development cycles and high product variability. Standards and norms as well as the derived supporting processes, described in the following sections, are common practice in the automotive industry. The so-called maturity models enable an evaluation of the companies in respect to the availability of the corresponding processes, which are increasingly being used as a basis for order placement. For the development of automotive specific software it is indispensable to have a continuous process, including all phases of the development process, as a result of the interaction of different engineering disciplines. The use of systematic development processes is a necessary prerequisite for continuous improvements and is mostly acknowledged or already mandatory. Next to standardized progress models for the software development, there are general, not software specific standards, which are partially having extensions and adjustments for specific domains.

2.1.2 Standards and Norms

Standards and norms serve the improvement of products and processes to increase the general quality. Especially in areas concerning safety relevant systems, it is most important to obey the defined processes. At the same time, the customer's requirements for product and process certification have to be added. The following standards are already followed for the development of embedded software within the automotive industry.

IEC 61508

The standard "functional safety of safety relevant programmable electronic systems" of the *International Electrotechnical Commission (IEC)* is relevant for the development of safety oriented functions. It defines requirements for the development of safe systems without being limited to one domain.

ISO/CD 26262

A standard of the *International Organization for Standardization (ISO)* is originating especially for the vehicle development, being an adaption of the IEC 61508. The ISO 26262 “Road vehicles - Functional safety" is to be passed in 2011 as a public standard. ISO 26262 is already adhered to within the automotive industry to further increase the reliability of safety products. The increasing demand for further safety measures is one reason for the research activities carried out as part of this thesis.

At the moment it is in a committee draft for examination and further processing. The basic organisational requirements of this standard refer to the development process and its support processes and also to the management of the functional safety. The technical requirements for the software development include recommendations for the use of techniques and methods, being realized dependent on the *Automotive Safety Integrity Level (ASIL)*.

ISO 26262 contains the definition of responsibilities and activities to guarantee the functional safety during the development and also independent from the project. The third part is the definition of the overall system and the corresponding requirements, e.g. the observance of certain standards and the legal framework. Here, dangers and risks of potential errors are estimated according to appropriate techniques and classified in ASIL.

The standard further describes the product development being divided into system, hard- and software development. It contains specific requirements, e.g. how the methods are used, being specified according to ASIL as optional, recommended or urgently recommended. As a reference process for the single phases of the product development the V-Model has been chosen. ISO 26262 defines the introduction of corresponding production and process work flows to guarantee the functional safety and also the observance of the requirements by all involved manufacturers.

Furthermore, the standard refers to all requirements in the area of the support processes. In the organisational area of the standard one can find agreements for the requirements of maturity models as the CMMI. That is why the certification of at least level two is seen as a prerequisite for the conformity of the development process referring to ISO 26262.

2.1.3 Quality Management

Quality management serves as a support process of the software development. The standard ISO 26262 described in the previous section contains guidelines and recommendations for test procedures of safety critical application in the automotive sector. It establishes certain measures to guarantee the fulfilment of specific quality requirements. All of these measures are summed up under the notion of quality assurance. Software quality assurance aims to increase the trust in software products and lead to a quality increase in development. Quality management is based on the assumption that software quality is directly defined by the quality of its development process. In opposition to the processes of manufacturing, from which the assumption is derived, there is also a certain influence by individual capabilities and experiences in software development. Factors as the novelty of a requirement have an effect on quality. But it has become clear that the process quality still largely affects the product quality in spite of the mentioned external factors.

Increasing process quality is the prerequisite for high-quality software products. The definition of standards in the development and their observance are the basis of this. The process related quality assurance takes place by the examination of the development process according to the agreement of the result to be delivered and the previously defined standards and goals. The product related quality assurance refers to the examination of the software products according to previously set quality features. Measures of *Quality Assurance (QA)* for software do not only aim to examinations but distinguish the following three categories. They complement themselves:

- *Organisational Measures*: measures for the systematic processing of the development and quality assurance, e.g. measures for planning and examining.
- *Constructive Measures*: measures for the avoidance of errors in software development, e.g. use of appropriate methods, training of employees support the use of a process model.
- *Analytic Measures*: measures for tracing errors in the results, e.g. reviews and software tests.

Constructive measures alone cannot guarantee the accuracy of a software product. If the development is phase oriented, there is the possibility of realizing constructive measures in every phase. At the end of each phase there is an evaluation of the interim results' quality by means of analytic measures. If the quality is insufficient, the interim result has to be altered until it accords with the previously defined quality and can be released for the next phase. This procedure enables an early defect recognition- and correction. It is advantageous that errors can be already recognised when they originate. It also keeps the effort to correct them very low.

Basically, the examination of embedded software does not differ from conventional software. However, for embedded systems there are higher requirements for the quality attributes. The primary goal of the examination of embedded systems is the examination of the correctness as potential errors might cause serious damage. In the framework of the analytical quality assurance several techniques exist to examine the software. They are classified as follows:

- *Dynamic*: the product is tested at runtime with explicit test data, e.g. with a threshold value analysis.
- *Static*: the artifacts describing the product are tested, e.g. with reviews.
- *Formal*: the software is tested according to formal means, e.g. with model checking.

Dynamic test procedures are applicable for embedded systems but examinations for correctness are only possible for previously defined test cases. Statements about reliability and safety can hardly be rendered by this, only in connection with other procedures. It is similar to the situation of static and formal procedures. To consider all aspects a combination of the available test procedures has to be selected dependent on correctness, reliability, availability and safety.

2.1.4 CMMI

For evaluating and optimizing the process the CMMI provides a vast reference model. The basis for process improvements are “Best Practices". These are practices which already proved themselves. The CMMI consists of different process areas, which among others are there to define procedures for system, hard- and software development and also the corresponding support processes.

The process areas of the CMMI each contain the specific requirements for a certain specialist field and are the central structural elements of the model. From them the single process areas are derived. They are divided into specific and generic goals. While the specific goals refer to certain process areas, the generic goals follow general formulations. They describe the “institutionalisation of a process area” and provide the instruction how the specific goals can be realized in the long-term. To reach the goal practices are described in detail by part practices and typical outputs. The CMMI differentiates between a cascade and a continuous display. In the cascade the so-called maturity levels have five levels:

1. *Initial*: each process is more or less un-defined or hardly defined. The success depends on the performance and the competence of single employees.
2. *Managed*: The most important management processes for planning, processing and controlling of projects are implemented.
3. *Defined*: consistent management and development processes for the overall organisation are defined.
4. *Quantitatively Managed*: Measurements and figures to forecast and control the processes are used systematically.
5. *Optimizing*: The processes underlie continuous improvement with the help of systematic selection, the introduction of improvements and also error and problem analysis.

These maturity levels complement each other. For reaching a higher level the level below including the corresponding requirements has to be fulfilled. The first maturity level is an exception as it has no assigned requirements. The cascade specifies two generic goals. The first one defines the institutionalisation of the processes on the second maturity level. The second one is valid for all higher maturity levels.

2.1.5 V-Model Development Process

The life cycle model underlying the models presented in this thesis fulfills the standards and norms described in the previous sections. Life cycle models are procedures by which software is engineered in a defined way using structured process steps from the requirements to delivery. The implemented process is based on the V-Model [IABG, 1992], an international standard for developing information technology systems. The V-Model is an enhancement of the common waterfall model and is intended for planning and executing interdisciplinary projects, combining hardware and software as well as high level system engineering activities within a single framework. For decades, the V-Model has been enhanced to reflect the needs of the automotive industry to develop high quality, failsafe products. The V-Model describes different phases of a software release life cycle and focuses on high level system specification and testing as well as on implementation level, module testing respectively. On the management level, every phase of the V-Model is supported by specific methods to ensure a seamless integrated engineering environment.

Figure 2.1 illustrates the V-Model phases. Starting with *System Requirements Engineering & Design*, high level functions are specified according to the desired product functionality. They are further specified and assigned to specific software components according to their functionality. These components contain software modules building the code base of the product. Detailing of requirements and testing is continued from the highest level of function definition down to the module level where module reviews and tests are performed. The right side of the V-Model is related to integration and testing. Modules are integrated into components and components are integrated into the overall software. Software engineers assess the software on a more detailed, software related perspective. After software testing, the overall software is integrated in the system forming an ECU as part of a vehicle's functionality. System engineers review and test high level product functionality from the customer's perspective.

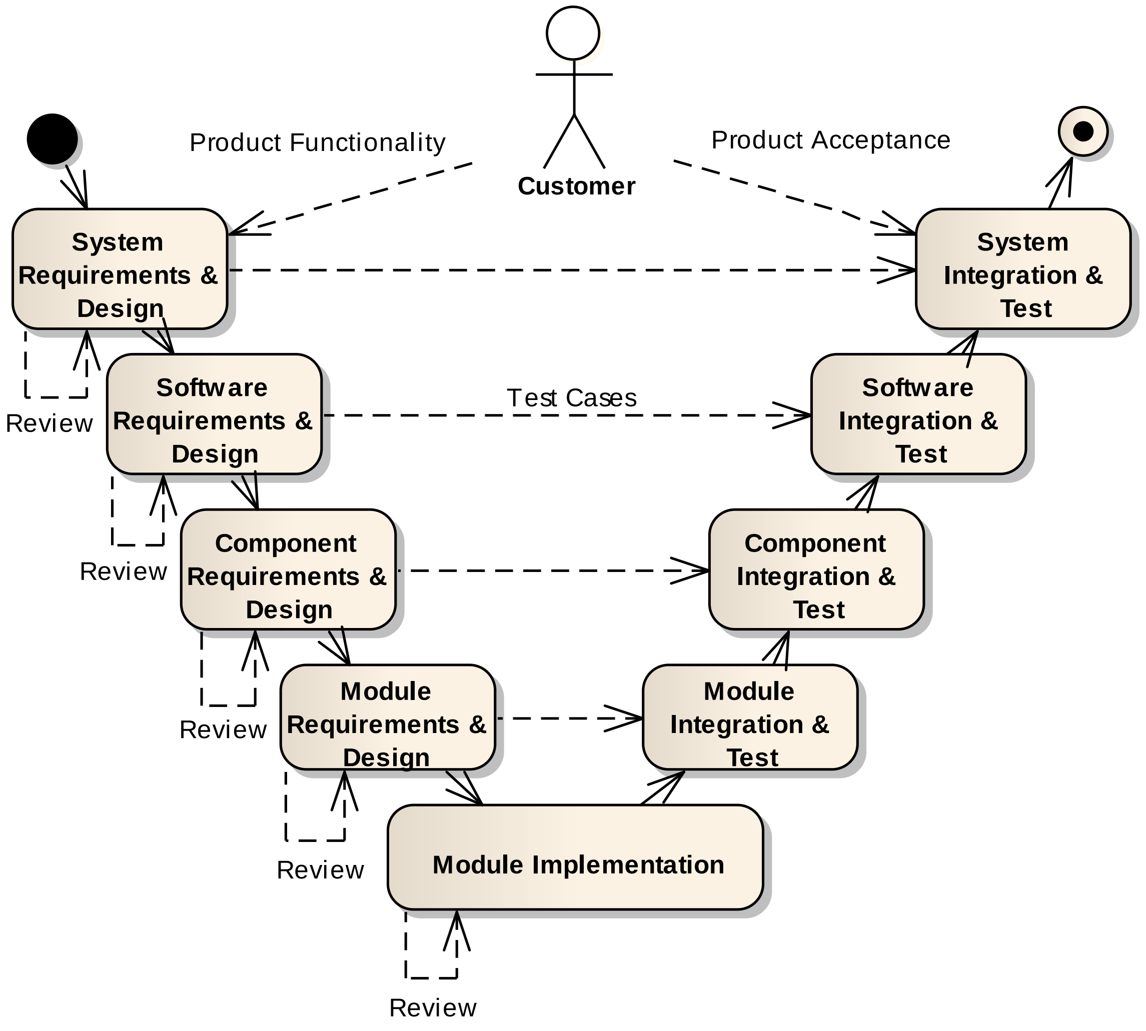

Product Functionality
Customer
Product Acceptance
System Requirements & Design
System Integration & Test
Software Requirements & Design
Test Cases
Software Integration & Test
Review
Component Requirements & Design
Component Integration & Test
Review
Module Requirements & Design
Module Integration & Test
Review
Review
Module Implementation
Review

Figure 2.1: V-Model development process

2.1.6 Automotive Software Development

The software development processes underlying the concepts and models developed as part of this thesis follow a defined and documented life cycle [CDQ, 2010]. Based on the criteria for each life cycle, the project determines which of the life cycle models will be used. As a minimum, each of the approved software life cycles must include the following phases:

- Requirements Engineering
- Design
- Coding / Implementation
- Integration & Testing

Depending on the life cycle, these phases are applied repeatedly, e.g. in incremental, iterative life cycles or when verification steps are applied after each phase.

Requirements Engineering

Within the *Requirements Engineering (RE)* phase, the customer specification is analyzed, or if it is not delivered by the customer, it is developed and refined into the internal specification. The project transforms the customer specification into the internal target specification in terms of completeness and adequate level of detail. Thereby the project checks whether further requirements need to be supplemented to the internal specification (e.g. non-functional requirements, standards, company-internal objectives). It is unambiguously communicated to the customer which requirements are accepted and which are not. The internal requirements specification forms the basis for the succeeding design phase.

Design

The *Design (DE)* phase, several architecture describing documents are developed to specify the functions as well as the interfaces within the software and to the hardware. In case there is a need to develop alternative solutions, systematic decision methods are applied to identify the best solution. The design phase is carried out either in one step or in several stages (architectural design, detailed design).

Coding / Implementation

Within the *Implementation (IM)* phase, work products used to create executable programs are defined and documented in terms of their structure and content. Rules are defined such as coding guidelines, provisions for using code generators, strategies for the later integration and configurations of tools (e.g. compilers, code checkers, etc.). The IM phase is followed by calibration and parametrization steps influencing the functional behavior of the software.

Integration and Testing

Phase *Integration & Test (I&T)* describes the software verification and validation process. At a minimum the following topics (type and scope of the evaluations that have to be carried out) are defined:

- Applicable dynamic software testing
- Regression testing
- Reviews and applicable methods (e.g. inspections, walkthroughs)
- Criteria and procedures for the release of software

These process phases form the software development life cycle followed for every iteration of the product development. There are further supporting processes, e.g. project-, configuration- or change & defect management needed to plan, assess and control every life cycle iteration as part of the overall product development.

2.2 Defect Flow Model

The automotive industry uses the DFM as a measurement system supporting the quantitative evaluation of QA measures in their engineering processes [Stolz and Wagner, 2005]. The DFM is based on the orthogonal defect classification concept for process measurements [Chillarege et al., 1992]. Its main goal is to provide transparency on phase specific defect rates. Defects are represented based on where they are made in relation to where they are found. Based on the DFM, an analysis of every phase can be assessed separately for its specific defect rates to identify phases where to focus QA measures on. The DFM uses the number of defects as indicator for development phase performance.

A sample DFM is illustrated in Figure 2.2 based on a data set presented in [Klaes et al., 2007]. It has defect data in the form of number of defects for specific development phases, i.e. RE, DE, IM, I&T and *Customer (CU)*. It depicts 55 defects introduced and 40 defects detected in 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 that 33 product defects are left, probably detected by the customer. According to [Stolz and Wagner, 2005] the DFM has proven its capability to monitor and improve quality processes in the domain of software development for automotive applications .

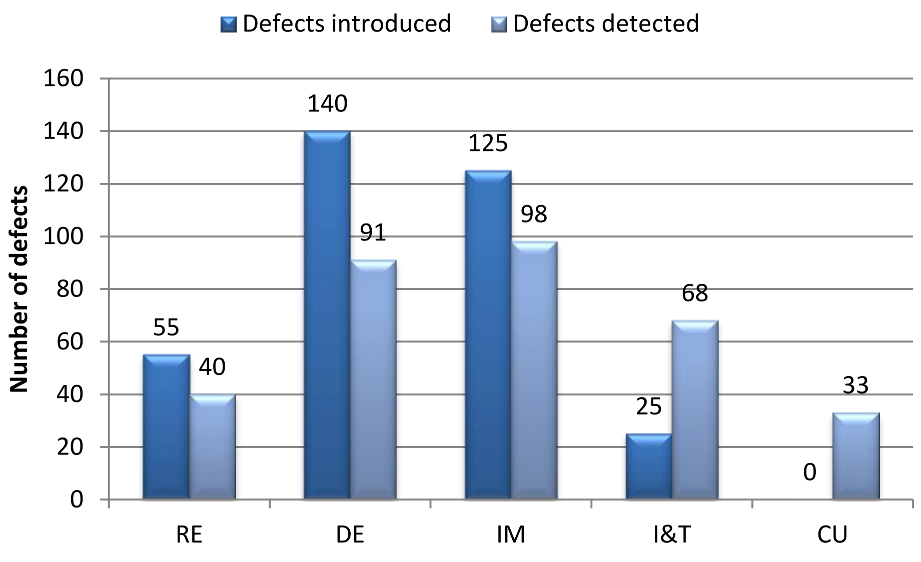


Figure 2.2: Defect Flow Model

2.3 Bayesian Networks

This section provides an introduction to BNs, the method used for creating the models of this thesis. It describes the motivation behind selecting BNs and introduces their main characteristics.

2.3.1 Introduction

Managers often have to assess projects without historical data available or where relevant data is difficult to collect. Managing software projects for innovative products implies adapting project conditions to actual needs of a specific project or the surrounding processes taking into account even process changes based on latest knowledge from process improvement activities. Even though from a statistical point of view there is no relevant data available. *Project Managers (PMs)* are still able to assess such projects mostly based on expert judgement as well as taking related information into consideration.

It is not possible to build a predictive model automatically reflecting the software engineering processes from data because the data sets 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, splbn are selected because of 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

BNs have received a lot of attention over the last decades from both scientists and engineers. They are part of modern techniques of artificial intelligence (AI). Although a fundamental theorem on conditional probability was introduced by The Reverend

Thomas Bayes back in XVIII century [Bayes, 1763], the term “Bayesian network” and its concepts were introduced in the 1980’s in pioneering work of Judea Pearl [Pearl, 1985, 1988]. Experienced researchers have recently put strong statements in the review of one of the books [Darwiche, 2009] on BNs: “Bayesian Networks are as important to *Artificial Intelligence (AI)* and Machine Learning as Boolean circuits are to computer science”, “Bayesian networks have revolutionized AI”. A BN can be perceived from two perspectives:

1. as a graph where the nodes represent random variables and the arcs represent the relationships between variables,
2. as a formal mathematical model definition where variables are expressed as conditional probability distributions.

The graphical representation of cause and effect chains makes it possible to overlook and communicate even complex structures. Bayes’ theorem [Bayes, 1763] enables us to reason from cause to effect and vice versa. Propagation of information from any part of the BN makes it possible to enter observations in specific nodes and revise probabilities of the depending nodes. Due to Bayes’ theorem trade-off analysis including sensitivity analysis and what-if scenarios are possible enabling us to reason under uncertainty.

2.3.2 Bayes’ Theorem

BNs describe causal relationships between multiple influencing factors with the possibility to combine expert knowledge with statistical data in a single model [Castillo et al., 1997]. A BN represents relationships between causes and effects. It is made up of nodes and arcs whereas arcs represent the relation between nodes and nodes represent the real cause or effect.

Numerically a BN consists of a set of *Node Probability Tables (NPTs)* quantifying the node and its relation to others. A node which does not have a link directed to it is called a *root node* and is therefore defined by an *unconditional probability distribution*. In contrast to that, a node for which there is at least one incoming link from another is called *child node* defined by a *conditional probability distribution*. NPT are used as synonym to these distribution types.

The inference process in a BN extensively uses the *Bayes' Theorem*:

$$Pr(\alpha|\beta) = \frac{Pr(\beta|\alpha)Pr(\alpha)}{Pr(\beta)} \tag{2.1}$$

Where α is the cause of an effect β. Typically, the probability $Pr(\beta|\alpha)$ for an effect β given the cause α is available. Based on this information Bayes' Theorem can be used to assess the cause α given the effect β. The following example explains the benefit of this theorem. Suppose that the ECU is part of a vehicle in a network with several other ECUs. In case the vehicle is failing it is tested with a diagnostic tester. These tests are not fully reliable with a false positive rate of 2% and a false negative rate of 5%. The goal is to assess the probability for the ECU failing if the tester identifies it to be the cause of a vehicle malfunction. The probability $Pr(F)$ that the ECU fails is

$$Pr(F) = \frac{1}{1000}$$

Since the false positive rate of the diagnosis tester $Pr(T)$ is 2% it is known that

$$Pr(T|notF) = \frac{2}{100}$$

whereas the expression that in 98% the tester works correct and the ECU is not failing is

$$Pr(notT|notF) = \frac{98}{100}.$$

Furthermore, the expression for the false negative rate of the diagnosis tester can be expressed as

$$Pr(notT|F) = \frac{5}{100}$$

where the expression for 95% of the tester working correct is

$$Pr(T|F) = \frac{95}{100}.$$

With the help of Bayes' Theorem it results to

$$Pr(F|T) = \frac{\frac{95}{100} \times \frac{1}{1000}}{Pr(T)}.$$

The probability that the diagnosis tester identifies all failing ECUs can be derived from the following expression:

$$\begin{aligned} Pr(T) &= Pr(T|F)Pr(F) + Pr(T|notF)Pr(notF) \\ &= \frac{95}{100} \times \frac{1}{1000} + \frac{2}{100} \times \frac{999}{1000} = \frac{2093}{100000} \end{aligned}$$

leading to

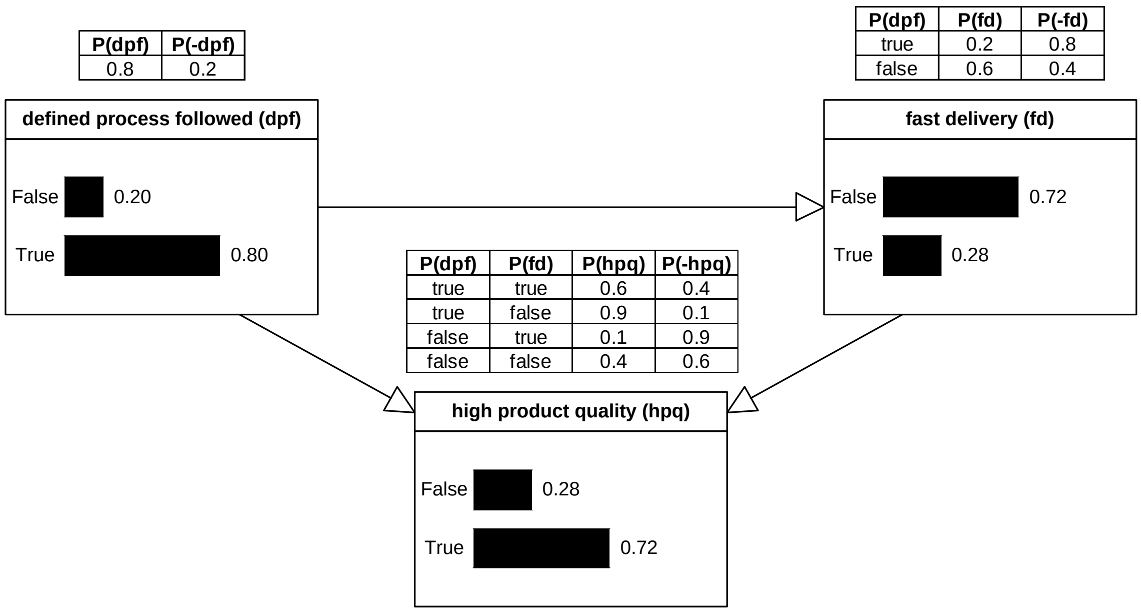
$$Pr(F|T) = \frac{95}{2093} \approx 4.5\%.$$

In real-life models with a multiple of dependant nodes, it is impossible to perform these inference calculations manually. Since the beginning of this decade, there are powerful BN tools [GeNIe, Hugin, AgenaRisk] ready for application. Decision support systems based on BNs are already in use in the domains of medical diagnosis, forensics, procurement and software engineering. 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 the various books [Darwiche, 2009, Jensen, 1997, Neapolitan, 2003, Pearl, 1988, Russell and Norvig, 2002, Winkler, 2003].

2.3.3 Propagation of Evidence

One very useful feature of BNs is their ability to run with an incomplete set of observations. This means, the user does not need to provide observations to all predictors in the model, as it is required for example in a model represented with an equation. In fact, a BN does not have a fixed list of predictors and dependant variables. The role of a variable is either a predictor or a dependent variable depending on model usage. In case the variable has an observation assigned, it becomes the predictor. Thus, it is able to predict the posterior distribution of variables where no observation is assigned.

Figure 2.3 illustrates a BN example consisting of three variables related to software projects. This model assumes that a *fast delivery* (FD) of a software product depends on the *defined process followed* (DPF), i.e. the defined process requires additional effort apart from pure development effort and thus reduces the ability of fast delivery of a software product. Both DPF and FD influence a *high product quality* (HPQ). Having a good process, gives a greater confidence in the quality of a developed product and rushing a project may cause a decrease in product quality. All variables are Boolean, i.e. a value “true” indicates a high level of intensity of a given feature and a value “false” indicates low level of intensity of this feature.



P(dpf)	P(-dpf)
0.8	0.2

P(dpf)	P(fd)	P(-fd)
true	0.2	0.8
false	0.6	0.4

P(dpf)	P(fd)	P(hpq)	P(-hpq)
true	true	0.6	0.4
true	false	0.9	0.1
false	true	0.1	0.9
false	false	0.4	0.6

Figure 2.3: Example Bayesian Network

In this example, root node DPF has the probability $P(dpf)$ of 0.8 indicating that 80% of all projects follow a defined process. In 20% of the cases the defined process is not followed, expressed where $P(-dpf)$ is 0.2. Following a defined process influences a fast software delivery. This is indicated by the arrow from node DPF to FD and expressed in the NPT of node FD. When the defined process is followed, indicated where $P(dpf)$ is *true*, there is a probability $P(fd)$ of 0.2 that the software is delivered fast and a probability $P(-fd)$ of 0.8 that the software delivery is slowed. In case the defined process is not followed, $P(dpf)$ is *false* leading to a probability for of 0.6 for the delivery to be fast and 0.4 for it to be slow. Both nodes DPF and FD have an influence on the product quality indicated by the arrows pointing to the node HPQ.

The corresponding NPT expresses this relation. For the case where the defined process is followed and a fast delivery is expected, $P(dpf)$ and $P(fd)$ is *true*. For this combination, the probability $P(hpq)$ to release a high quality product is 0.6. The case where processes are followed and there is enough time for development is depicted where $P(dpf)$ is *true* and $P(fd)$ is *false*. In this case the probability for $P(hpq)$ is 0.9. In case where the process is not followed as defined is expressed where $P(dpf)$ *false*. The probability $P(hpq)$ to release a high quality product in case the software needs to be delivered fast is 0.1. The probability to release a high quality product increases to 0.4 in case there is enough time for the development.

Another important feature of BNs is the ability of forward and backward reasoning. The direction of a link between the nodes represents the direction of statistical influence between this pair of variables, which in many cases also corresponds to their causal relationship. However, this direction does not determine the only way of reasoning. The concept of reasoning is explained based on the example illustrated in Figure 2.4 and 2.5. In the previous example illustrated (see Figure 2.3) there is no information about the model state included. In this case, no observation is entered in the model and it is calculated using prior distributions only. Therefore, the probability of a fast product delivery is 0.28 achieving a high product quality with a probability of 0.72.

For the following project it is known that the defined process will not be followed. BNs enable us to update the model based on new information as shown in Figure 2.4. Taking this into consideration, the model predicts a faster delivery $P(fd)$ with a probability of 0.6 with the consequence to release the software at lower quality where $P(hpq)$ is 0.22.

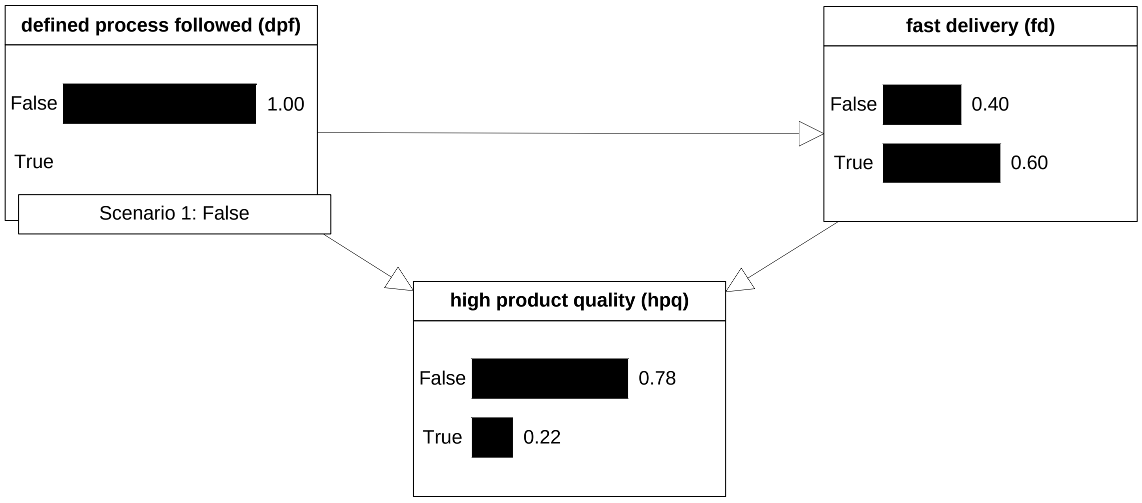


Figure 2.4: Entering observations

To demonstrate the ability of backward reasoning let us assume that high product quality becomes a constraint. Figure 2.5 illustrates the updated model. Based on the fact that the defined process is not followed but the product should be of high quality, the model predicts that more time has to be spent on the development of the product. As a consequence, the probability for a fast delivery $P(fd)$ is only 0.28.

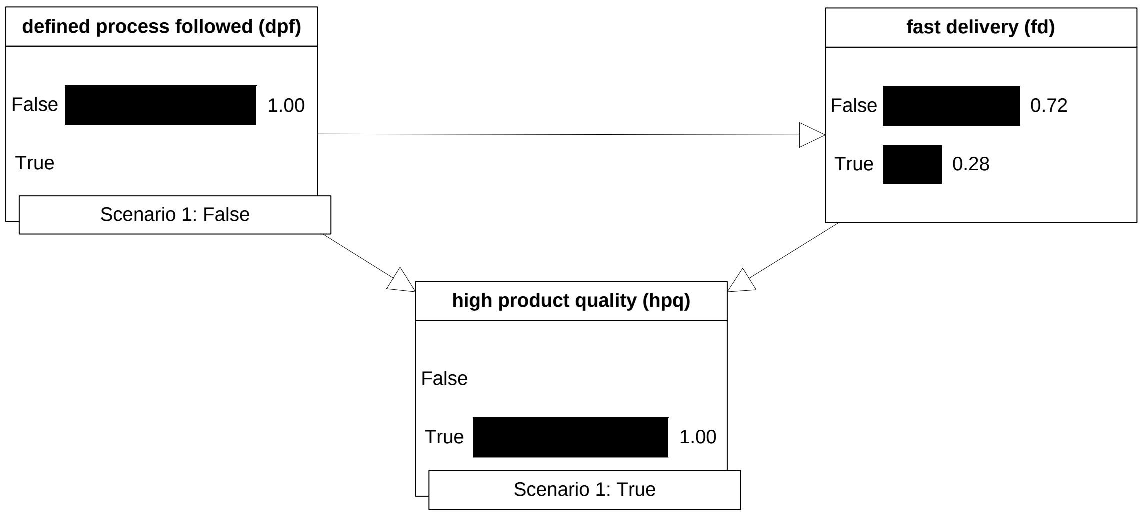


Figure 2.5: Backward reasoning

An important topic when building a BN is related to obtaining reliable prior distributions. These distributions can be assessed using either the objective approach by observing the frequencies of data in a system. The subjective approach is a measure of the personal degree of belief. Thus, it may vary among individuals leading to different predictions according to the knowledge of the expert.

2.3.4 BN Structures

Depending on the aim of an experiment, the availability of empirical data and expert knowledge, four main structures of BNs are used in modeling software engineering issues as illustrated in figure 2.6.

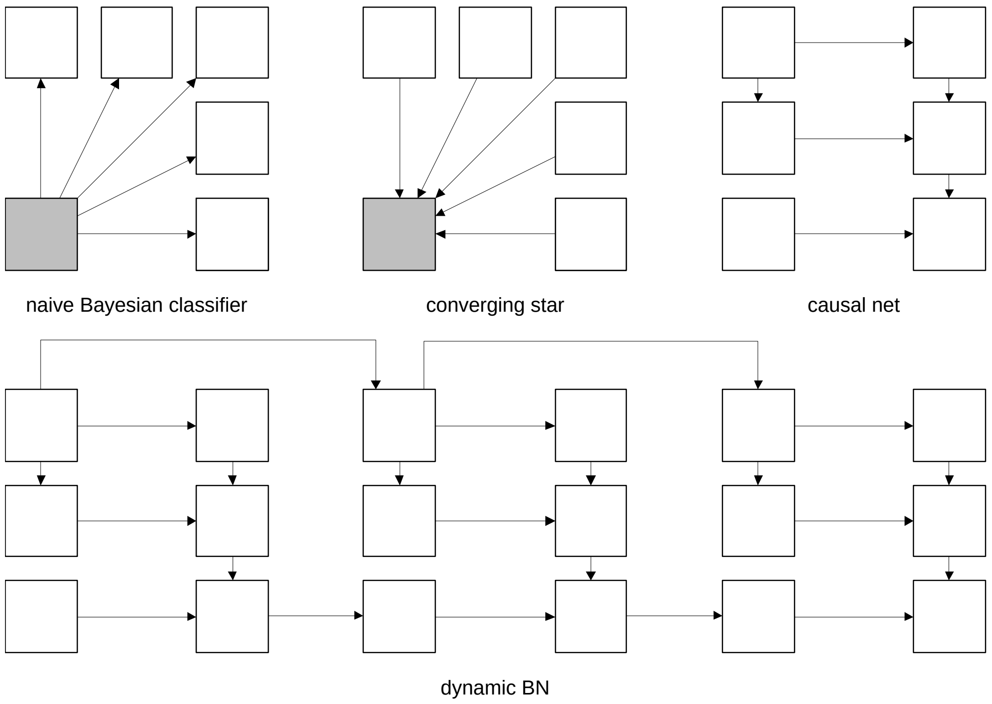


Figure 2.6: Common Bayesian Network structures

The *naive Bayesian classifier* has a structure of a diverging star with a single dependent variable in the middle of it. All links are directed from the dependent variable to each of the predictors. These models are usually built automatically, based on empirical data. Experts only select the appropriate predictors.

A *converging star* has a similar topology to the naive Bayesian classifier with the difference of having reversed directions of links from each predictor to the single dependent variable. The complexity of the definition for the dependent variable causes these models only to be built automatically based on the collected data.

Causal Bayesian Networks contain causal relationships between variables. In software engineering both structure and parameters can be built automatically based on empirical data. It is also possible to use a mixture of expert knowledge and empirical data, e.g. the structure could be defined by expert knowledge whereas the parameters are collected using learning algorithms based on data. Finally, it is also possible to define both structure and parameters using expert knowledge only.

Dynamic Bayesian Networks are a sequence of causal BNs where each individual BN reflects the state of the system at a specific point of time. With *Dynamic Bayesian Networks (DBNs)* it is possible to use the results of one BN as input for another BN.

The models presented in this thesis are based on causal BNs and DBNs. Both existing models extensively use the notion of *ranked nodes* [Fenton et al., 2007b]. Ranked nodes are used to define qualitative variables which can be expressed on a ranked scale. These ranks can be ordered e.g. depending on the degree of belief on a 5 point scale from *very low* to *very high*. The range of a ranked node is defined internally as the interval between zero and one. This range is divided in as many equal-width intervals as there are labeled states where every state has a label assigned. During the calculation process, ranked nodes are treated as continuous nodes when 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. The ability to use various weighted expressions, such as weighted mean, weighted max, weighted min [Fenton et al., 2007b] drastically simplifies the process of defining NPTs.

2.4 State of the Art

BNs in the domain of software engineering have been used mainly for effort and quality prediction. Extensive analysis of BNs models developed for effort and quality prediction have been performed in a variety of studies. There are different approaches used for building the BNs presented in table 2.1. Some are generated automatically from data without expert input. Others do not have a causal structure. Furthermore, the level of detail varies among those approaches. This is due to their different analytical purpose including their point of view. Only a few studies are directly related to this thesis. Bibi and Stamelos [2004] 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. This concept seems to be clear and intuitive. However, the authors have not yet performed any validation and only published a basic topology of the model. Thus, a detailed analysis of their model is difficult.

Fenton et al. [2004] discussed making resource decisions for software projects. They discussed the problems with traditional metric approaches, i.e. regression based models. One possible solution to this is the use of causal models like BN. In their work they pointed out the need to incorporate empirical data as well as expert judgement within a single model. Based on the resulting model, they explained how managers can use their approach for decision support. The model has evolved in a number of collaborative projects and was validated as part of the EC funded project MODIST.

Fenton et al. [2008] analyzed the effectiveness on the early life cycle defect prediction with BNs. The resulting model predicts the number of residual defects after development, i.e. the number of defects to be found during testing and in the field. Since their method does require detailed domain knowledge it can be applied early in the development life cycle. The resulting model incorporates quantitative and qualitative factors describing a project and its development process. These factors, including the relationships between them, have been identified as part of a major collaborative project. A dataset on multiple software projects was taken to validate the results. The model assumes that effort required to fix a defect is constant, i.e. that it does not depend on the defect itself. This assumption is valid only when proportions of different defect types are constant among multiple projects and components. Radlinski et al.

[2008] developed a DBNs where each instance reflects a testing and rework iteration. The model aims to predict the number of defects remaining after each iteration and thus may be used to estimate the release time. This model is restricted to be used in situations where no functionality is added to the code during the testing and rework phase. The predictions are based mainly on the amount of effort, process quality factors and amounts of defects found and fixed. This model has been validated using a set of synthetic data.

The following table 2.1 gives an overview on further related research activities in the fields of BNs in software engineering.

Table 2.1: Summary of recent BNs in software engineering

Source	**Main problem investigated**
Abouelela and Benedicenti [2010]	productivity, duration, quality
Bibi and Stamelos [2004]	effort
Dabney et al. [2006]	defect rate
de Melo and Sanchez [2008]	maintenance delays
del Sagrado and d'Aguila [2010]	need for requirement review
Fenton et al. [2010]	development process support
Fenton et al. [2008, 2007a]	defects
Fenton et al. [2004]	size, effort, quality
Hearty et al. [2009]	project velocity
Radlinski et al. [2007]	project resources
Radlinski et al. [2007], Radlinski [2008]	size, effort, quality
Schulz et al. [2010b,a, 2008]	effort
Stewart [2002]	effort, productivity
Torkar et al. [2010]	effort
Wagner [2009]	various aspects of software quality
Wang et al. [2010]	variance of project schedule
Wooff et al. [2002]	testing process

3 Development Methodology

This chapter describes the methodology followed to develop the models presented in this thesis. Every research stage, from defining the problem to validating the model is described along the procedure model for designing expert systems.

3.1 Procedure Model

The procedure model follows the guidelines to designing expert systems [Weiss, 1984]. Their main application is the support of *Project Managers (PMs)* and process owners on decisions related to *Defect Correction Effort (DCE)* estimation. Figure 3.1 illustrates the procedure used to build the models.

Essential for every expert system is the problem definition and the key performance indicators (KPIs) related to it. The problem definition of the models described in this thesis are derived from the research goal defined earlier (see section 1.2). The *Goal Question Metric (GQM)* approach [Basili and Rombach, 1988] has been used to systematically identify relevant KPIs of the model and their relations. According to the GQM approach, the problem definition includes its purpose, object of interest, issue and user's point of view. Based on the problem definition KPIs and their relations are used to build the causal structure of the model.

Model data has been obtained mostly from internal sources within the specific automotive engineering environment. Main data source here is the change & defect management system where all change specific information is stored. A change entry summarizes all developed artifacts needed to realize a specific part of a feature. Further internal sources are process documentation as well as expert knowledge. Data used for model scenarios has been made anonymous due to their confidential-

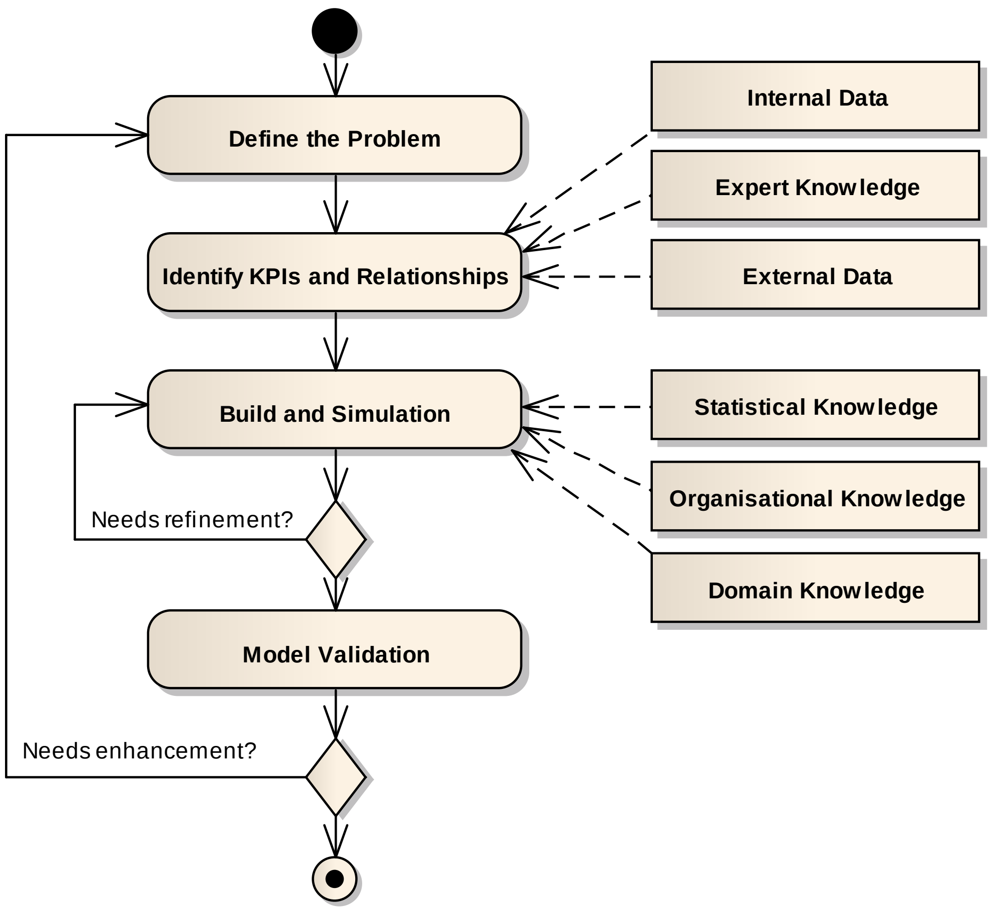


Figure 3.1: Research procedure

ity. 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. It is software engineering for the models presented in this thesis.

Second, the problem under discussion has to be fully understood including its *Key Performance Indicators (KPIs)* and how they are related to each other. In this case this means being an expert in the specific automotive work environment and being able to map the problem definition from theory to practice.

Finally, statistical know-how is required to understand the consequences of combining data from various sources and how it affects the simulation results. Multiple iterations were needed to design and calibrate the models to fulfill all requirements from the fields mentioned above. After this, multiple scenarios are defined to reflect different aspects of the problem definition for the final analysis. The final step is model validation. According to the definition of the problem the predefined scenarios of the *Bayesian Network (BN)* model are analyzed and assessed by experts.

Due to the novelty of this approach, it was not possible to validate all of the simulation scenarios. Instead, domain experts analyzed and evaluated the model according to the definition of the problem.

3.2 Statement of the Problem

The requirements were developped to support the PMs' decisions. The system reflects the development process and software structure. The following major decisions were identified:

- Can a specific software be released based on its defect rate?
- Can the development for new software be started with the given specific projects boundary conditions ?
- Can a specific process area be optimised to decrease the error rate for artifacts being developed here?

Every decision requires different indicators related to it. Therefore the decision support system provides all the information needed for the specific decisions. Different aspects of a single decision need to be taken into account:

- How likely is a specific error rate for the project achieved when implementing a function of a certain complexity with given resources?
- How many resources are needed to develop a certain functionality given the project's boundary conditions?
- What is the risk of delivering malfunctioning software to the customer given a specific software release?
- Which are the process areas where the lowest optimization effort results in the highest product quality?

Consequently, the statement of the problem is to build a decision support system, which represents the structure of a specific software development process and a specific software product. The system needs to point out the different aspects of a decision. It has to combine these results to support the decision maker in this specific decision. It needs to be able to consider process KPIs as well as expert knowledge or a combination of both.

To solve the problem stated, experts were consulted, in this case decision makers. They were able to identify the most critical decisions, where a support system could supply relevant information. Experts help to describe the

1. definition of the problem,
2. KPIs to be taken into account,
3. causal relationship of these KPIs,
4. strength of relation between the KPIs.

Decisions as well as how KPIs lead to a specific decision can be provided by experts. These are managers who have to come to a decision as a part of their daily work. KPIs with high relevance for a specific decision can be identified in databases, e.g. process or product metric databases. Also the relation of KPIs among themselves as well as their relation to other indicators of the system can be elicited from such databases. Furthermore, it is possible to get quantitative descriptions by explicitly designing experiments. If this is not possible, a qualitative description of such relationships can still be done with the help of experts to parametrise the model. Finally data from the research community, e.g. from Promisedata [Promisedata] or Cocomo [Center for Systems and Software Engineering, 1995] can be used as expert knowledge. This leads to the following sources to obtain expert knowledge from:

- Specific experts
- Specific product and process metrics
- Specific experiments (quantitative description)
- Specific questionnaires (qualitative description)
- Publically available models
- Publically available databases
- Publically available research literature

3.3 Building and Simulating

The design of the system itself is based on requirements resulting from the statement of the problem and the consultation of human experts. Every decision to be supported should be assigned to a high level domain. If the system needs to support decisions from multiple domains it might be necessary to create multiple domain related specifications. Interfaces, meaning the exchange of information over multiple domains are described in the domain specification. The creation of a new domain might also be necessary in case the reuse of indicators among multiple decisions is low. This encapsulation of supported decisions increases the overview and especially the performance of the overall system. In contrast to this, the more indicators are used in a single domain the more consistent decisions are supported by the system. In every domain all decision are specified together with the indicators leading to the particular decision.

A further aspect considered when designing the expert system is the later verification of the system described in subsection 3.4. Not only KPIs but also relationships to other indicators must be verifiable. KPIs and relationships should be meaningful and measurable ideally extractable from either process or product metrics.

To model a complex structure such as an software engineering environment we needed a method supporting the following aspects

- Combining diverse types of data for a single decision.
- Reasoning from any aspect of the network to another.
- Making predictions with incomplete data.
- Quantifying the uncertainty of a supported decision.

These requirements towards the development tool are typically not supported by traditional modelling techniques.

Based on the domain specification a first prototype model has been created in an iterative process where build cycles are planned explicitly after *Testing the Prototype* and *Refinement and Generalisation*.

3.4 Model Validation

To test the prototype different validation techniques [Leary et al., 1990] have been considered to cover different aspects of validity. Problems found during testing which do not affect the system's specification are reworked immediately. Validation techniques cover *High Level Validity*, *Subsystem Validity* and *Input-Output Comparison* described in the following subsections

This step of validation ensures that the structure of the model allows to support the decisions which have been specified during *Design of the Expert System*. It is a high level validation assuring that the model supports an expert as he expects it to be. The model should exactly address the *Statement of the Problem* as described in the beginning of this section.

The next step of validation focuses on the internal validity of the model. Therefore the model is divided into logical groups with functions where every group can be defined on its output by a function of its input. It is defined how a specific group should respond as part of the system. Whenever possible existing data should be taken to stimulate the group and to validate its reaction on the stimulation. In case there is no data available for validation experts need to define the function of the group. Experts and the systems designers need to agree that the model's major assumptions are specified according to the statement of the problem.

Finally, the model is tested as a whole. Comparable to the subsystem validity, the overall decision support system can be seen as an output generator based on a function of its input. For every decision which had been specified as part of the system design a set of input and output data are compared to how an expert would respond to the specific input data. There are different techniques available for input-output comparison [Shannon, 1975] [Sargent, 2005]. When validating an expert system, both quantitative as well as qualitative validation techniques are required. Classical goodness-of-fit tests are taken where large amount of data sets are available. In areas of the system where observations and controlled data with statistical relevance are not available, subjective methods need to be taken, for example the *Turing Test* where multiple experts define the appropriate sets of input and output data to validate the system's behaviour.

3.5 Refinement and Generalisation

To finalise the model prototype it is necessary to refine it. Also parts of one domain of the system might be of interest in an other domain. Generalisation techniques can be used in this case. Further calibration and fine-tuning as well as correcting faults should also be applied in this step. After planning the refinement the prototype is rebuild.

Feedback of experts and users as well as major corrections and enhancements are integrated in this step to create another iteration for building the expert system.

4 Model Data Chapter

This chapter describes the data base for the models presented in this thesis. The data set has been collected from a historical project for the development of automotive applications. Thus, it does not represent the typical defect rates of the automotive industry but focuses on a single project with specific boundary conditions. The data set is described in the *internal data* section. The data base provides insights to specific defect rates of the project as well as the development process. During the data analysis, the *feature classification* has been introduced to distinguish areas of the software product at different defect rates. Furthermore, the *Defect Cost Factor (DCF)* is defined to quantify the impact of defects on the software development. In addition to that, this chapter introduces an experiment that has been conducted to understand the impact of major influencing variables.

4.1 Overview

The data used for model building, calibration and validation has been collected from the following domains:

- Internal process data from the automotive development process described in section 2.1. It is based on change & defect management (CM) information of a specific component used for developing a specific customer project. This data contains information about every *change*, describing the development of a specific requirement. A change summarizes all developed artifacts needed to realize a specific part of the software product. It is part of the change management process. Data has been collected from the beginning to the end of a single customer project. Further internal sources are process documentation as well as expert knowledge. It has been made anonymous due to their confidentiality.

- Expert knowledge from project managers and stakeholders where no internal data is available, e.g. on how process maturity and process activity influence the rework of a project.
- Data based on an experiment carried out to identify the influence of specific *Key Performance Indicators (KPIs)* on the *Defect Correction Effort (DCE)* as a part of a separate master's thesis [Wuchenauer, 2009].
- External data from literature.

4.2 Internal Data

Internal data is collected based on a measurement concept describing what types of data are collected and how they are related. This data is used to describe the following characteristics of the engineering process.

- Feature volatility indicated by a feature's requirements change rates.
- Type of task as the category for changes.
- Effort needed for every type of task, change, defect and defect correction.
- Defect analysis and defect correction as specific type of tasks.

Internal data is collected in two different management systems. The *requirement management system* (RM) is used for managing and developing requirements. The *change & defect management system* (CM) is needed for managing changes. The CM system is a standardized database system used for the management of all product related changes and defects. It holds information about the development of artifacts which are needed to realize a specific part of a feature. Traceability is supported both from requirements to source code as well as from source code to requirements. Furthermore, it is one major metric source of metrics to support the assessment and improvement of the development process. Entries in the CM have one of the following categories:

- Change entries describe the development of a newly added, modified or removed requirement. They hold information about the development effort needed for this change.

- Defect Analysis entries describe a defect in an existing feature. They store 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 entries describe what changed to remove the defect. It stores information about the correction effort.

Figure 4.1 illustrates the data measurement concept. Every node has a relation to the *feature* it describes. A feature has at least one *requirement* related to it, whereas every requirement is related to exactly one feature. The nodes *change*, *defect analysis* and *defect correction* also relate to exactly one feature. There is at least one change related to a feature in the sense that the measurement concept needs at least one data point to be applied to. Features might have zero defects and therefore no defect correction nodes related to it. But if a feature has defects there is always a defect analysis node describing the defect and, in case it can be fixed, a defect correction node describing what needs to be fixed to remove the defect.

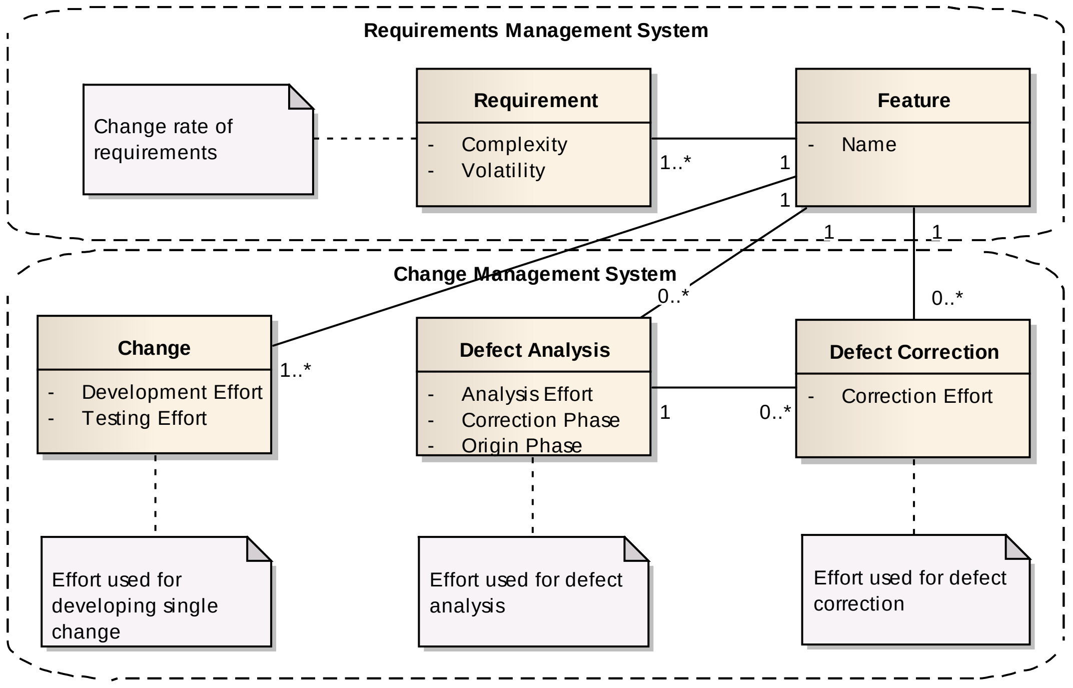


Figure 4.1: Measurement concept

4.2.1 Project Data

The internal data set has been collected from a single completed project. It holds entries with information about the *type of change*, whether it is considered as a change describing development- or defect correction activities. Furthermore, each of these entries holds the information about the effort needed to complete the change, described as *development effort* or DCE. In addition to that, every change has an attribute describing the relation to the *feature* it realizes.

Change Categories

Every entry has a *type of change* attribute. The categorization of changes is made based on its purpose. It is either related to the development category where functionality is added, removed or changed. Or the change is of type defect correction where corrections have to be made due to a defective feature. The categorization of changes is defined as follows:

- *small* for changes where development effort <= 10 hours.
- *medium* for changes where development effort > 10 and <= 30 hours.
- *large* for changes where development effort > 30 hours.
- *other* is used for unclassified changes, e.g. architectural changes.
- *bugfix* describing a change for defect correction.

Change Distribution

Every change holds unique information about the effort needed for its completion. Figure 4.2 gives an overview of the data set. Considering the number of change entries, there are 242 development changes and 170 bugfix changes. Regarding the effort, there are 2045 hours of development effort and 1202 hours of defect correction effort.

Table 4.1 summarizes the data set. There are 412 changes, including both, changes for development as well as for defect correction. It shows an overall effort of 3247

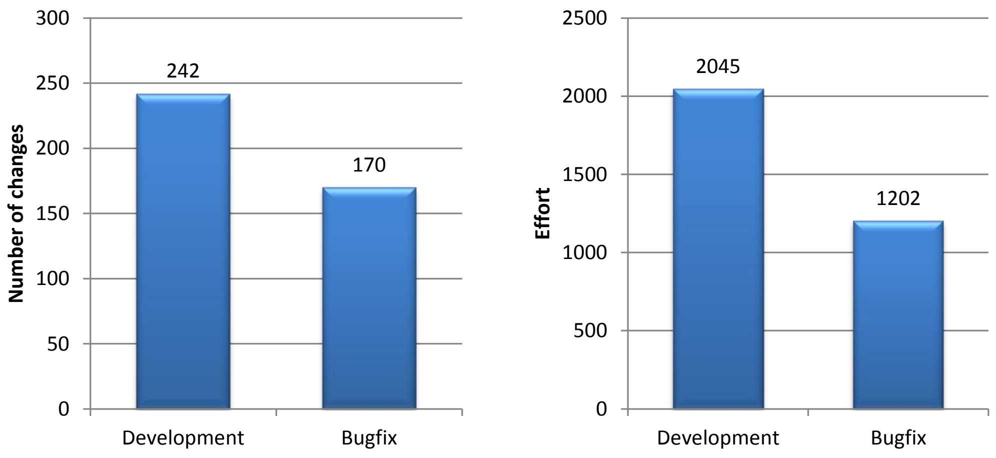


Figure 4.2: Overall effort

hours for both development and defect correction changes. This results in an average effort over all changes of 7.88 hours. The number of development changes is 242 representing 2045 hours of feature development. Here, the average development effort per change is 8.45 hours. For bugfixes, there are 170 of change entries in the data set. They represent an overall of 1202 hours of defect correction resulting in an average DCE of 7.07 hours per change.

Table 4.1: Development and defect correction effort

Overall number of changes	412
Overall effort	3247 hours
Avg. overall effort per change	7.88 hours
Number of development changes	242
Development effort	2045 hours
Avg. development effort per change	8.45 hours
Number of bugfix changes	170
Defect correction effort	1202 hours
Avg. DCE per change	7.07 hours

Figure 4.3 depicts the distribution based on the number of changes whereas Figure 4.4 shows, for the same data set, the distribution of effort.

Regarding the number of changes, the data set contains 164 small changes, 35 medium changes, 8 large, 35 other and 170 bugfix changes. Accordingly, the distribution of changes consists of , 40% small, 9% medium, 2% large, 8% other and

41% bugfix changes.

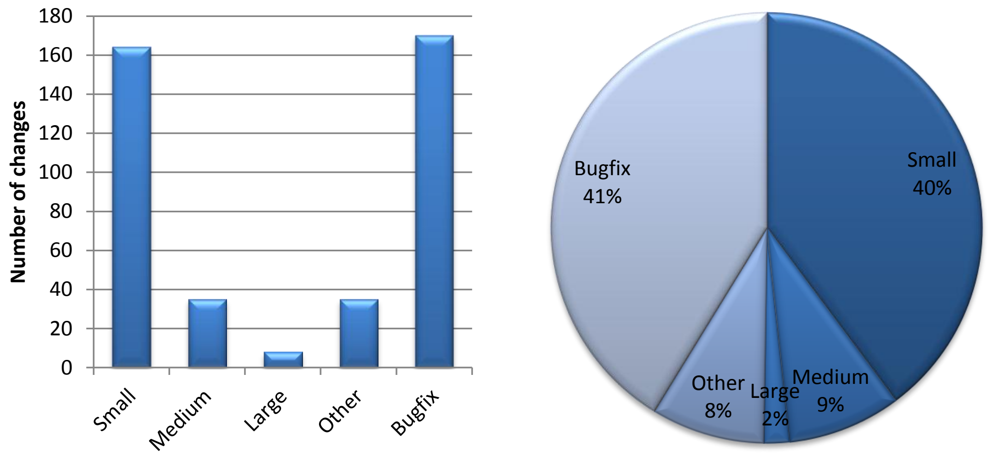


Figure 4.3: Number of changes

Considering the effort, the distribution changes. There are 805 hours for the development of small changes, 587 hours for medium changes and 377 hours for large changes. Other changes have 276 hours. For defect correction, there are 1202 hours. The percentage distribution of effort results in 25% for small changes, 18% for medium changes, 12% for large changes. 8% for other and 37% for bugfix changes.

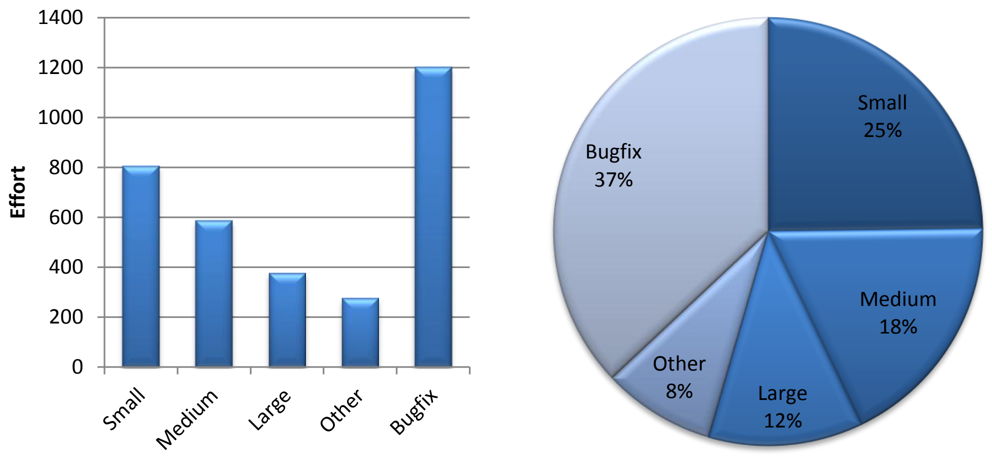


Figure 4.4: Effort distribution

The difference between the average number of types and the average effort per type is summarized in Table 4.2. Small changes reflect 40% of the data set entries but only 25% of the overall effort resulting in a deviation of 15%. Medium changes represent 9% of the overall number of changes but reflect 18% of the overall effort. For medium changes, the percentage deviation is 9%. Large changes represent only 2% of the overall number of changes whereas their effort reflects 12% of the overall effort. Their percentage deviation is on 10%. Finally, for other changes the deviation is 0% because both, number of changes and effort are at 8%.

The bugfix cateogry represents 41% of the data set entries. Considering effort, it reflects 37% of the overall effort. This results in a percentage deviation of 4%. If number of defects would have been used within the models to predict potential defect correction effort, this deviation would represent a potential calcualtion error within the models. In other data sets, the deviation could be even higher, depending on the other categories.

Table 4.2: Type distribution error

Type	No. of changes	Effort	Deviation
Small	40%	25%	15%
Medium	9%	18%	9%
Large	2%	12%	10%
Other	8%	8%	0%
Bugfix	41%	37%	4%

Considering this thesis goal to optimize the DCE, effort is used instead of the number of changes to describe the models.

4.3 Feature Classification

Every change entry describes a modification of a specific part of the software. It is always related to specific functionality of the software product, e.g. of a bus interface. These functional groups are called *features*. Every entry of the data set belongs to either one of the following feature:

- *CAN*. Interface to the vehicle's network.
- *HMI*. Interface for the driver.
- *PAD*. Parameter dispatcher for variable handling.

- *VDB*. Component interface.
- *CNV*. Central number and versions.

The optimization of rework in software development requires an assessment of the DCE. It is known that specific parts of a software product have lower defect rates than others. Features are used to distinguish these areas of the software product with different defect rates 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 a defect occurring when implementing parameter changes is lower than for using a complex state machine to realize *Human Machine Interface (HMI)* logic.

4.4 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 is often expressed as the number of defects but this is only an indicator for the actual rework spent to fix product requirement deviations. As a single measure it is not capable of quantifying the amount of rework, e.g. a simple defect might be fixed in hours, whereas another defect might take days for analyzing and fixing. Furthermore, defects detected in later development phases, e.g. in *Integration & Test (I&T)*, leading to a change in early phases, e.g. *Requirements Engineering (RE)* or *Design (DE)*, are more cost intensive than defects detected in development phases where they originated.

In this thesis, the potential DCE is used 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 following definition of the DCF:

$$\text{Defect Cost Factor} = \frac{\text{Defect Correction Effort}}{\text{Development Effort}} \tag{4.1}$$

Where:

- DCE is effort needed for defect correction (applies to all changes of category "bugfix").
- Development effort is effort needed for implementation (applies to all changes of categories "small", "medium", "large" and "other")

This relation implies, with increasing development effort the potential effort needed for later defect correction also increases. For the data set used in this thesis, the DCF results to

$$\text{DCF(Overall)} = \frac{1202}{2045} = 0.59 \tag{4.2}$$

Where an overall DCF of 0.59 represents the potential product defect rate. Therefore, for 100 hours of development effort, the potential DCE is expected to be 59 hours.

To express different error rates for different features, this thesis defines a separate DCF for every product feature. Figure 4.5 illustrates development effort in relation to DCE for every feature.

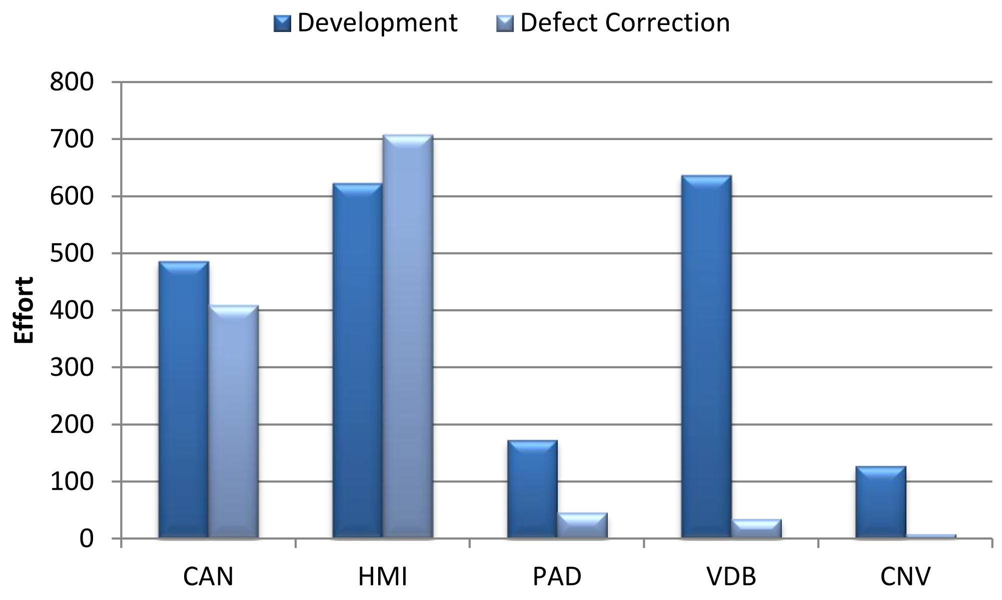


Figure 4.5: Effort over feature

The figure shows an unequal distribution of DCE. Features CAN and HMI have a high

DCE in relation to their development effort whereas the DCE for other features are lower. The comparison also shows nearly the same DCE for features PAD and VDB.

Table 4.3 depicts the DCF for every feature. For feature CAN, the development effort is at 486 hours. The DCE is 408 hours resulting in a DCF of 0.84. For feature HMI with a development effort of 622 hours and a corresponding DCE of 707 hours, the resulting DCF is 1.14. This is due to HMI development is far more complex than e.g. maintaining a feature where only version numbers are stored. For feature PAD, the DCF is 0.26 resulting from 173 hours of development effort in relation to 45 hours of DCE. Feature VDB has the lowest DCF of 0.05 based on 637 hours of development and only 34 hours of defect correction. Feature CNV is at a similar DCF of 0.06 based on 127 hours of development and only 8 hour of defect correction.

Table 4.3: Defect Cost Factor over feature

Category	**CAN**	**HMI**	**PAD**	**VDB**	**CNV**
Development effort	486	622	173	637	127
Defect correction effort	408	707	45	34	8
DCF	0.84	1.14	0.26	0.05	0.06

4.5 Experiment Data

In [Wuchenauer, 2009], a *Design of Experiment (DOE)* has been conducted to quantify KPIs needed for the estimation of DCE. The experiments were carried out with 17 participants, both students of the University of Stuttgart and professionals from the Robert Bosch GmbH.

The DOE's main focus was to analyze the influence of deadline pressure on software defects. The experimental setup was specific to the automotive engineering domain. Every participant had to solve a typical engineering problem from this domain.

The results for the influence of deadline pressure on the number of software defects are illustrated in Table 4.4. Regarding the median for every deadline pressure level, the results indicate an increase of defects for medium and high deadline pressure.

Table 4.4: Number of defects over deadline pressure

Measure	**Deadline pressure**		
	High	**Medium**	**Low**
Minimum	1	0	1
25% quartile	3	4.5	1
Median	4	6.5	1
Mean	3.83	5	2,4
75% quartile	5	7	4
Maximum	6	7	5
Standard deviation	1.83	3.37	1.95
Variance	3.37	11.33	3.8

The strong statistical variation is due to the small amount of participants. However, the results are taken as indicators for the statistical distribution.

5 Software Process Model

This chapter presents a methodology to estimate the amount of defects for a specific software product. It focuses on an estimation of defects based on the changes introduced in a specific software release cycle. The resulting model is called the *Software Process Model (SPM)*. The *overview* section is followed by the main goal of the SPM based on which the model's *Key Performance Indicators (KPIs)* are identified. After this, the section on *building and simulating* describes the actual build of the model. It is followed by a section on the *model usage*. After that, scenarios are defined, analyzed and presented as part of the *results* section. Finally, the *conclusion* section summarizes this chapter and discusses the results.

5.1 Overview

The most common way of inserting defects into a software product is by applying changes to it. In every software release, hundreds of changes are applied to the software product. Every change has its unique characteristic based on process and product factors. With regard to these factors, every change has its own probability of injecting a defect.

The SPM is mainly designed to optimize the *Defect Correction Effort (DCE)* over time. It enables to assess measures of the process influencing the DCE. Due to the dynamic structure of the model, it further allows to analyze the impact of possible process and product changes over time. These capabilities give various views to project managers on the current and future software product that are discussed at the end of this chapter to highlight the practical benefit of SPM.

The SPM is a *Dynamic Bayesian Network (DBN)* with capabilities to represent multiple change characteristics of a specific software release. The dynamic structure represents the successive development of changes as part of a software release life cycle. For every change a single DBN is used to estimate its development effort and DCE. Hereby, the SPM enables project managers to oversee the impact of every change made within this software release while taking into account the individual change characteristics. A DBN is a model forming a sequence of individual *Bayesian Networks (BNs)* where each individual BN reflects the state of the system at a specific point of time. Databases on historical and current projects are used to validate the results. The resulting model was assessed by process and project experts for predictive accuracy and usability.

The following sections describe how SPM was set up and what the simulation results were. According to the methodology described in the previous chapter 3, this chapter starts with the statement of the problem. This is followed by a description of SPM's main variables and how they are related. The main section 5.4 focuses on the creation of SPM. It describes the structure, variable calibration and gives details about the setup of the simulation environment. The final sections 5.5 and 5.6 focus on model validation followed by a result discussion.

5.2 Problem Definition

SPM has been developed to estimate DCE in relation to the development effort for a specific feature of a software product, e.g. the *Human Machine Interface (HMI)*. SPM is intended to be used in a number of analyses to support project managers planning or assessing their software project. The major advantage of SPM is its capability to take KPIs into account having a significant influence on the estimation variables, i.e. development effort and DCE. It incorporates product, project and process information in form of objective data (e.g. process metrics) and subjective data (e.g. expert knowledge) in a causal relationship.

KPIs used in SPM can be divided into five categories:

1. *Process activities* represent the influence of process activities, such as process maturity grade and its degree of implementation.

2. *Product factors* such as quality of documentation and further engineering artifacts.
3. *Project factors* are project constraints which may vary over time.
4. *Change factors* are related to the feature to be developed.
5. *Test activities* incorporate the amount of tests carried out as well as how effective tests are.

These categories for KPIs hold the core parameters of SPM. They are identified to have major influence on the estimation variables, especially on a change's error-proneness and consequently to the DCE of the overall software product. SPM focuses on additional rework in form of the DCE because it has the highest influence on the overall effort, apart from the the development effort itself. The influence of the DCE on the overall effort needed for a specific feature has been discussed earlier in chapter 4. This leads us to the following problem definition:

Estimate the engineering effort as sum of development effort and DCE of every change for a specific feature in a specific software development process from a project manager's point of view.

Figure 5.1 depicts the process of effort estimation as a part of the software release planning which the SPM's structure is based on.

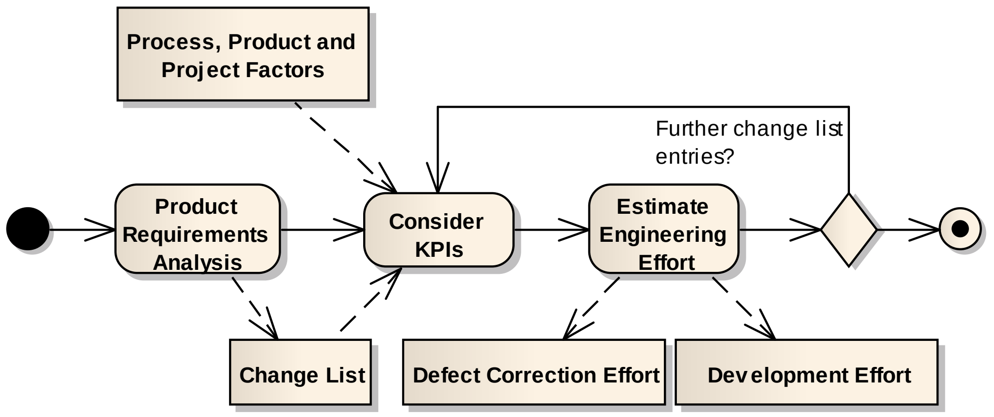

Process, Product and Project Factors
Further change list entries?
Product Requirements Analysis
Consider KPIs
Estimate Engineering Effort
Change List
Defect Correction Effort
Development Effort

Figure 5.1: Effort estimation

The process starts with analyzing product requirements for a specific feature. It is followed by the creation of a change list to define work packages for a specific release.

This change list is used by project managers for planning and tracking as well as by developers assigned to implement specific change. For every change of the change list, KPIs are considered to estimate the engineering effort as a sum of development effort and DCE.

5.3 Identifying KPIs and Relationships

The *Goal Question Metric (GQM)* approach has been used to systematically identify relevant KPIs of the model and their relation. According to GQM, KPIs are derived from the problem definition which is broken down into its major components for further characterization. Questions are used for this refinement. Questions which also project managers might ask to understand the problem being assessed. For every question there is a set of metrics characterizing it. Metrics represent a model to measure properties of any engineering artifact, KPIs in SPM respectively. The following enumeration holds a set of *questions* (Q) along with their *metrics* (M) characterizing the goal of SPM. All questions are related to a single change implementing a specific feature.

- Q1 What is the difficulty of a feature?
  - M1 *feature difficulty*. How difficult is it to realize?
  - M2 *feature volatility*. How often are requirements changed?
  - M3 *feature complexity*. How complex are the requirements of a specific feature?
- Q2 What is the engineering effort?
  - M1 *engineering effort*. Effort used to develop a single change including effort spent on test activities.
  - M2 *type of change*. Category of a specific change describing its complexity.
- Q3 What is the potential defect correction effort?
  - M1 *defect correction effort*. Potential effort to fix a deviation according to the requirements.

- M2 *development effort*. Effort to implement a single change in hours.
- M3 *defect cost factor*. Factor indicating a feature's error-proneness.
- M4 *defect correction effort after testing*. Residing defect correction effort after finishing a change.

Q4 What is the potential defect correction effort?

- M1 *testing effectiveness*. How many defects can be found by testing activities?
- M2 *effort spent on testing*. Effort used for testing after implementation.
- M3 *test ratio*. Amount of testing as a proportion of development effort.
- M4 *test payoff*. Defect detection potential. How good are test activities established?
- M5 *defects found*. Reduction of potential defect correction effort.

Q5 What are product factors?

- M1 *product factor*. Existing product as a base for further development.
- M2 *documentation quality*. Understandability of the product base.
- M3 *code quality*. Ease of understanding, changing, enhancing the existing code base.

Q6 What are project activities?

- M1 *deadline pressure*. Amount of pressure put on the engineering team to finish a change within a specific time frame.

Q7 What are process activities?

- M1 *process activities*. Supporting measure for less error-prone engineering.
- M2 *process organization*. Maturity level of the organizational process definition.
- M3 *process application*. Implementation of maturity level.

5.4 Building and Simulating

The actual build and test stages require expertise from multiple fields of knowledge, which is the main reason making the setup of SPM so challenging. On the one hand the model designer has to be a statistician knowing how to collect data and how to represent this data in a cause and effect chain within a BN. On the other hand he has to be fully aware of the problem to be solved by the model. A software engineering expert in the SPM domain has to be able to describe how defect correction effort is influenced by a software engineering process. Furthermore, the model designer needs to know the organizational structure in detail. How is the implemented software engineering process established, what kind of data is available and where to acquire it? For every KPI, a mapping of real world data to model data has to be done. The model designer needs to know what the meaning of the data is and which relevance it has for SPM.

SPM offers the possibility to assess characteristics of a feature's defect correction effort in relation to its influencing factors. It is even possible to calculate different what-if scenarios to compare alternatives to the current model settings. The change model in form of a BN illustrated in Figure 5.2 represents the central part of SPM. It depicts causal relationships represented by the arcs among key performance indicators represented by its nodes. Strong borders indicate the calibration nodes, used by the simulation environment to adjust the model according to the simulation conditions. Model data is elicited based on internal data, expert knowledge and external data whereas internal data has been made anonymous due to their confidentiality. All KPIs and relations described in the previous section provide the basis for building the model. There are five main categories in the change model: change factors, test activities, product factors, process activities and project factors.

5.4.1 Calibration

After building the model it can be used for test runs. Typically models like SPM need calibration until they perform as expected. From simulation perspective Figure 5.3 shows how the SPM is set up. It is crucial to every form of effort estimation to calibrate

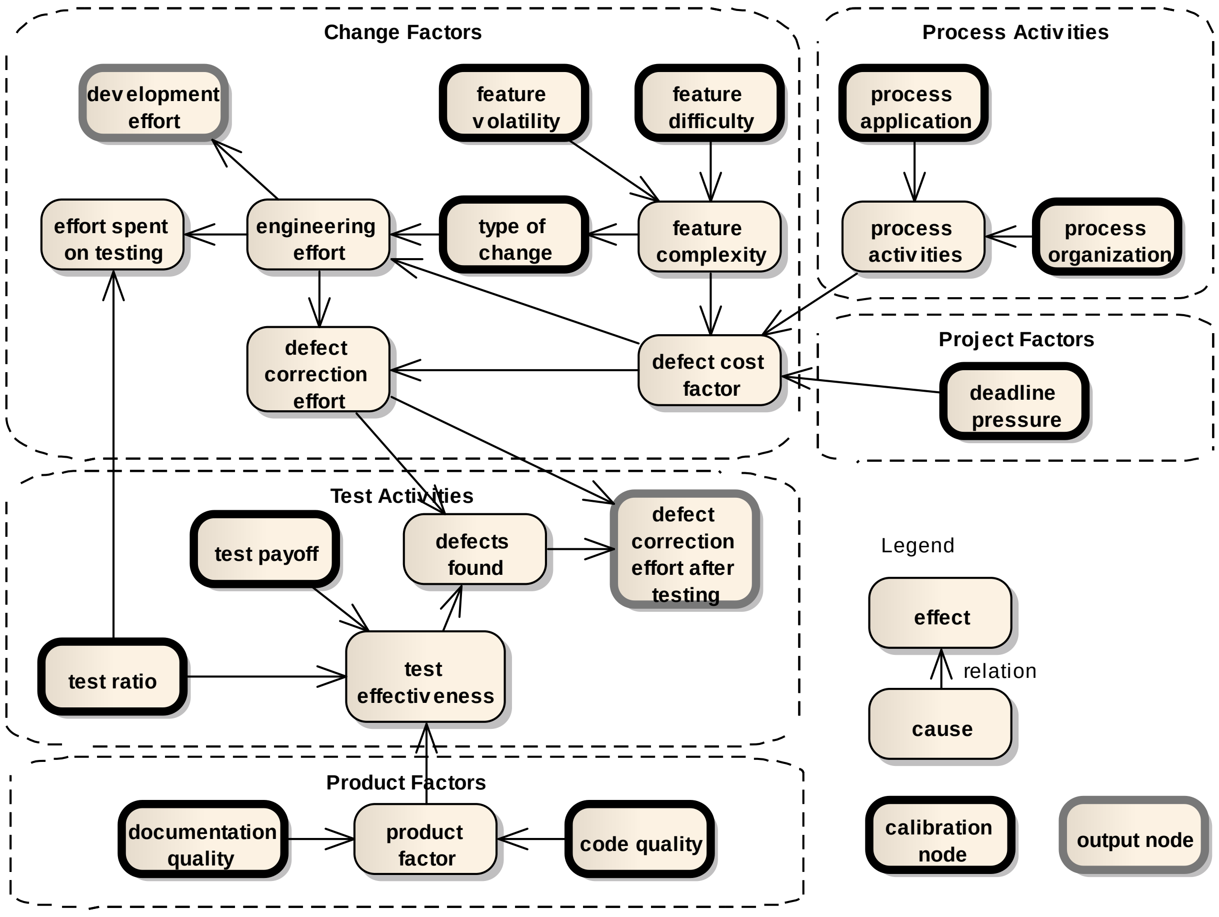


Figure 5.2: Change model BN

the KPIs influencing the development effort of a specific product feature. Therefore *product requirement analysis* acts as an entry point to SPM's setup to categorize the feature to be assessed according to its feature complexity. Product features are assigned to one of the predefined complexity and volatility levels. *Historical project data* is used to support the classification process. In typical engineering environments this data can be derived from historical project data but could also be obtained based on expert judgment. If a feature has already been developed in another project the former distribution of changes can be taken as basis for the current estimation and adjusted by an expert according to their degree of belief. The feature might also be similar to another feature already implemented in the past. In this case the former distribution could serve as a reference distribution list and be adjusted according to its new characteristics.

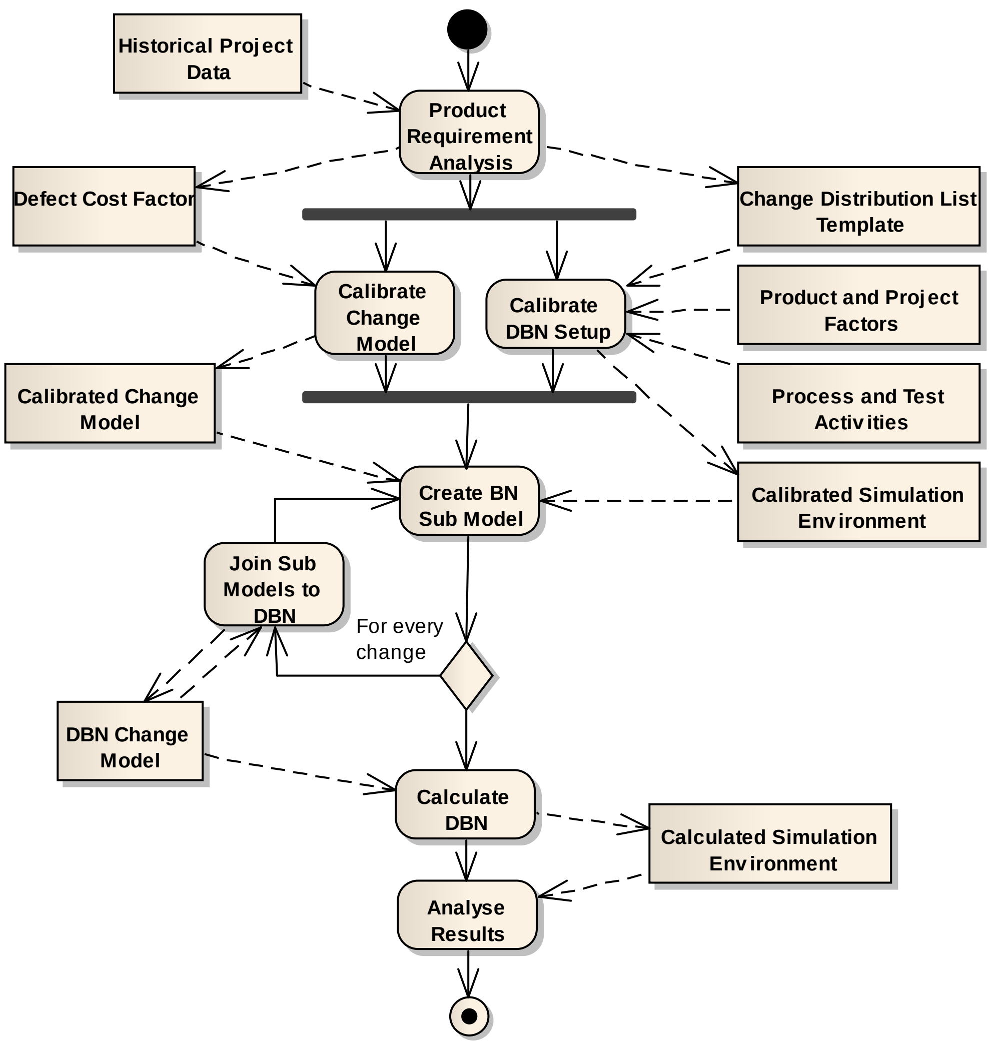


Figure 5.3: Setup SPM

The following step *calibrate change model* is used to setup the *defect cost factor* in the BN model itself to reflect the characteristics of the feature to be realized. In parallel, the step *calibrate DBN setup* adjusts the parameters within the simulation environment. The DBN setup is responsible for creating the final DBN in a later step. It uses the *change distribution list template*, *product and project factors* and *process and test activities* to individually setup every change model. The change distribution list template is selected based on the product requirement analysis. It defines the expected type of change distribution for the specific feature, e.g. focusing on small changes for a feature at lower complexity. Every change in the change distribution list is calibrated according to its KPIs enabling SPM to vary these factors over time. Product and project factors define the quality of existing engineering artifacts, e.g. source code or documentation quality on which the feature development is based. Furthermore, project specific factors are adjusted in this step. Process and test activities represent the performance of the engineering environment.

The *calibrated change model* and the *calibrated simulation environment* are used in the following steps to create the final *DBN change model*. The simulation environment holds information for every change about the feature to be developed.

The steps *create BN sub model* and *join sub models to DBN* iterate over the predefined changes. For every change an instance of the change model is created and linked to to the DBN change model. Finally, SPM is ready to run its calculations according to the evidence that has been entered to it. This is done in step *calculate DBN*. Calculation takes place for every BN sub model taking previously calculated models into account if required. Results are stored distinctly in the *calculated simulation environment* for every BN sub model, for every change and for every related KPI. This enables project managers to assess the estimation variables developing over time. This is followed by the *results analysis* which is the central topic of the following section.

5.4.2 Change Factors

Change factors describe relations among change specific KPIs. The complexity of a feature is influenced by feature specific factors, feature difficulty and feature volatility. It describes the complexity of requirements and its impact on the difficulty to realize the feature. Figure 5.4 illustrates this specific part of SPM with the goal to estimate DCE based on the main influencing change factors.

In SPM it is possible to distinguish between three different levels of feature volatility and feature difficulty, from low to high. These nodes are merged into the average node feature complexity with levels from low to high. Feature volatility and complexity are elicited from internal data. They are acquired from the requirement management system used. Feature complexity also influences the DCF due to different error rates for different features. Some features are much easier to realize resulting in a low DCF,

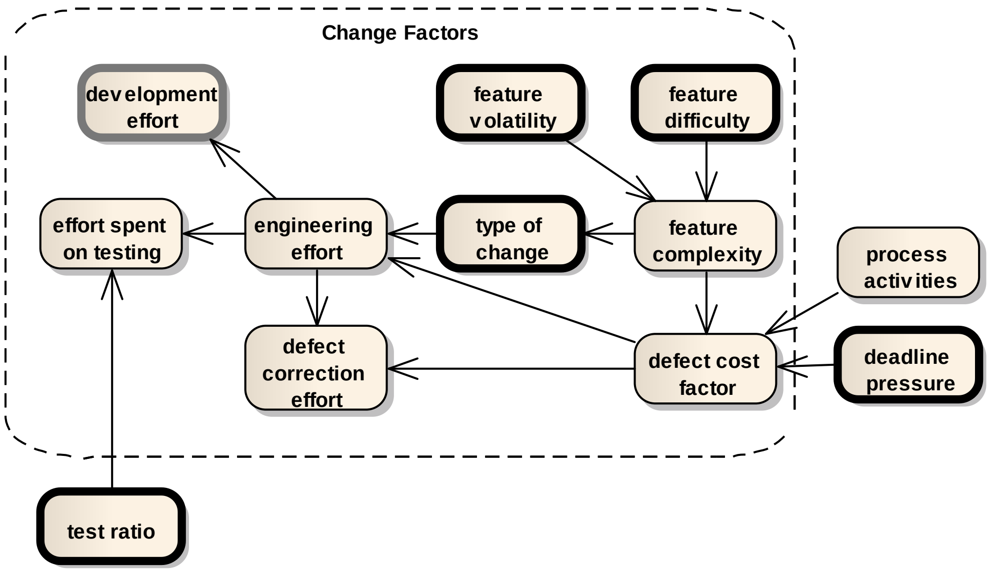


Figure 5.4: Change factors

whereas other features, e.g. with a high volatility and complexity, have much higher DCFs as described in chapter 4. Furthermore, it is influenced by process activities and project factors indicating the engineering environment quality. The higher the quality of these influencing factors the lower the DCF.

SPM is calibrated as described in the previous section 5.4.1, based on statistical data both from literature and project data and expert knowledge. Calibration values for feature volatility and complexity are based on expert knowledge. They are illustrated in table 5.1. The resulting *Defect Cost Factor (DCF)* is based on internal data.

Table 5.1: Features

	CAN	**HMI**	**PAD**	**VDB**	**CNV**
Volatility	Medium	High	Low	Low	Low
Complexity	Medium	High	Low	Low	Low
DCF	0.84	1.14	0.26	0.05	0.06

It can be seen that developing the HMI has the highest volatility and complexity leading to the highest DCF of 1.14. The HMI is followed by the CAN development at both medium volatility and complexity leading to a DCF of 0.84. The features PAD, VDB and CNV are at a low volatility and complexity level leading to a lower DCF of 0.26 for PAD, 0.05 for VDB and 0.06 for CNV.

Based on feature complexity the type of change is determined. It holds a distribution defining the amount of change categories used for a specific feature complexity, e.g. a feature with low complexity responsible for parameter maintenance within the product is developed mostly within smaller changes consuming only a low amount of effort because in the majority of cases only parameters are changed. Whereas features with higher complexity, e.g. a human machine interface development are realized mainly by larger changes consuming more effort. The average engineering effort needed for development is elicited based on the type of change. It includes all activities for a change including requirement engineering, design, implementation and testing. Engineering effort spent without testing is expressed in node development effort. It is dependent on the effort spent on testing as a result of the test activities. SPM uses the following data illustrated in table 5.2. It is based on the data set described in chapter 4. SPM extended the internal data set by mapping specific features, e.g. HMI to more general complexity classes. Thus, feature complexity high uses HMI data to elicit the corresponding type of change distribution.

Table 5.2: Type of change distribution

Type	**Feature Complexity**			**Engineering**
of Change	**Low**	**Medium**	**High**	**Effort**
Small	67.61%	34.55%	20.00%	3.84
Medium	7.04%	14.55%	10.00%	19.58
Large	2.82%	0.00%	2.50%	53.9
Other	12.68%	3.64%	5.00%	5.31
Bugfix	9.86%	47.27%	62.50%	7.15

The potential defect correction effort results from the average engineering effort needed for a specific type of change in combination with a feature's DCF. It estimates how much defect correction effort will be needed after change implementation. It is the basis for the potential amount of defects to be found by all test activities.

5.4.3 Process Activities and Project Factors

Process organization and process application represent the maturity level of an organization and its application. It is a measure of how capable process activities are to continuously ensure the development of high quality products. Figure 5.5 illustrates this specific part of SPM with the goal to describe the influence of those factors to the DCE.

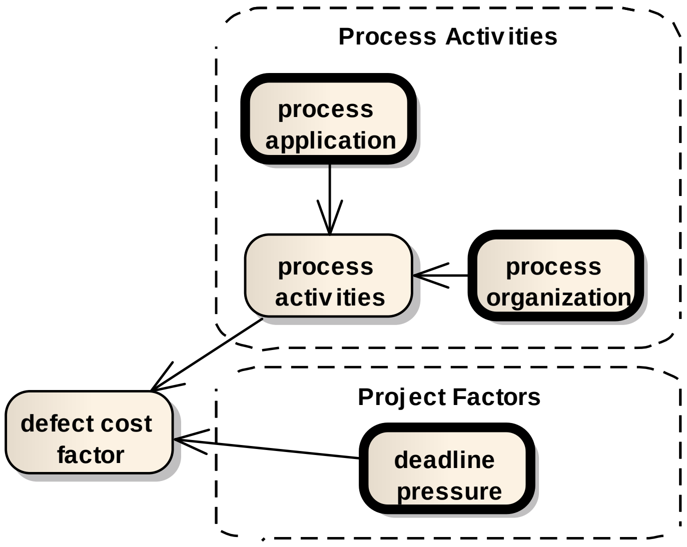


Figure 5.5: Process factors

Both process organization and process application are merged to process activities by their weighted minimum. The deadline pressure of a project influences the DCF of a feature as well as all process activities. Data leading to table 5.3 has been acquired based on the *Design of Experiment (DOE)* described in chapter 4. The experiment showed that at higher deadline pressure and low process activities it is possible to lower the average engineering effort to realize a specific change down to 50% of its original effort.

Table 5.3: Deadline pressure

Type of Effort	Process Activities & Project Factors		
	Low	Medium	High
DCE	100%	350%	500%
Engineering Effort	100%	50%	37.5%

The consequences lead up to a 350% higher DCE compared to ideal engineering conditions. Further pressure in combination with low process activities reduce the average engineering effort down to 37.5% with the effect to increase the potential DCE up to 500%.

5.4.4 Product factors

Product factors documentation quality and code quality represent a software product's testability. They have a major influence on a feature's error rate after the tests. The better these product factors are, the more effective testing is possible. Figure 5.6 illustrates this specific part of SPM with the goal to describe the influence of those factors to the DCE.

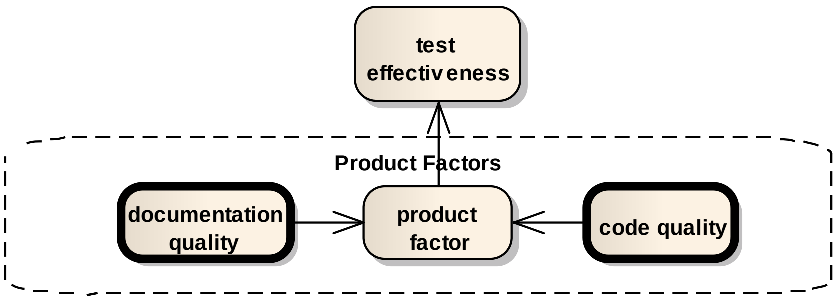


Figure 5.6: Product factors

Higher documentation as well as code quality represent a high product factor leading to higher defect detection rates. These nodes are defined by expert knowledge and merged in node product factors by their weighted minimum.

5.4.5 Test Activities

The better the test process is established, assuming the product is testable and there is enough effort spent on testing, the fewer defects reside in the final product. Figure 5.7 illustrates this specific part of SPM with the goal to describe the influence of those factors on the actual DCE resident after testing.

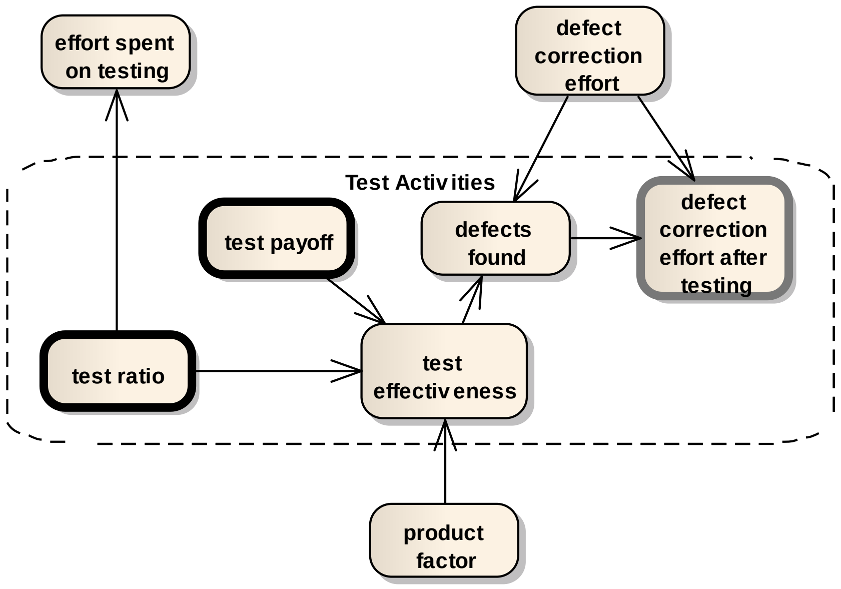


Figure 5.7: Test activities

Test payoff and test ratio influence the test effectiveness. Test payoff is the amount of defects that can be found by test activities as a proportion of the estimated defect correction effort. Test ratio is the relative amount of engineering effort spent on testing. The higher the test ratio the more effort is spent on testing as illustrated in table 5.4.

Table 5.4: Effort spent on testing

Test Ratio	**Test Effort / Engineering Effort**
Low	10%
Medium	20%
High	40%

Both *test payoff* and *test ratio* are merged by their weighted average in node *test effectiveness*. Table 5.5 illustrates different detection rates for all combinations of test ratio and test payoff levels. Detection rates are elicited based on external data [Jones, 2000] indicating how successful test activities can be. Experts supported the application of this data to SPM. *Defects found* and *defect correction effort after testing* are arithmetical nodes combining the estimated *defect correction effort* from change factors with test activities.

Table 5.5: Defect detection rates

Test Effectiveness		**Detection**
Test Payoff	**Test Ratio**	**Rate**
Low	Low	1%
Low	Medium	30%
Low	High	40%
Medium	Low	5%
Medium	Medium	37%
Medium	High	53%
High	Low	10%
High	Medium	75%
High	High	87%

5.5 Model Usage

The next step after building the SPM is the evaluation of its performance. SPM is evaluated from two perspectives:

1. Usage: How does the model work regarding the problem definition?
2. Accuracy: How accurate does the model predict estimation variables?

5.5.1 Validation Scenario

This scenario is used to verify how SPM performs regarding DCE estimation. All features (CAN, HMI, PAD, VDB and CNV) along with their volatility and complexity levels are part of the validation. The data set introduced in chapter 4 is split randomly into two equally sized parts, one is used for calibration, the other for validation. The results are shown in section 5.6.1.

5.5.2 Usage Scenarios

Finally, SPM is validated against its initial requirements whether it fulfils the solution for the problem definition derived from the first stage. To demonstrate SPM's practical capabilities two scenarios are compared, S1 and S2 as described below. In both scenarios, two features are developed, i.e. CAN and HMI. The simulation has 93 iterations, every iteration represents a change according to the feature's correspondent type of change distribution. The scenarios are defined as follows:

- **Scenario S1** represents a project under normal development conditions of a mature organization, medium deadline pressure, high process application, high defect detection potential with a high amount of testing.
- **Scenario S2** represents a project where deadlines are close, low process and test activities with focus to reducing the development effort to a minimum.

Table 5.6 illustrates the main differences between scenario S1 and S2.

Table 5.6: Project scenarios

Scenario	**Deadline Pressure**	**Process Application**	**Defect Detection Potential**	**Amount of Testing**
S1	Medium	High	High	High
S2	High	Low	Low	Low

The results can be seen in section 5.6.2.

5.6 Results

5.6.1 Validation Results

The validation results after simulation are presented in table 5.7 in form of median of resulting probability distribution. For the feature CAN, SPM predicts 246 hours of engineering effort (eng) and 229 hours of defect correction effort (corr). Prediction accuracy is calculated in relation to the reference data set as described in chapter 4. SPM achieves an accuracy of 95% for engineering effort and 97% for defect correction effort. Similar to feature HMI, PAD and VDB prediction accuracy is over 75%. For the feature CNV the prediction accuracy is lower, 63% for engineering effort and 14% for defect correction effort. The deviation in prediction accuracy for feature CNV compared to the other features could be a result of only a small amount of reference data available for model calibration.

Table 5.7: Prediction accuracy

Feature	Prediction		Reference		Accuracy	
	eng	**corr**	**eng**	**corr**	**eng**	**corr**
CAN	246	229	260	221	95%	97%
HMI	257	322	258	351	100%	92%
PAD	126	43	143	33	88%	77%
VDB	186	21	218	18	85%	84%
CNV	81	7	51	1	63%	14%

5.6.2 Scenario Results

Figure 5.8 illustrates the results for scenario S1 and S2. The scenarios are based on the scenario definition described in the previous section 5.5.2. The calculated values represent the median of the resulting probability distribution. SPM estimates 1316 hours of development effort for scenario S1 and 678 hours for scenario S2. The development effort needed to realize the features is reduced down to 50% where scenario S2 meets the target to reduce development effort to a minimum.

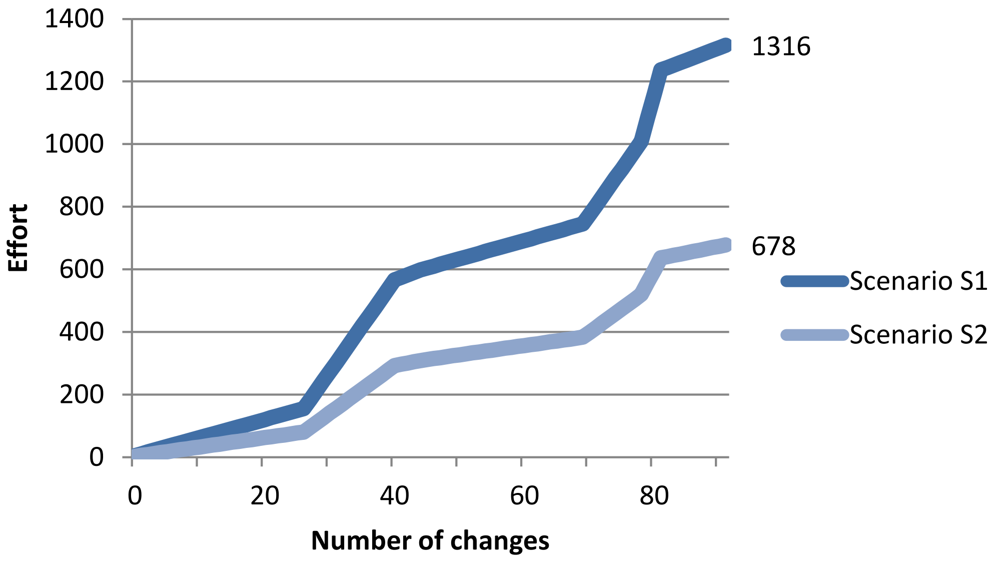


Figure 5.8: Development effort

Due to the missing quality and testing measures there is a higher amount of hidden defect correction effort in scenario S2. Figure 5.9 illustrates this. Scenario S1, with normal deadline pressure and established quality measures, supports the development of robust software leading to a very small amount of potential defect correction effort (41 hours). In contrast to that, scenario S2 has 1119 hours of potential defect correction effort still pending at the end of the project.

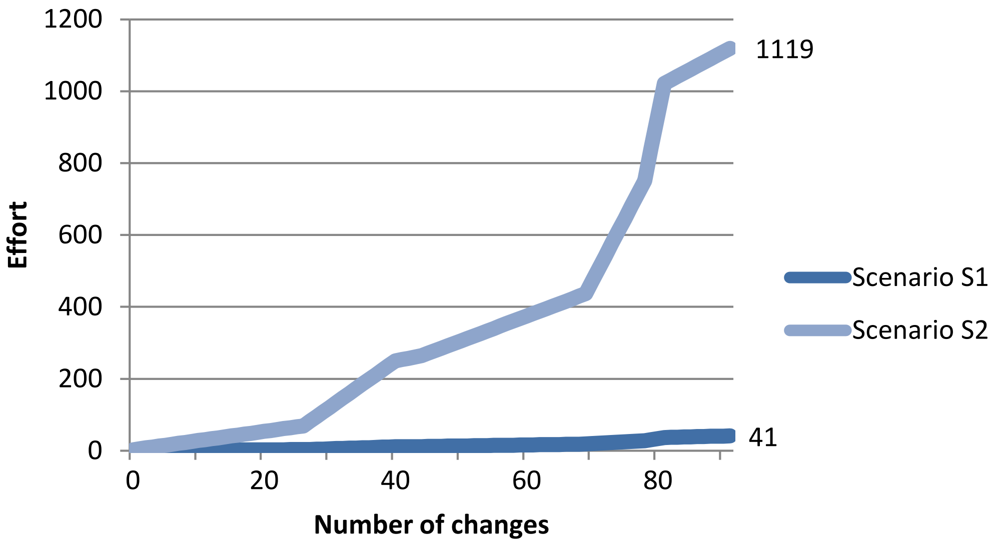


Figure 5.9: Defect correction effort

Table 5.8 depicts the overall effort needed to get a product at a similar quality level. In scenario S1 SPM estimates 1316 hours of development effort and 41 hours of potential defect correction effort. The resulting overall effort is 1357 hours. Scenario S2 has 678 hours of development effort and 1119 hours of defect correction effort resulting in 1797 hours overall effort. Thus scenario S2 working at high deadline pressure with supposed low overall effort overruns scenario S1 by 440 hours, 24% respectively.

Table 5.8: Simulation results

Scenario	Development Effort	DCE	Overall Effort
S1	1316 hours	41 hours	1357 hours
S2	678 hours	1119 hours	1797 hours

5.7 Model Complexity

DBNs can get very complex regarding their development and calculation time. SPM as a single instance represents a single change within the software development process. It consists of 20 nodes with 24 relations among each other. There are 10 calibration nodes representing the KPIs for every single change. Two output nodes connect one DBN to another. Calculation speed depends on processor speed and BN specific algorithmic features, e.g. does the BN engine use dynamic discretization or not. Calculation time for one instance takes about 1 minute on a 2GHz processor.

A single scenario in SPM consists of 100 changes similar to a single software releases. Therefore, several tools had to be developed to handle this complexity. Therefore, a tool was developed to handle the creation and linking of 100 DBNs. Furthermore, an other tool had to be developed to create the calibration data for every instance of the DBN 100 times in a row. Both tools had to be connected to the calculation engine where its results for every single node had to be exported to a database for later analysis.

Summing up, the creation of a single scenario in SPM from definition to analysis takes around one day and involves several pre- and post processing steps.

5.8 Conclusion

Even though there are many metric and related effort estimation programs which have been established for a long period of time [Ordonez and Haddad, 2008], the final breakthrough and commitment to metric based software quality assessment is still missing. These metric programs form the basis for further effort estimation methods such as SPM. SPM shows that based on accurate data it is possible to handle complex structures as well as incomplete and changing data. To achieve the goal of building a decision support system based on BNs it is crucial to have data representing the real world. It is essential to have a positive way of managing errors so that employees as well as project managers have a system of tracking defects that potentially have negative influence on their reputation. Furthermore employees need to belief that continuous improvement programs alleviate daily work. With the commitment to such programs the results of SPM show that it is possible to predict defect correction effort at high accuracy.

The SPM demonstrates the potential of using a system based on BNs to support decisions in the context of software effort estimation. Even though it is very challenging to build such complex models, it includes a wide range of benefits. BNs offer the possibility to support decision makers based on objective process and product data as well as on expert knowledge. Using BNs project managers could not only monitor the current situation of a project but also estimate and predict project behavior under specific circumstances. BNs enable decision makers to justify and document their decisions. Based on the graphical representation of cause and effect it is easy to discuss even complex problems and identify weaknesses or even new aspects in such complex domains as software effort estimation.

Explicit scenario analysis enables project managers to argue for their scenarios based not only on the knowledge of a single expert but incorporating knowledge from multiple experts possibly from different domains, e.g. project management, development or testing. Furthermore, the argumentation is based on empirical data from various sources of the implemented engineering process and hereby supports a decision in this specific context.

A further step to enhance the estimation of a software product's potential defect correction effort is to represent every engineering phase separately at a higher level of detail. SPM in its current version estimates the total defect correction effort over all phases of the engineering process. The design of a model explicitly incorporating the KPIs of every single engineering phase offers a wider range of optimization possibilities regarding effort estimation. This improvement enables not only to estimate the potential defect correction effort per engineering phase but also to consider the flow of defects from one phase to another. The resulting model is called the *Defect Cost Flow Model (DCFM)* where a defect's origin is assessed in relation to the place where it is detected. The DCFM takes the variation of effort respectively costs into account over multiple process phases. Because of that not only project managers but also process responsibles could benefit from the DCFM.

6 Defect Cost Flow Model

This chapter describes the creation of the *Defect Cost Flow Model (DCFM)* including its structure and how it performs. First, the *overview* section provides background information on the DCFM. Based on the *problem defintion*, the *Key Performance Indicators (KPIs)* and their relations can be defined. The following sections describe the DCFM in detail including the simulation of several scenarios demonstrating its capabilities. The simulation results are discussed after that. Finally, the conclusion section summarizes the simulation results and describes further steps in the domain of DCFM.

6.1 Overview

Process improvement activities identifying the optimal QA effort to detect most of the software defects tend to optimize locally in their specific domains. For example, a typical improvement activity is to focus on a defect reduction in development phases where most of the defects occur. However, the correction of defects especially is more costly if these defects are detected in a development phase later than the one they were detected in. Current models for software effort prediction do not incorporate the increase of defect correction costs over time.

As part of the initial problem analysis and definition, a different impact on the overall *Defect Correction Effort (DCE)* has been recognized for defects originating in other development phases than they are detected in compared to defects detected in the same development phases where they are made. Therefore, to reduce the DCE of the software product, every phase of the development process described in 2.1.5 has been assessed along with its specific DCE characteristics and focused on the shift of defects over development phases. The idea of a *Defect Flow Model (DFM)* has already been established (see 2.2) but focusing on the number of defects instead of DCE. The DCFM has been created to identify development phases in which an optimization of QA effort will lead to lowest costs.

The DCFM enables to estimate the DCE dependent on KPIs which represent product, process and project characteristics. 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, the DCFM follows the concept of focusing on a single high level function of the software product due to different error rates for different features.

The underlying technique of the DCFM is based on *Bayesian Networks (BNs)* to incorporate both process data and expert knowledge within a single model. Historical and current process data as well as expert knowledge have been used for model calibration and validation.

6.2 Problem Definition

The continuous improvement of processes demands a reduction of the development effort needed to realize a specific feature. At first sight, QA effort represents additional effort and seems to stand in contrast to an overall effort reduction. However, the DCE increases if defects stay undetected over development phases. The problem was to create a model to identify development process phases in which optimization leads to a reduction of the DCE and thereby of the overall effort, which is the basis of our statement of the problem:

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

Model data in form of DCFM's KPIs 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 good are the development phases with regard to unnecessary rework?

- How much effort is spent on QA measures?
- How effective are these QA measures?
- What is the additional effort needed for defects shifting from one development 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. For example, one scenario focuses on the elimination of all defects just before customer delivery whereas another scenario focuses on the maximum amount of QA effort spent in early phases.

6.3 Defect Cost Flow

The concept of flowing defects is illustrated in Figure 6.1. There are four development phases built into the DCFM according to the V-Model development process definition in section 2.1.5: *Requirements Engineering (RE)*, *Design (DE)*, *Implementation (IM)* and *Integration & Test (I&T)*. For every phase, the DCE is determined separately based on process specific KPIs.

In phase RE, there is only one possibility to have defective requirements, represented by the node *DCE after QA*. QA activities, e.g. requirement reviews, detect defects and hereby reduce the potential DCE. The remaining, undetected defects are handed over to the subsequent phase DE.

In phase DE, there are two defect types: one originating in phase RE, e.g. a defective requirement leading to problems creating the design artifacts. Second, there are defects introduced in the DE phase itself represented by the grey node *DCE after QA*. RE defects are adjusted by its *phase multiplier*. It represents the increase of effort if defects are not corrected in the same phase in which they are injected. This for example is the case, if the design process has to be followed twice due to a defective requirement. The phase multiplier varies from company to company and depends on the development process. The more development phases are involved, the higher the overall effort needed to fix these defects. There are different phase multipliers for different phases. Possible QA measures in this phase are design reviews.

In phase IM there are three different defect types. The first are defects with their origin in phase RE, e.g. a defective requirement passed through the design process on which an implementation is based. Second, there are DE defects, e.g. if a defective design is used as basis for an implementation. Defects with their origin in phase RE and DE are adjusted by their specific phase multipliers. And finally, there are defects originating from a defective implementation. In this phase typical QA activities are code reviews and several types of tests.

In phase I&T, there are defects from RE, DE, IM and I&T itself. This phase represents final integration and test activities for development artifacts of all phases. This is the point where where the software product is delivered to the customer. The resulting DCE after I&T represents rework for defects detected by the customer and thereby the additional effort needed to finish the project.

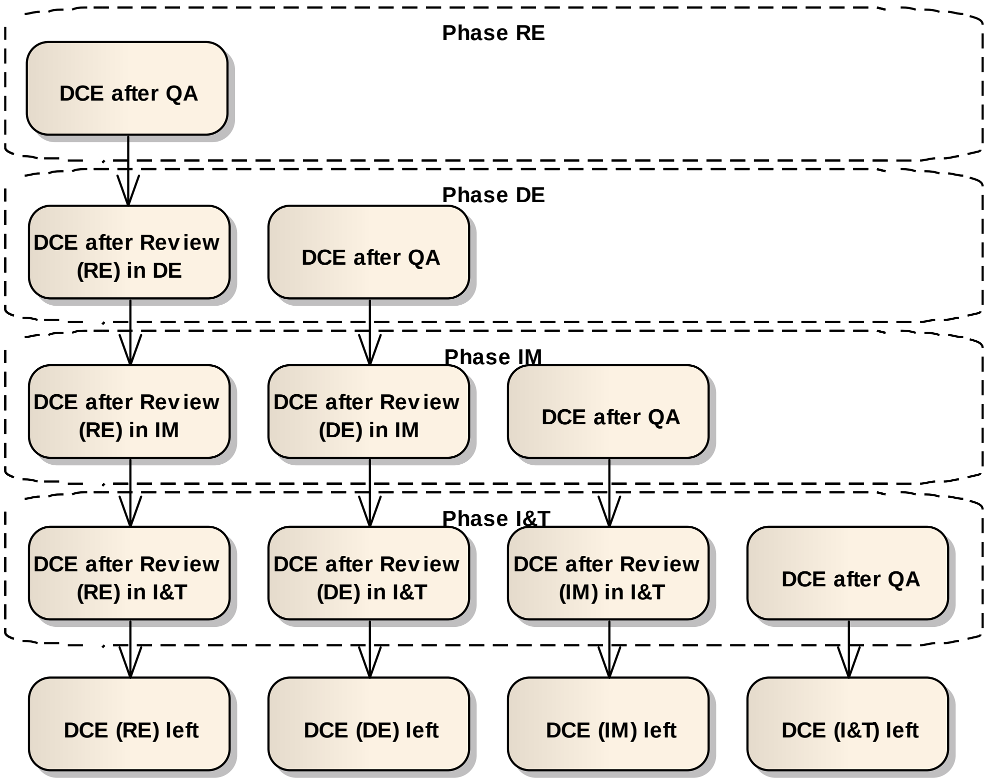

Phase RE
DCE after QA
Phase DE
DCE after Review (RE) in DE
DCE after QA
Phase IM
DCE after Review (RE) in IM
DCE after Review (DE) in IM
DCE after QA
Phase I&T
DCE after Review (RE) in I&T
DCE after Review (DE) in I&T
DCE after Review (IM) in I&T
DCE after QA
DCE (RE) left
DCE (DE) left
DCE (IM) left
DCE (I&T) left

Figure 6.1: DCFM concept

6.3.1 Example

The following example is based on the DFM introduced in chapter 2.2. It is extended based on the internal data set described in chapter 4 to reflect DCE instead of number of defects. The cost development in DCFM is shown in Figure 6.2 and Table 6.1 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. The positive axis illustrates the effort spent on defect correction whereas the negative axis depicts the reduced DCE after QA measures. Starting with defects originating in phase RE, there are 440 hours of potential DCE residing in the software product. With QA measures and a very high DCE reduction 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 remain undetected and shift from phase RE to DE. Here, 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. Here, the DCE is adjusted again by a phase multiplier of 5 leading to 198 hours of DCE injected originally in phase RE. In the IM phase it is more difficult to detect defects from RE, resulting in a DCE reduction rates of 43%. Finally, the DCE for RE defects shift to the I&T phase. Here it is adjusted by a phase multiplier of 4 resulting in 455 hours of DCE. With the help of QA measures, it is reduced by 387 hours to a DCE of 68 hours. This final DCE represents the residing defects potentially detected by the customer.

Table 6.1: Defect Cost Flow data

Phase	RE	DE	IM	I&T	CU
RE	440	264	198	455	68
DE		1120	840	1932	290
IM			1000	880	194
I&T				200	50
I&T				-150	
IM			-780	-686	
DE		-952	-357	-1642	
RE	-374	-224	-84	-387	

In every development phase there is a correspondent DCE, either reduced by specific QA measures or shifting to further phases. This example shows that even with very high DCE reduction rates, every undetected defect flowing from one phase to another causes a multiple of the correction effort when compared to the QA measures.

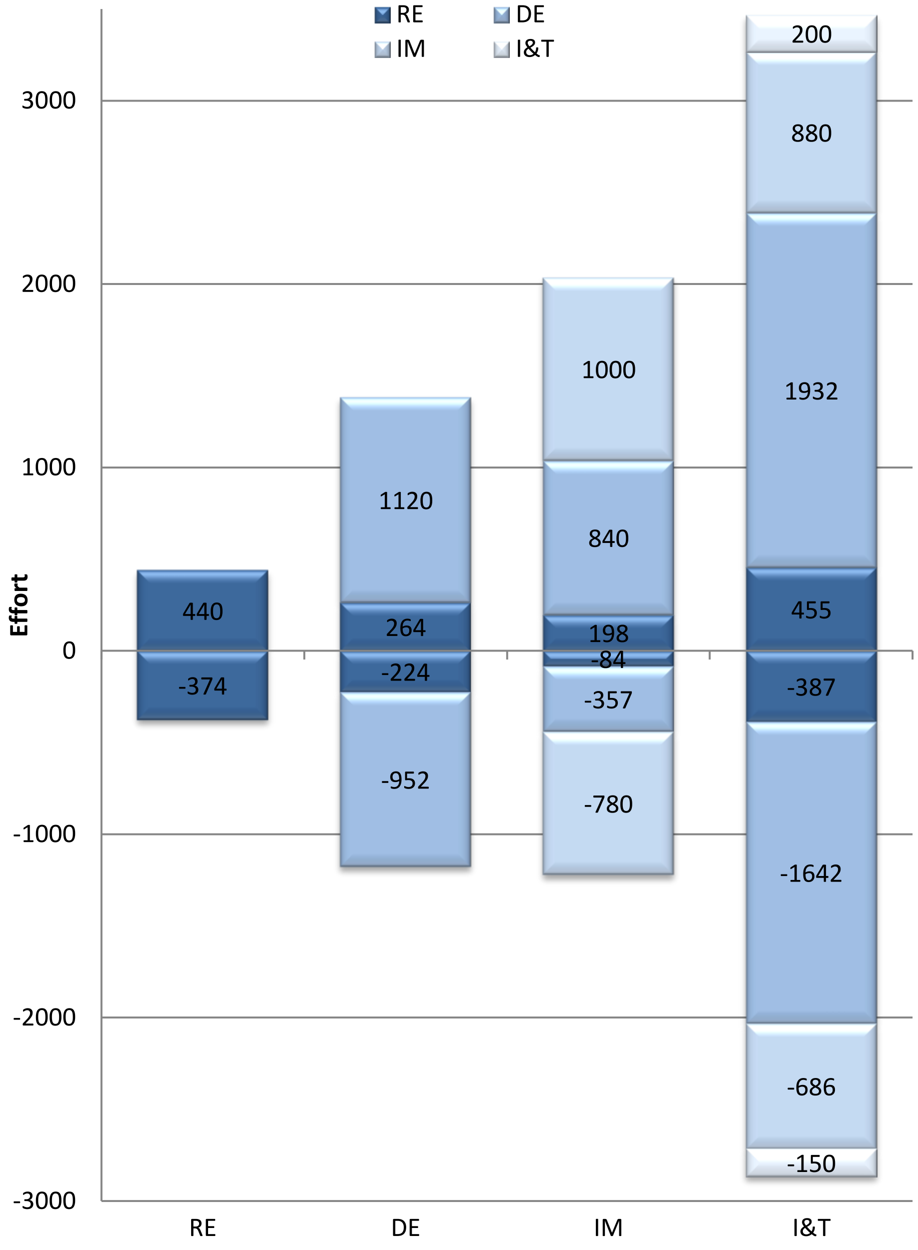


Figure 6.2: Defect Cost Flow

6.4 Identifying KPIs and Relationships

The aim of the model is to identify the ideal amount of effort spent on QA measures per development phase to reduce the overall effort. The identification of KPIs and their relations is the first step in creating such a model. According to the 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 the model. The following enumeration holds 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.

Q1 How is the defect correction effort determined?

- M1 *DCF*. Defect Cost Factor indicating a feature's error-proneness.
- M2 *DCE*. Potential defect correction effort (in hours) spent on correcting defects. Multiplication of defect correction factor and development effort.

Q2 How is the benefit / effort of QA determined?

- M1 *sufficiency of QA effort*. Degree of QA activity.
- M2 *QA effort*. Effort (in hours) spent on QA measures.
- M3 *DCE reduced*. Reduction of DCE based on the sufficiency of QA effort. It is an integrated part of the QA activities.
- M4 *DCE after QA*. Remaining DCE. Difference between defect correction effort and defect correction effort after QA.

Q3 How are additional costs taken into account for defects shifting over phases?

- M1 *phase multiplier*. Company specific factor representing additional rework if defects flow over development phases.

Q4 How is the overall effort determined?

- M1 *development effort*. Effort (in hours) spent on artifact development in RE, DE, IM and I&T excluding QA and correction effort.

M2 *overall effort*. Includes all types of effort. Sum of development effort, QA effort, defect correction effort reduced and defect correction effort after QA

6.5 Model Creation

Finally, all necessary information can be put into the DCFM BN. It is based on the problem definition to identify product areas where optimization has an optimal cost-benefit ratio regarding DCE reduction rates. Based on the problem definition, KPIs are identified including internal process data, expert knowledge and external data. The resulting indicators are put into the resulting model consisting of 60 nodes representing four development phases: RE, DE, IM and I&T. The overall model structure is depicted in Figure 6.3. As indicated by its legend, Figure 6.3 illustrates the BN as a cause and effect chain where nodes represent events and arcs their relation. Calibration nodes are used for setting up the model.

The following sections describe every phase in detail, about the relation of nodes as well as how they are calibrated. The basic principle repeated in every phase is explained based on the initial phase RE. Following phases are explained based on additional principles increasing with every phase, e.g. every phase holds information about defects from earlier phases.

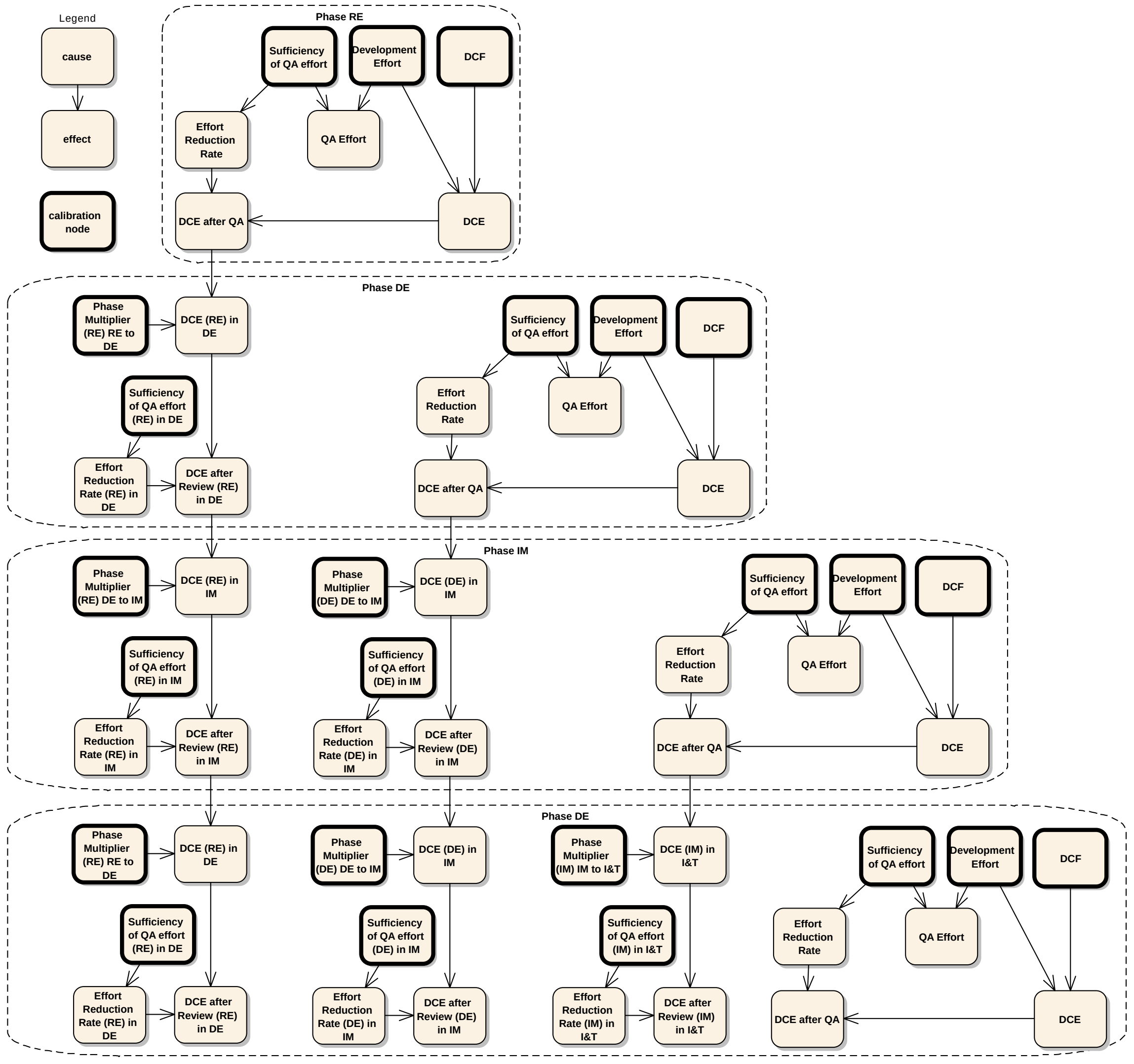

Legend
cause
effect
calibration node
Phase RE
Sufficiency of QA effort
Development Effort
DCF
Effort Reduction Rate
QA Effort
DCE after QA
DCE
Phase DE
Phase Multiplier (RE) RE to DE
DCE (RE) in DE
Sufficiency of QA effort (RE) in DE
Effort Reduction Rate (RE) in DE
DCE after Review (RE) in DE
Phase IM
Phase Multiplier (RE) DE to IM
DCE (RE) in IM
Sufficiency of QA effort (RE) in IM
Effort Reduction Rate (RE) in IM
DCE after Review (RE) in IM
Phase Multiplier (DE) DE to IM
DCE (DE) in IM
Sufficiency of QA effort (DE) in IM
Effort Reduction Rate (DE) in IM
DCE after Review (DE) in IM
Phase DE
Phase Multiplier (IM) IM to I&T
DCE (IM) in I&T
Sufficiency of QA effort (IM) in I&T
Effort Reduction Rate (IM) in I&T
DCE after Review (IM) in I&T

Figure 6.3: Bayesian Network DCFM

6.5.1 Requirements Engineering

Starting in phase RE there are three calibration nodes as illustrated in Figure 6.4: *Sufficiency of QA effort*, *Development Effort* and *DCF*. These nodes are calibrated based on data as well as on expert knowledge representing the varying factors. Based on these factors, every phase is adjusted according to its specific boundary conditions. These calibration nodes are combined in child nodes. The resulting calculation is illustrated by an output node which again could be an input node to an other network. All nodes are described in detail throughout the following section 6.6.

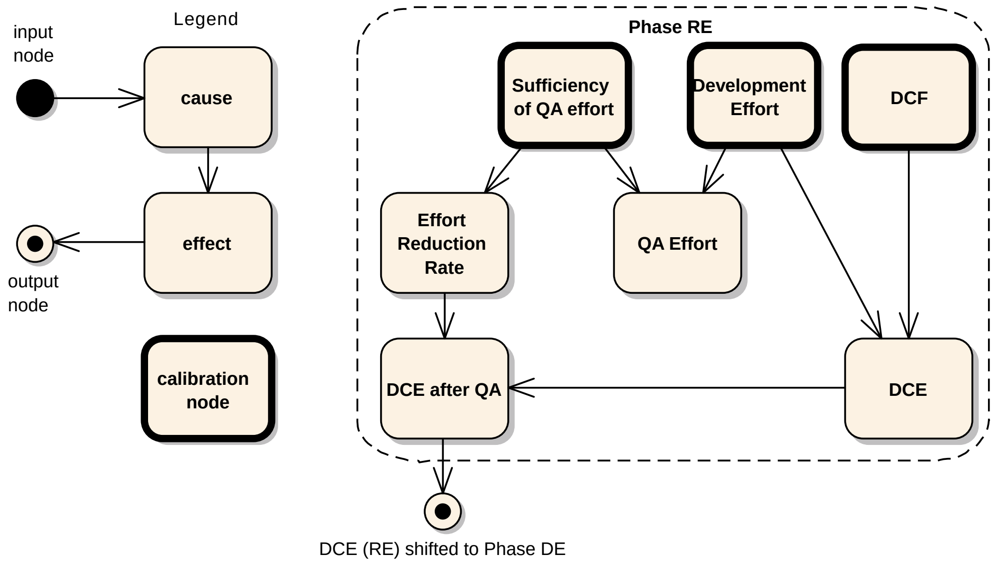


Figure 6.4: DCFM part RE

Parent (calibration) as well as child (following) nodes are described throughout the following sections. Calibration nodes only represent a single dimension *Node Probability Table (NPT)*, e.g. for a specific sufficiency of QA effort there is a corresponding probability of occurrence between 0% and 100%. However, for child nodes with more than one parent node, NPT become multidimensional to represent all possible combinations from all incoming nodes. This results to, for example, specific DCE reduction rates for every sufficiency of QA effort. The final processing step *DCE after QA* estimates the DCE for this specific phase. This node is the output node for this phase and the input node for the following, indicated by the node *DCE (RE) shifted to phase DE*.

6.5.2 Design

Estimating the DCE for phase DE is similar to as it is in the earlier phase RE. This is illustrated by the right side of Figure 6.5. Here, the DE phase specific DCE after QA is calculated and shifted to the following phase IM through output node *DCE (DE) shifted to phase IM*.

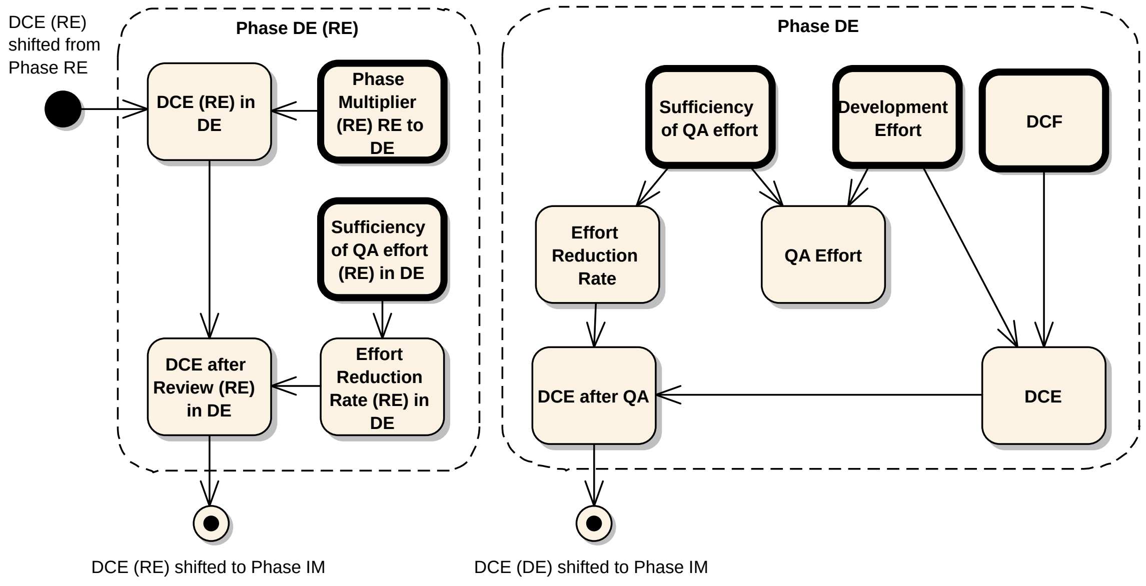

DCE (RE) shifted from Phase RE
Phase DE (RE)
DCE (RE) in DE
Phase Multiplier (RE) RE to DE
Sufficiency of QA effort (RE) in DE
DCE after Review (RE) in DE
Effort Reduction Rate (RE) in DE
Phase DE
Sufficiency of QA effort
Development Effort
DCF
Effort Reduction Rate
QA Effort
DCE after QA
DCE
DCE (RE) shifted to Phase IM
DCE (DE) shifted to Phase IM

Figure 6.5: DCFM part DE

According to the concept of the DCFM, every phase has information about defects from earlier phases. In Figure 6.5, this is expressed as *phase DE (RE)* indicating the shift of defects from the earlier phase RE to the current phase DE. Input node *DCE (RE) shifted from phase RE* holds the DCE. It is multiplied by the *Phase Multiplier (RE) RE to DE* representing additional effort that has to be spent for RE defects shifting from phase RE to DE. Details on phase multipliers are given in section 6.6. In phase DE, nodes *sufficiency of QA effort (RE) in DE* and *effort reduction rate (RE) in DE* are different than in phase RE corresponding to phase specifics. For example, the DCE reduction rate in phase DE is lower for defects introduced in RE than for defects introduced in phase DE. This is because defective or even missing requirements lead to a defective design even if the design itself is correct. The different defect detection potentials for every phase are explained later in section 6.6. The pattern of shifting defects is repeated for every succeeding phase.

6.5.3 Implementation

In phase IM, there are three different types of defects already. Based on their origin they can be separated into defects with origin in phase RE, DE and IM itself. Figure 6.6 illustrates those three categories. *Phase IM (RE)* represents defects in phase IM with origin in phase RE. This is illustrated by input node *DCE (RE) shifted from phase DE*. After calculating the *DCE after Review (RE) in IM* it shifts to the final phase I&T. From model perspective, calculating *Phase IM (DE)* is similar. Only calibration data varies from phase to phase.

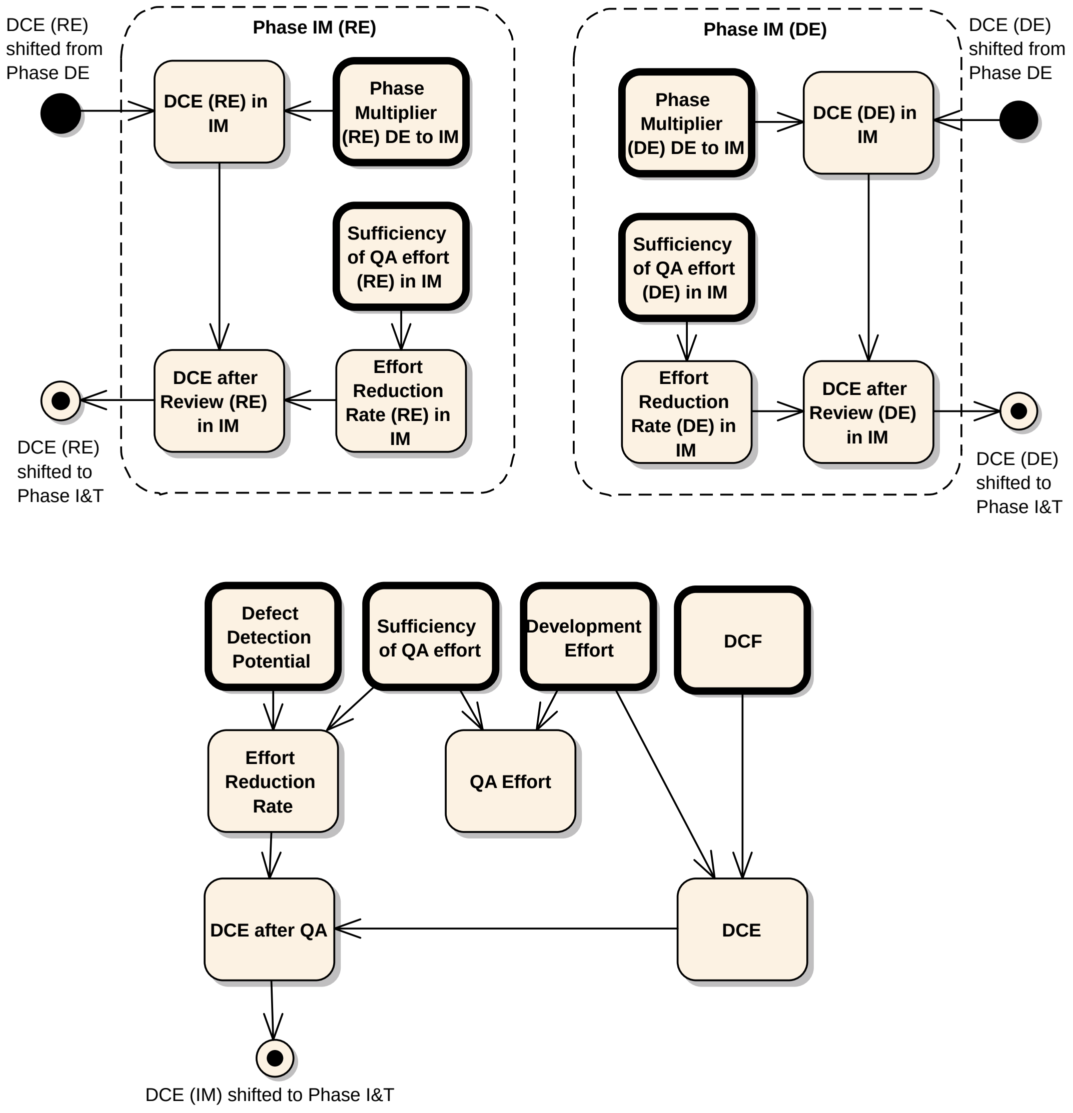

DCE (RE) shifted from Phase DE
Phase IM (RE)
DCE (RE) in IM
Phase Multiplier (RE) DE to IM
Sufficiency of QA effort (RE) in IM
DCE after Review (RE) in IM
Effort Reduction Rate (RE) in IM
DCE (RE) shifted to Phase I&T
Phase IM (DE)
DCE (DE) shifted from Phase DE
Phase Multiplier (DE) DE to IM
DCE (DE) in IM
Sufficiency of QA effort (DE) in IM
Effort Reduction Rate (DE) in IM
DCE after Review (DE) in IM
DCE (DE) shifted to Phase I&T
Defect Detection Potential
Sufficiency of QA effort
Development Effort
DCF
Effort Reduction Rate
QA Effort
DCE after QA
DCE
DCE (IM) shifted to Phase I&T

Figure 6.6: DCFM part IM

6.5.4 Integration & Test

Finally in phase I&T there are four different types of defects, one for every phase. This is illustrated in Figure 6.7. The calculation of a phase specific DCE is handled in *phase I&T (RE)*, *phase I&T (DE)* and *phase I&T (IM)*. Output nodes *DCE (RE) left*, *DCE (DE) left*, *DCE (IM) left* and *DCE (I&T) left* illustrate residing DCE before software release.

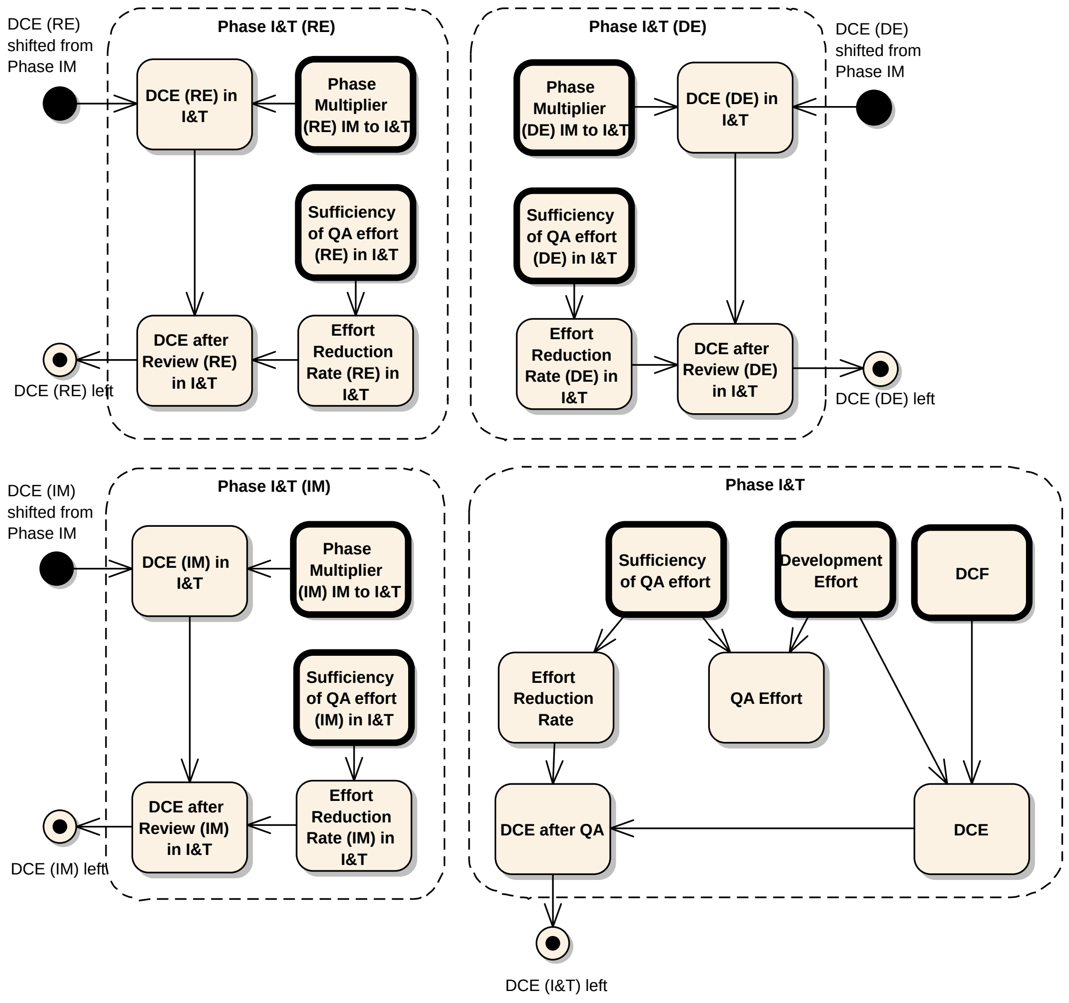

DCE (RE) shifted from Phase IM
Phase I&T (RE)
DCE (RE) in I&T
Phase Multiplier (RE) IM to I&T
Sufficiency of QA effort (RE) in I&T
DCE after Review (RE) in I&T
Effort Reduction Rate (RE) in I&T
DCE (RE) left
Phase I&T (DE)
DCE (DE) shifted from Phase IM
Phase Multiplier (DE) IM to I&T
DCE (DE) in I&T
Sufficiency of QA effort (DE) in I&T
Effort Reduction Rate (DE) in I&T
DCE after Review (DE) in I&T
DCE (DE) left
DCE (IM) shifted from Phase IM
Phase I&T (IM)
DCE (IM) in I&T
Phase Multiplier (IM) IM to I&T
Sufficiency of QA effort (IM) in I&T
DCE after Review (IM) in I&T
Effort Reduction Rate (IM) in I&T
DCE (IM) left
Phase I&T
Sufficiency of QA effort
Development Effort
DCF
Effort Reduction Rate
QA Effort
DCE after QA
DCE
DCE (I&T) left

Figure 6.7: DCFM part I&T

6.6 Model Calibration

The following sections describe how calibration nodes are set up and how they are related among each other.

6.6.1 Defect Cost Factors

In every phase, there is a specific development effort and *Defect Cost Factor (DCF)* 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 sufficiency 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, e.g. I&T, have lower DCFs than early phases, e.g. RE.

The DCFM is calibrated using phase specific DCFs. They are illustrated in Table 6.2. This calibration of DCFs is based on the assumption to develop a complex feature, e.g. the HMI. Other features of lower complexity have lower DCFs. These DCFs are taken based on the data described in chapter 4.

Table 6.2: DCFs per development phase

RE	**DE**	**IM**	**I&T**
1	0.8	0.7	0.01

6.6.2 Quality Assurance Activities

In every development phase, both nodes *development effort* and *defect cost factor* are combined to *defect correction effort* in a multiplication node. The nodes *effort reduction rate* and *Sufficiency of QA effort* represent the effectiveness of all QA activities for specific development phases. The *effort reduction rate* enables to define DCE reduction rates resulting from 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:

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

Table 6.3 illustrates the review effort $E_R(E_D)$ for every sufficiency of QA effort 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 6.3: Sufficiency of QA effort

Rank	$\mathbf{E_R(E_D)}$
Low	5%
Medium	10%
High	20%
Very High	40%

6.6.3 Effort Reduction Rates

The nodes *effort reduction rate* and *sufficiency of QA effort* result in *DCE after QA*. This node defines the possible reduction of DCE for every phase. The corresponding defect DCE reduction rates are shown in Table 6.4.

In every development phase there is a specific DCE reduction rates dependent on the sufficiency of QA effort and defect type, e.g. Table 6.4 column $D(DE)$ represents the DCE reduction rate in the DE phase for defects created in the DE phase itself and defects created in the previous phase RE.

Table 6.4: Effort Reduction Rates

Sufficiency of	$\mathbf{D(RE)}$	$\mathbf{D(DE)}$		$\mathbf{D(IM)}$			$\mathbf{D(I\&T)}$			
QA effort		**RE**	**DE**	**RE**	**DE**	**IM**	**RE**	**DE**	**IM**	**I&T**
Low	10%	10%	5%	5%	5%	8%	5%	5%	8%	8%
Medium	60%	60%	30%	30%	30%	53%	30%	30%	53%	53%
High	75%	75%	38%	38%	38%	68%	38%	38%	68%	68%
Very High	85%	85%	43%	43%	43%	78%	43%	43%	78%	78%

6.6.4 Development Phase Multipliers

Figure 6.5 illustrated the residing effort shifting from phase RE to DE. At this point it is adjusted by the *phase multiplier* to node *DCE (RE) in DE*. The phase multiplier represents additional rework that has to be done over two phases, e.g. if a defective requirement is detected in phase DE, first it has to be corrected (this effort cannot be saved) and, in addition to that, the design might have to be reworked. The phase multipliers of the DCFM are illustrated in Table 6.5. They are defined based on expert knowledge.

Table 6.5: Development phase multipliers

$\mathbf{RE \rightarrow DE}$	$\mathbf{DE \rightarrow IM}$	$\mathbf{IM \rightarrow I\&T}$
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 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.

6.7 Scenario Definition

Following the goal to identify the ideal distribution of QA effort through all development phases (RE, DE, IM and I&T) of the development process under discussion, four scenarios have been defined based on the assumption of developing 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 through all development phases.
- S2 uses a high amount of QA effort typically used if you consider optimizing a 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 in its own context only.
- S4 uses a very high amount of QA activities on all development phases.

Table 6.6 gives an overview on the amount of QA effort used for scenarios S1 to S4.

Table 6.6: Scenario overview

Phase	**Sufficiency of QA effort in**			
	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

6.8 Simulation Results

The simulation results are presented from two perspectives.

1. The development phase perspective focuses on how the scenario simulation performs in every development phase RE, DE, IM and I&T. Result illustration and data have been divided into two charts due to the high level of detail, especially in later phases. The development of effort for defects with their origin in phase RE or DE is explained in detail. Defects with their origin in phases IM and I&T are presented in form of an illustration along with its result data table.
2. The scenario perspective uses the DCFM to present the final simulation results for every scenario S1, S2, S3 and S4. The DCFM charts are explained along with their corresponding data tables.

All values represent the calculated median of the predicted probability distribution for DCE.

6.8.1 Development Phase Results

RE Results

For the RE phase, Table 6.7 and Figure 6.8 illustrate a constant DCE of 200 hours in all scenarios. In S1, the DCE could only be reduced by 23 hours as indicated in row RE. 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. The remaining DCE is shifted from the RE to DE phase and adjusted by the phase specific multiplier.

Table 6.7: RE scenario result data

Phase	S1	S2	S3	S4
RE	200	200	200	200
-RE	-23	-150	-170	-170

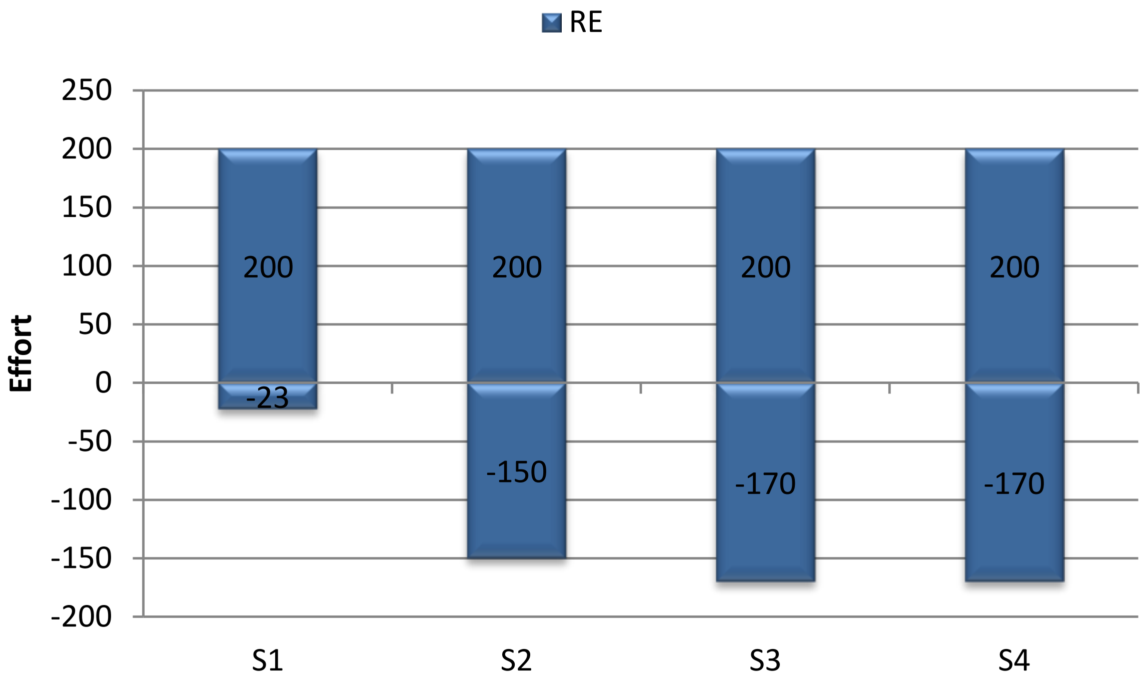


Figure 6.8: RE scenario results

DE Results

The development of DCE is illustrated in Table 6.8 and Figure 6.9. All scenarios have an additional DCE of 160 hours caused by defects originating in the DE phase. 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 the DE phase could be reduced by 16 hours in S1, 120 hours in S2 and 136 hours in S3 and S4. 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.

Table 6.8: DE scenario result data

Phase	S1	S2	S3	S4
DE	160	160	160	160
RE	598	199	119	119
-RE	-17	-76	-52	-52
-DE	-16	-120	-136	-136

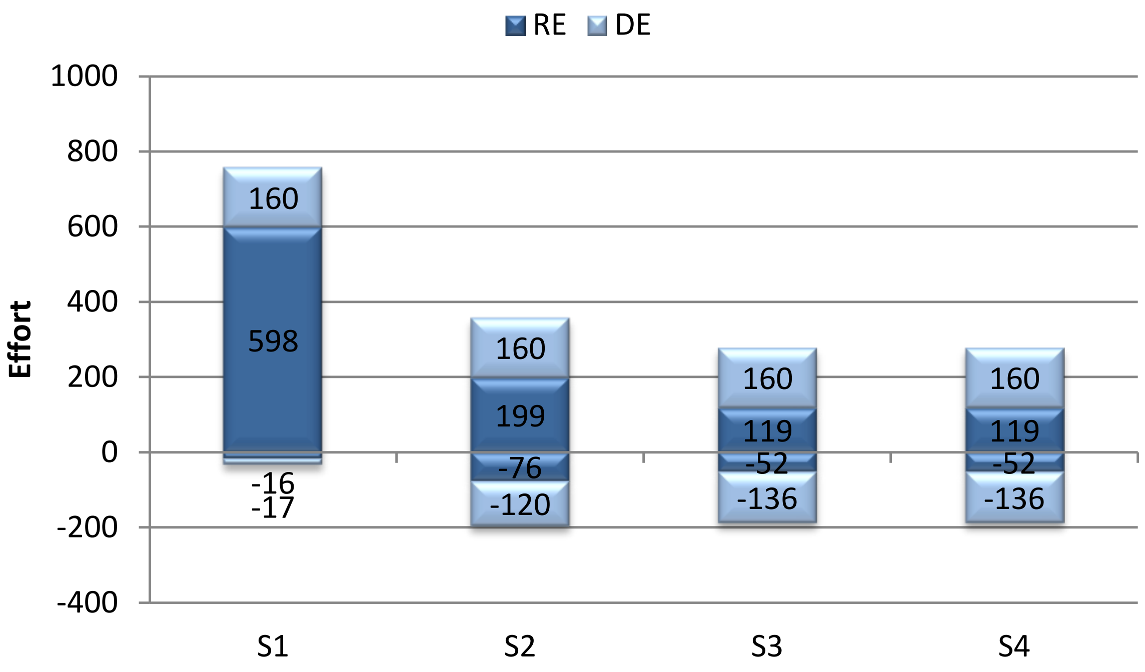


Figure 6.9: DE scenario results

IM Results

Table 6.9 and Figure 6.10 summarize IM results. The development of DCE for RE defects has increased to 2529 hours in S1, in S2 the DCE is still 601 hours whereas S3 and S4 only have 332 hours. For defects with their origin in phase DE, the DCE is 648 hours in scenario S1, 198 hours in S2 and 119 hours in S3 and S4. The DCFM assumes low effort reduction rates for RE and DE defects in phase IM. Thus, the DCE for defects with origin in phase RE is reduced by 36 hours in S1, 225 hours in S2 and 146 hours in S3 and S4. For DE defects, the reduction of DCE is 38 hours in S1, 74 hours in S2 and 53 hours in S3 and S4.

Table 6.9: 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

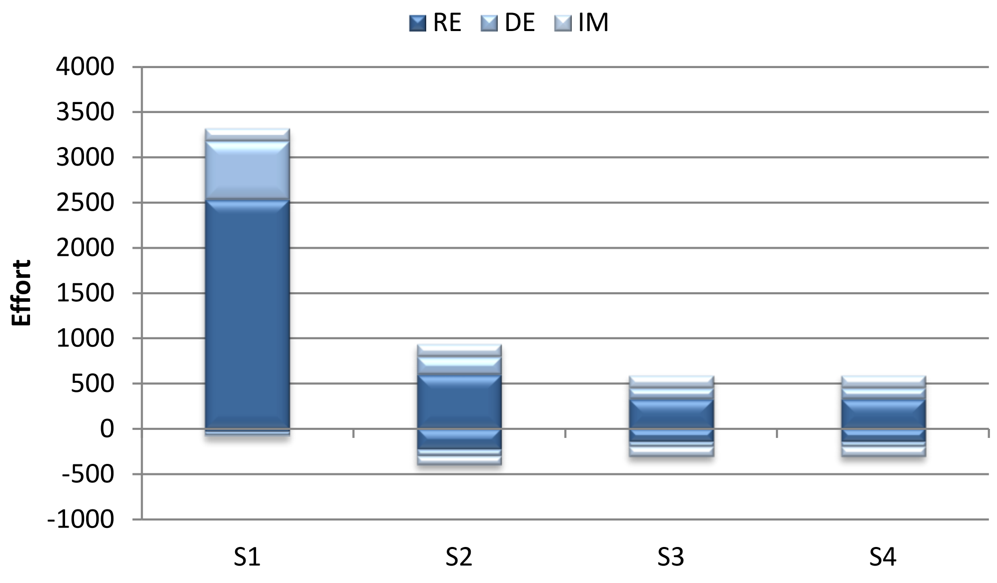


Figure 6.10: IM scenario results

I&T Results

The results for phase I&T are illustrated in Table 6.10 and Figure 6.11. The DCE for defects with their origin in phase RE increases to 8361 hours in scenario S1. In S2 the DCE is 1462 hours whereas in scenario S3 and S4 it is 722 hours. For DE defects, the DCE is 2048 hours in scenario S1, 482 hours in S2 and 265 hours in S3 and S4. The reduction of DCE for RE defects as a result of QA activities is 851 hours in scenario S1, 1115 hours in S2, 375 hours in S3 and 614 hours in S4. For DE defects the DCE could be reduced by 189 hours in scenario S1, 361 hours in S2, 198 hours in S3 and 225 hours in scenario S4.

Table 6.10: 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

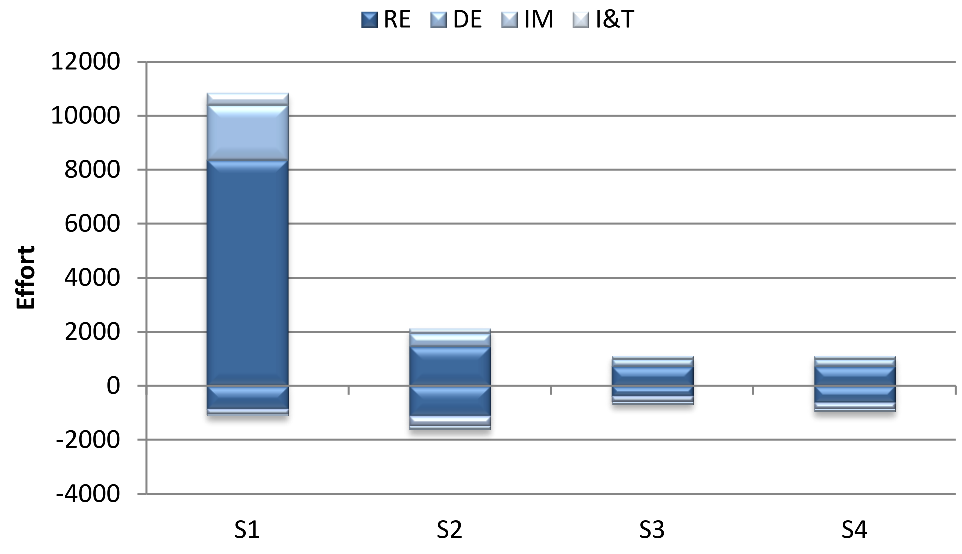


Figure 6.11: I&T scenario results

6.8.2 Scenario Results

S1 Results

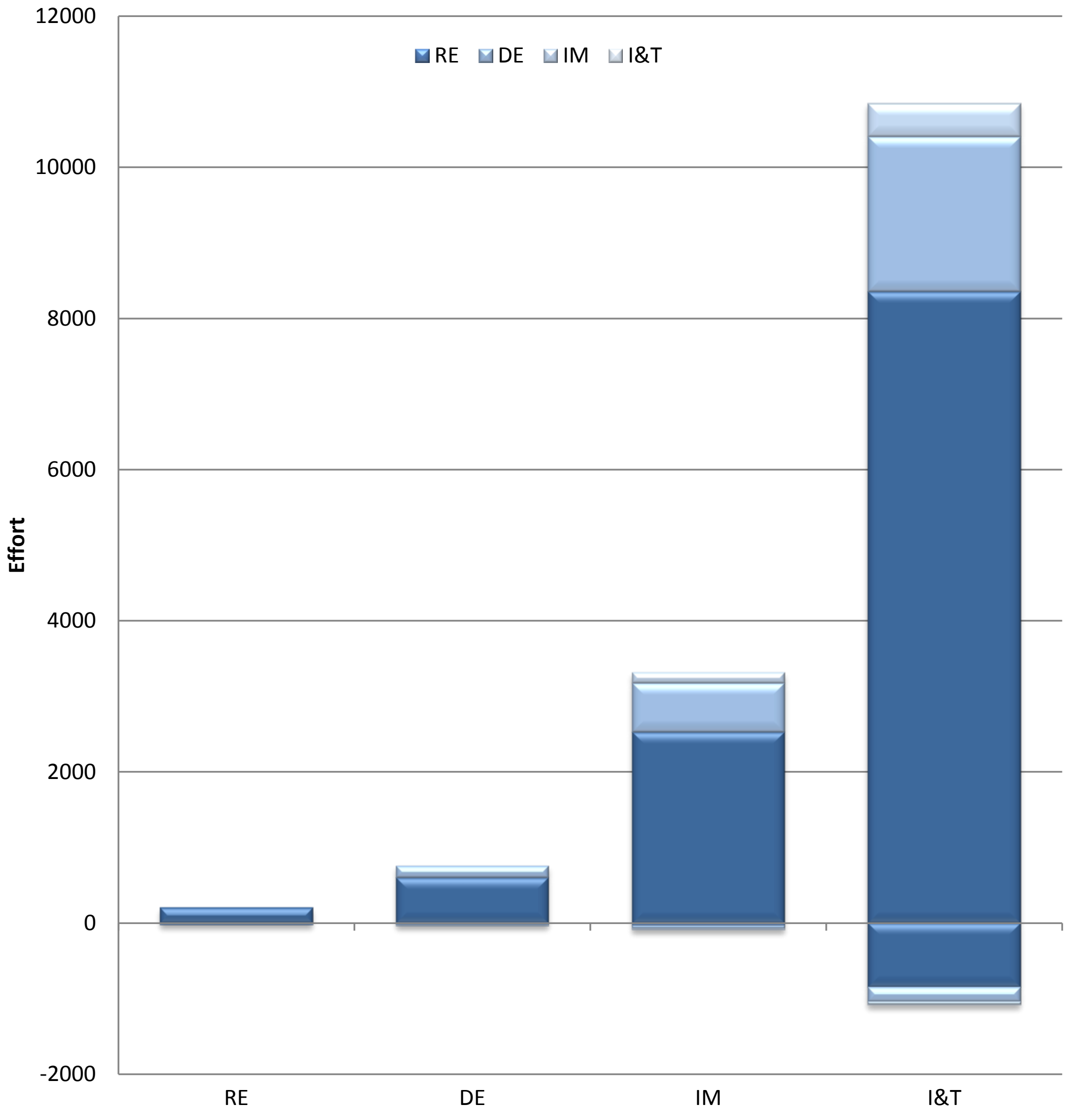


Figure 6.12: S1 DCFM chart

Table 6.11: S1 result overview

Development Effort	QA Effort	QA DCE	Residual DCE	Overall
1000 h	50 h	1224 h	9771 h	12045 h

Figure 6.12 and Table 6.11 on page 96 give an overview of the overall results for scenario S1 where QA activities are reduced to a minimum. Based on a development effort of 1000 hours, 50 hours are spent for QA activities. Low QA activities lead to low DCE reduction rates and therefore to a low DCE as part of QA. In case of S1, 1224 hours. This results in a residual DCE of 9771 hours in the product, which potentially can be detected by the customer. The overall effort needed to finish the desired feature is expected to be 12045 hours, 12 times the amount of development effort than expected. Table 6.12 shows the results in detail. In phase RE, there are 200 hours of DCE inserted while developing the feature. With QA activities, the DCE could only be reduced by 23 hours. Therefore, the defects with their origin in phase RE shifting to phase DE already have a DCE of 598 hours. In addition, 180 hours of DCE for defects with origin in phase DE are inserted. QA activities only reduce the DCE by 16 hours for DE defects and 17 hours for defects with origin in phase RE. At this point, the DCE for defects introduced in early phases increases exponentially in phase IM. Illustrated by column IM in Figure 6.12, it is 2628 hours for RE defects, 648 hours for DE defects and an additional 140 hours of DCE for IM defects. With low QA activities, the reduction of DCE is only 12 hours for IM defects, 38 hours for DE defects and 36 hours for RE defects. The exponential increase continues in phase I&T where the DCE for defects with origin in phase RE is 8361 hours, for DE defects it is 2048 hours and 434 hours for defects introduced in phase IM and 10 hours for I&T defects. Even with low QA activities, in this phase the DCE could be reduced by 1 hour for I&T defects, 41 hours for IM defects, 189 hours for DE defects and 851 hours for defects with origin in the RE phase.

Table 6.12: S1 result data

Phase	RE	DE	IM	I&T	CU
RE	200	598	2529	8361	7510
DE		160	648	2048	1859
IM			140	434	393
I&T				10	9
I&T				-1	
IM			-12	-41	
DE		-16	-38	-189	
RE	-23	-17	-36	-851	

S2 Results

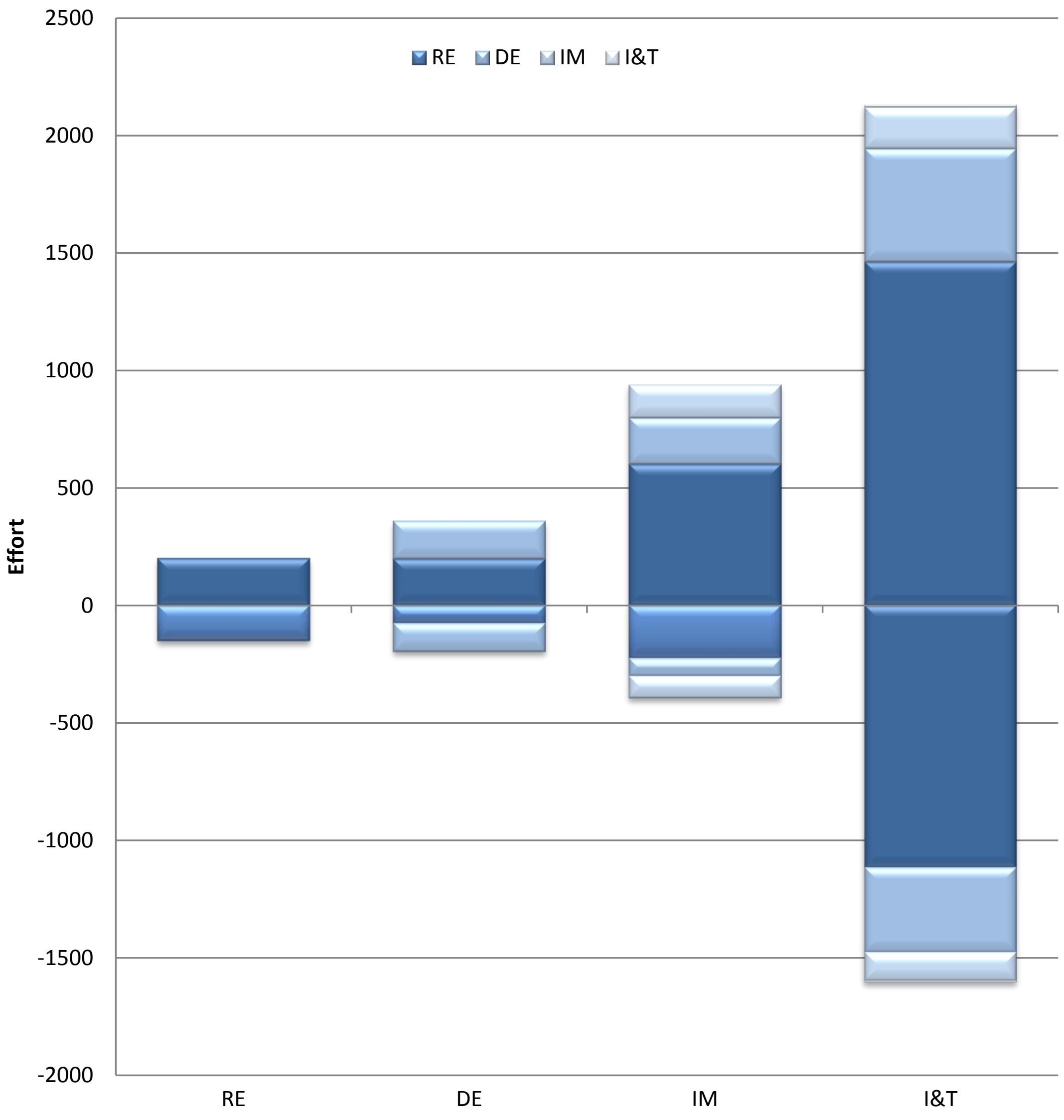


Figure 6.13: S2 DCFM chart

Table 6.13: S2 result overview

Development Effort	QA Effort	QA DCE	Residual DCE	Overall
1000 h	200 h	2343 h	528 h	4071 h

In scenario S2, all development phases are optimized in their specific domains to have a high cost benefit ratio among the optimal QA effort spent and the reduction of DCE per phase. Figure 6.13 and Table 6.13 on page 98 illustrate the results. Based on a development effort of 2000 hours, a total of 200 hours are spent for QA activities. The DCE as part of the QA activities is 2343 hours. The residual DCE after I&T is 528 hours leading to an overall effort of 4071 hours.

Table 6.14 depicts the details of this scenario. Starting with 200 hours of DCE for defects introduced in phase RE, the QA activities are able to reduce it by 150 hours. In phase DE, the RE defects are already at a DCE of 200 hours in addition to the 160 hours of DCE for defects with origin in phase DE. By QA activities these defects could be reduced by 120 hours for DE defects and 76 hours for RE defects. In phase IM, RE defects are at 601 hours of DCE, DE defects at 198 hours and defects with origin in the same phase IM at 140 hours of DCE. Even with high QA activities, it is difficult to detect defects from the earlier phases RE and DE in phase IM, leading to a reduction of only 225 hours for RE defects and 74 hours for DE defects. DCE reduction rates for IM defects are higher leading to a reduction by 95 hours for IM defects. In the final phase I&T RE defects are at 1482 hours, DE defects at 482 hours, whereas IM defects and I&T defects are only 177 hours and 10 hours respectively. In this final phase it is easier again to detect defects from early phases RE and DE because integration and testing is performed based on requirements and design. This leads to a reduction of 1115 hours of DCE for RE defects, 361 hours for DE defects, 121 hours for IM defects, and 6 hours for defects introduced in the final phase I&T.

Table 6.14: S2 result data

Phase	RE	DE	IM	I&T	CU
RE	200	199	601	1462	347
DE		160	198	482	121
IM			140	177	56
I&T				10	4
I&T				-6	
IM			-95	-121	
DE		-120	-74	-361	
RE	-150	-76	-225	-1115	

S3 Results

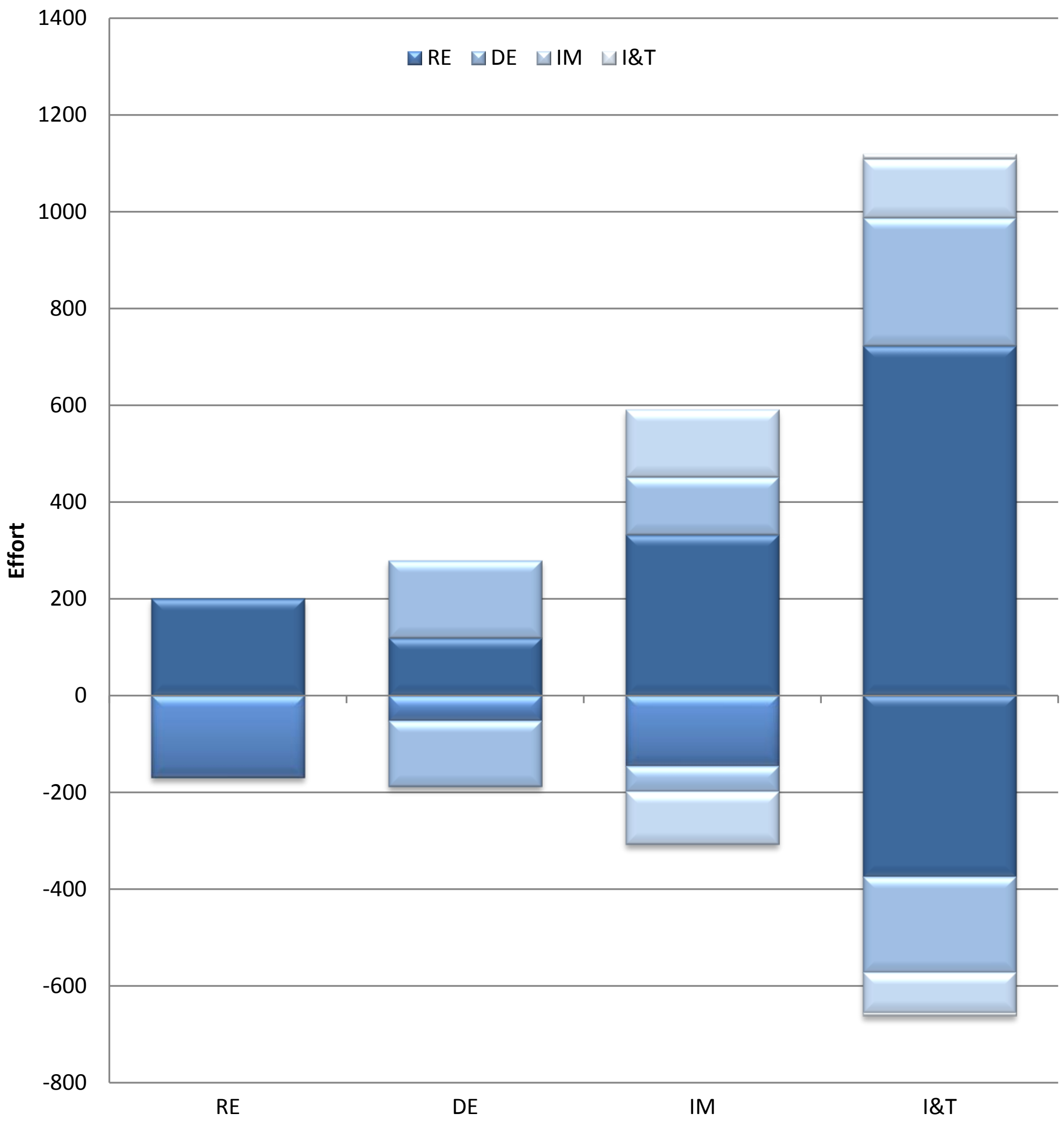


Figure 6.14: S3 DCFM chart

Table 6.15: S3 result overview

Development Effort	QA Effort	QA DCE	Residual DCE	Overall
1000 h	320 h	1328 h	457 h	3105 h

Scenario S3 is expected to be too cost intensive due to its maximum QA effort spent in early phases. Figure 6.14 and Table 6.15 on page 100 summarize the simulation results. For 1000 hours of development effort, 320 hours are spent for QA activities. The DCE as part of the QA activities is 1328 hours. The DCE residing in the product is 457 hours, mostly for defects with origin in phase RE and DE. Figure 6.14 illustrates a exponential increase of DCE through all phases. However, the overall DCE in final phase I&T is low at around 1000 hours due to the very high QA activities in early phases. Furthermore, the overall DCE could be reduced by more than 50% as indicated by the bar for negative I&T values.

Table 6.16 illustrates the detailed result data. Based on 200 hours of initial DCE for defects with origin in phase RE, 170 hours could be reduced due to the very high level of QA activities. RE defects have 119 hours of DCE in phase DE. In addition to that, 160 hours of DCE are inserted for defects with origin in phase DE. These defects are reduced by 136 hours for DE defects and 52 hours for RE defects. In phase IM, there are 332 hours of RE defects, 119 hours of DE defects and 140 hours of IM defects. Most of IM defects could be detected which lead to a reduction of DCE by 109 hours. Furthermore, the DCE could be reduced by 52 hours of DE defects and 146 hours of RE defects. Finally, in phase I&T, there are 722 hours of RE defects, 265 hours of DE defects, 122 hours of IM defects and an additional 10 hours for defects with origin in phase I&T. Final QA activities reduce the DCE for I&T defects by 6 hours, 83 hours for IM defects, 198 hours for DE defects and 375 hours for RE defects. This leads to a residing DCE of 347 hours for RE defects, 67 hours of DE defects, 39 hours of IM defects, and 4 hours of I&T defects potentially detected by the customer.

Table 6.16: S3 result data

Phase	**RE**	**DE**	**IM**	**I&T**	**CU**
RE	200	119	332	722	347
DE		160	119	265	67
IM			140	122	39
I&T				10	4
I&T				-6	
IM			-109	-83	
DE		-136	-53	-198	
RE	-170	-52	-146	-375	

S4 Results

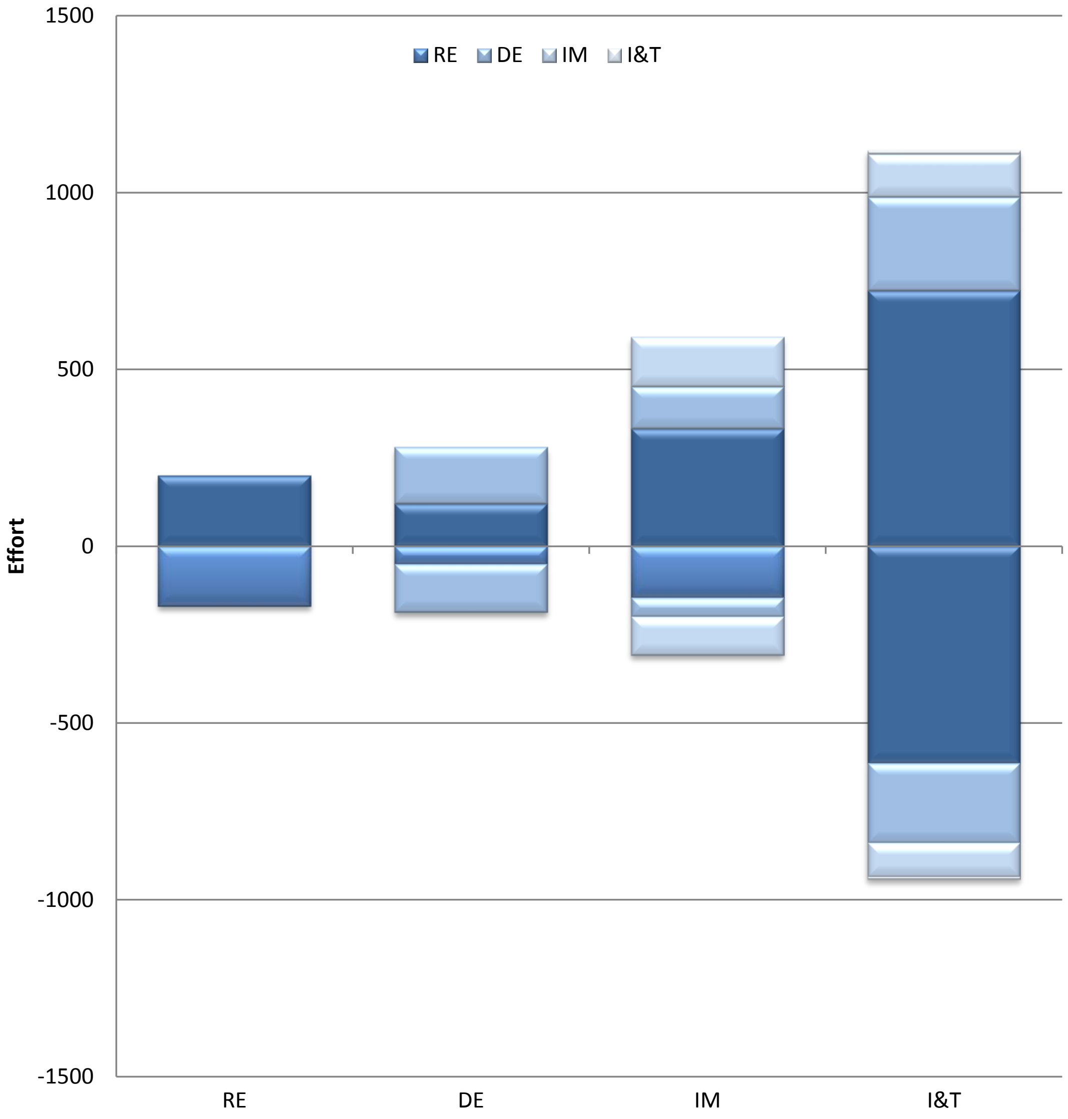


Figure 6.15: S4 DCFM chart

Table 6.17: S4 result overview

Development Effort	QA Effort	QA DCE	Residual DCE	Overall
1000 h	400 h	1608 h	177 h	3185 h

Figure 6.15 and Table 6.17 on page 102 depict the results for scenario S4 where QA activities are set to a maximum for all development phases. For a development of 1000 hours, 400 hours are spent on QA activities. This leads to a DCE of 1608 hours as part of the high QA activities resulting in 177 hours of DCE residing in the product. The overall effort needed to develop the feature is 3185 hours. Due to the focus on QA activities, the increase of DCE of all phases could be reduced to a minimum.

Table 6.18 shows for defects with their origin in phase RE an initial DCE of 200 hours, reduced by 170 hours. In phase DE, the DCE for RE defects is at 119 hours and for DE defects on 160 hours. They could be reduced by 136 hours for DE defects and 52 hours for defects with their origin in phase RE. The following phase IM has a DCE of 332 hours for RE defects, 119 hours for DE defects and an additional 140 hours for defects with their origin in this phase. The reduction of DCE for defects from earlier phases RE and DE is 146 hours and 53 hours whereas the DCE for IM defects could be reduced by 109 hours. When these defects shift to the final phase I&T, they are at 722 hours for RE defects, 265 hours for DE defects, 122 hours of IM defects and 10 hours of I&T defects. Due to the high level of QA activities, they could be reduced by 614 hours for RE defects, 225 hours for DE defects, 96 hours for IM defects and 7 hours for I&T defects. The residing DCE leading to potential defects detected by the customer are 108 hours for RE defects, 40 hours for DE defects, 26 hours for IM defects, and 3 hours for I&T defects. This final DCE is extremely low considering the phase multipliers for each development phase. For example, 108 hours divided by the overall phase multiplier of 80 hours result in 1.35 requirement defects.

Table 6.18: S4 result data

Phase	**RE**	**DE**	**IM**	**I&T**	**CU**
RE	200	119	332	722	108
DE		160	119	265	40
IM			140	122	26
I&T				10	3
I&T				-7	
IM			-109	-96	
DE		-136	-53	-225	
RE	-170	-52	-146	-614	

6.8.3 Overall Results

Table 6.19 summarizes the overall results based on the initial development effort of 1000 hours for every scenario. The effort spent on QA activities is 50 hours for scenario S1, 200 hours for S2, 320 hours S3 and 400 hours for S4. These activities lead to the effort for fixing the detected defects, 1224 hours for scenario S1, 2343 for S2, 1328 for S3 and 1608 for S4. The residual DCE after the final phase I&T is 9771 hours for S1, 528 hours for S2, 457 hours for S3 and 177 hours for S4. Finally, the overall effort needed to realize the feature is 12045 hours for scenario S1, 4071 hours for S2, 3105 for S3 and 3185 for S4.

Table 6.19: Scenario overall results

Results	**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

6.9 Conclusion

Final results from scenario simulation demonstrates the capabilities of the DCFM. They are evaluated by process as well as project experts regarding their accuracy to describe the development process where DCFM is based on. According to DCFM's results, the development process is optimized in terms of effort spent for QA activities to reduce the overall effort.

Scenario S1 was selected to demonstrate the worst case scenario regarding the overall effort needed for the development of a specific feature. The scenario indicates the consequences on saving the effort on QA activities to focus on feature development only. In early development phases, QA effort and DCE are reduced to a minimum. The DCE in early phases is low because only little effort is spent on QA activities leading to low DCE reduction rates. In later phases, especially in phase *Customer (CU)* where the product is delivered to the customer, the potential DCE is very high. It can be seen that the residing DCE for requirement defects has the largest percentage of 75%. This figure results from the necessity to re-run the complete development process for the correction of a defective requirement. For a design defect, less process phases are involved with their correction. The DCE for defects with their origin in phase DE is lower accordingly. However, it is still 20% of the final DCE. The final 5% are due to defective implementations, integration and tests. This worst case scenario confirms the strategy to focus on QA activities especially, in early development phases to reduce a project's overall DCE.

In scenario S2, every development phase is optimized in its own domain. Typical cost benefit optimization strategies regarding the optimal effort spent on QA activities tend to optimize locally. Even though every development phase has an optimal cost benefit ratio, the overall effort needed to realize the feature is very high. This is mainly because of defects with their origin in phase RE and DE, staying undetected until the final phase I&T. Even with high detection rates in phase I&T, correcting defects here from early phases is very costly. Nevertheless, the DCE residing in the product until the final phase CU is much lower compared to scenario S1. Scenario S2 also supports the strategy to intensify an invest in QA activities in early development phases.

Scenario S3 and S4 are calibrated using the maximum level of QA activities, S3 with focus on early phases, S4 throughout all phases. These scenarios demonstrate the return of QA invest. Considering the overall amount of effort, it is still cheaper to invest in QA activities close to a maximum level than in optimizing them. Especially QA activities in early phases are profitable because longer process iterations involve more process activities and therefore more effort is needed to correct a defect. The effort invested in QA activities is limited to a single development phase. Defects flowing from one development phase to later phases involve activities in these phases. The more phases involved the higher is the effort needed for fixing.

A further important aspect concerning the ideal sufficiency of QA effort, is its dependency on feature specific characteristics. There are more and less complex features resulting in different DCFs. QA activities are more effective for features with higher DCFs. Furthermore, the ideal sufficiency of QA effort per development phase depends on the involved process activities. The more activities needed for the rework of an engineering artifact, the higher the overall effort.

The overview of 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 tailoring. Furthermore, the DCFM could support higher process maturity levels, e.g. the Capability Maturity Model Integrated [CMMI Product Team, 2010], a process improvement approach also used as reference for appraising a company's engineering processes.

Next steps towards higher effort estimation performance and therefore better software projects in time, quality and costs are the establishment of this method as part of the continuous improvement process. One major part of it would be the establishment of a long term measurement framework. Based on this data, models could be further enhanced and thereby provide decision support to process optimization and project estimation.

7 Practical Benefits

This chapter describes the benefits of using the models presented in this thesis in a real-life environment. Working with such models can reveal improvement potential in various ways. Several usage areas of such models could be identified. It starts with the gain of knowledge in different areas, e.g. in modelling techniques like *Bayesian Networks (BNs)*, accompanying the work of *Software Process Model (SPM)* and *Defect Cost Flow Model (DCFM)*. These benefits are described in section 7.1. Beyond this, the *Return of Invest (ROI)* is described based on a change in the development process within the real-life environment. For this, an additional evaluation has been carried out resulting from SPM and DCFM which is presented in section 7.2.

7.1 Developing Knowledge

It is challenging to create predictive models on a specific software development process because expert knowledge is developed in multiple domains. There are several aspects to be considered when building predictive models, e.g. the SPM and DCFM for software defect prediction, i.e.

- What are the *Key Performance Indicators (KPIs)* leading to software defects?
- What are the KPIs of the target development process?
- What is the adequate modeling technique?
- What process data is available for model calibration?

The following sections introduce these aspects and describe their benefits regarding the gain of knowledge for the organization working such models.

7.1.1 Subject Area

An expert in the theory of software engineering is needed to describe the domain of the model in general. For this thesis for example, a deep understanding on software defect prediction is necessary, especially on KPIs leading to software defects. This includes knowledge on general techniques to describe software defects as part of a development process as well as defect prediction specifics, e.g. detection rates for different *Quality Assurance (QA)* techniques.

This knowledge is of high value for an organization to not only build complex models on defect prediction but also to define and continuously improve its processes with regards to the theory of software engineering.

7.1.2 Field of Application

Furthermore, if models are intended to be used in real-life, as they are for this thesis, there has to be a deep understanding of the model's field of application. The development processes behind the models developed in this are needed to build large scale embedded systems for the automotive industry. On a single product, potentially more that 100 engineers are working simultaneously in a distributed environment. Such complex process structures evolve over time and adapt to specific boundary conditions. Thus, it is very challenging to model software defects as part of such a development process.

Process experts are essential for an organization because they understand the complex structure of their current development process. Only with the help of process experts, it is possible to predict the impact of process changes which are essential to the continuous improvement of an organization.

7.1.3 Modelling

There is a variety of defect prediction models in existence. The most relevant for this thesis are discussed in section 2.4. One of the most well known models is the *Constructive Cost Model (COCOMO)* mentioned already in the previous section 8.4.

Models like COCOMO are general purpose models implementing universal correlations in software engineering, e.g. the more test activities are carried out, the more defects are found. Due to the generality of these models, they are often large and complex covering a wide range of KPIs. With such models it is possible to get a general understanding of the KPIs leading to software defects.

The major challenge of establishing a general model in a real-life software development environment is its calibration. The more parameters are used in a model, the more complex it is to calibrate. Furthermore, it is important to have a detailed understanding of the model, of every single KPI, how it is related to the others, and especially how it influences the overall model calculation.

Modeling experts have a good understanding of translating a given problem into a mathematical or even graphical representation. With the help of the resulting model, the problem can be assessed from various perspectives. The modelling expert is of high value for an organization, because he links the theoretical experts to the practice. The potential for improvement increases further with the help of modelling experts.

7.2 Process Changes

This section shows the benefit of using the models presented in this thesis in a real-life environment. Although it is very challenging to build models like the SPM and DCFM it can be shown that working with these models can reveal improvement potential in various ways. The models have been calibrated based on real data and expert knowledge from the target environment. Expert knowledge was needed to describe process dependencies, e.g. relevant development phases as well as the corresponding metric databases. Historical project data has been used to calibrate process dependencies among each other, e.g. specific *Defect Cost Factors (DCFs)* for specific features lead to an environment specific *Defect Correction Effort (DCE)* which was the base for optimization. After that, the resulting DCFM has been used to simulate various hypothetical scenarios. For further analysis, scenario S2 and S3 described in the previous chapter 6 were focused. where scenario S2 represents the current level of QA in the real-life development environment. Scenario S2 uses a high level of QA activity equal distributed over all development phases. Such a scenario is typically used to optimize a single development phase only. Scenario S3 is an optimization of scenario S2. Scenario S3 uses very high QA activities in early phases RE and DE. Such QA effort typically is often too expensive considering the return of invest per development phase. However, the DCFM calculates an overall effort reduction for scenario S3.

The following subsections discus the results of a review process change, introduced based on several analyses from the model's prediction results. The process change has been evaluated as part of this thesis.

7.2.1 Prior Predictions

Table 7.1 depicts the DCFM overall results which has been discussed in the previous chapter 6.

Table 7.1: DCFM overall results

Results	**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

Scenario S2 represents the current level of QA in the real-life development environment. Comparing scenario S2 with S3, the model predicts an overall effort reduction from 4071 hours to 3105 hours. This can be achieved by increasing the effort for QA from 200 hours to 320 hours.

Based on these predictions, the real-life development processes has been adapted accordingly. This results in an change for the level of QA activity in phases *Requirements Engineering (RE)*, *Design (DE)* and *Implementation (IM)* from High to Very High. In effort, it can be expressed as an increase for QA effort from 20% to 40% of the development effort. It should be noted that in scenario S3 the level of QA activity in phase IM is still on High unlike the process change in the real-life environment. With this process change an overall effort reduction of approximately 30% is expected.

7.2.2 Review Data

The evaluation of the process change lasted over a period of three months. A single project was selected as the evaluation project for the process change. It mainly intensifies the review activities in phases RE, DE and IM.

Figure 7.1 illustrates the overview on the evaluated data including the effort spent on development and reviews. The effort spent in phases RE and DE is 320 hours. An additional effort of 41 hours is spent on reviews in these phases, 13% of the development effort respectively. The effort spent on implementation on phase IM is 80 hours. For implementation reviews 16 hours are spent representing 20% of the development effort. Summing up, there is an overall effort of 400 hours of development activities that have been reviewed. The overall effort spent on reviews is 57 hours.

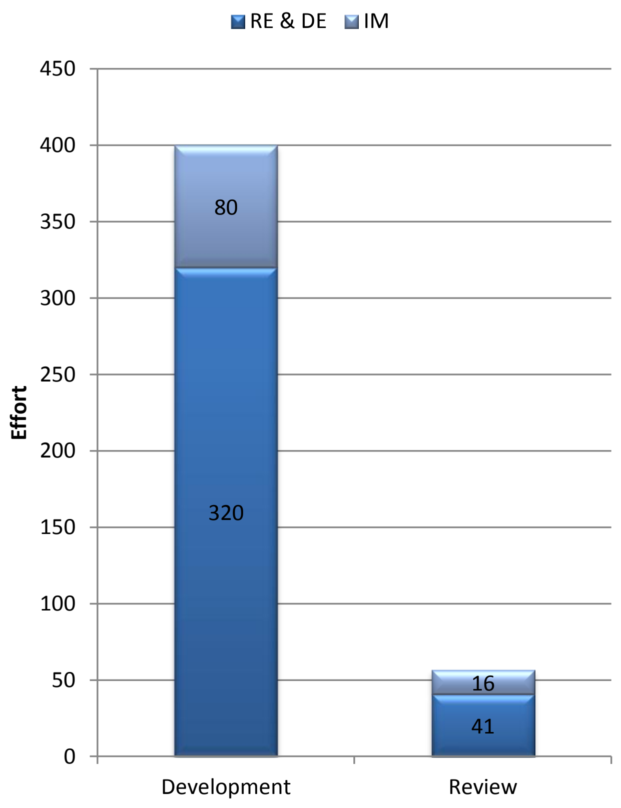


Figure 7.1: Effort

Figure 7.2 depicts all detected defects and comments during review. Comments are given, if the finding is not an obvious defect but further clarification is needed. It is assumed to have one additional defect on every fourth comment. In phase RE and DE the reviewers identified 8 defects and 28 comments leading to an overall of 15 defects detected in phases RE and DE. Based on these reviews, one defect and 5 comments could be identified leading to an overall of 2.25 defects detected in phase IM. With the help of these additional reviews 9 defects are detected as well as 33 comments.

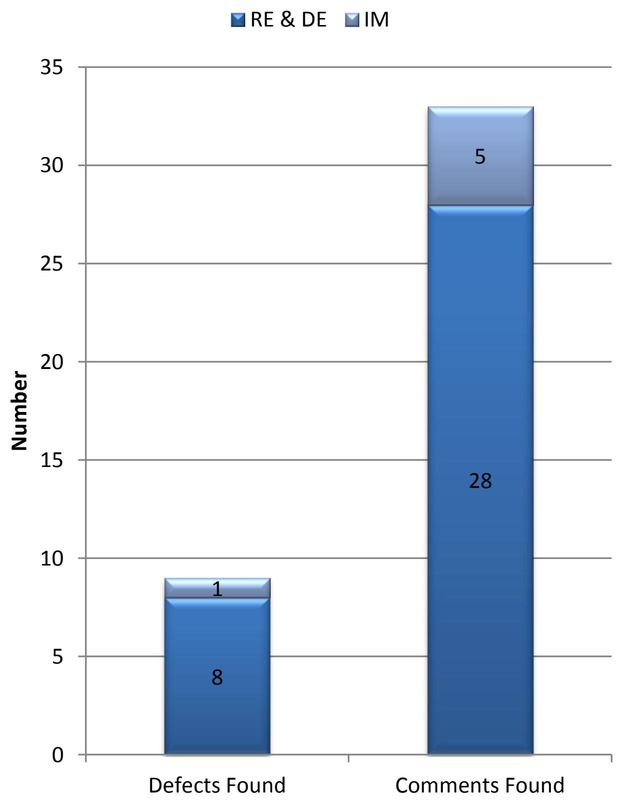


Figure 7.2: Occurrence

7.2.3 Evaluation

A review performance analysis is is not possible purely based on the number of detected defects. To solve this, the concept of the DCF is taken. It determines the potential DCE per defect for a specific feature based on statistical data. The data set is described earlier in chapter . The performance analysis uses the DCF in combination with the corresponding development effort to determine the potential DCE. Furthermore, the average DCE per defect in its specific development phase is taken to express the detected defects in form of their potential DCE.

For this evaluation a DCF of 0.75 is assumed due to the high difficulty of the feature development for this specific project. The phase multiplier for defects shifting from one phase to another is 20 for undetected defects from phases RE and DE. For undetected defects with origin in phase IM it is 4. The average DCE for all phases RE, DE and IM for one defect to be fixed is 10 hours. To evaluate the overall performance, the effort needed for integration & tests needs to be taken into account which is, based on internal cost estimation tables, an additional 40% of the average DCE. The results are shown in figure 7.3.

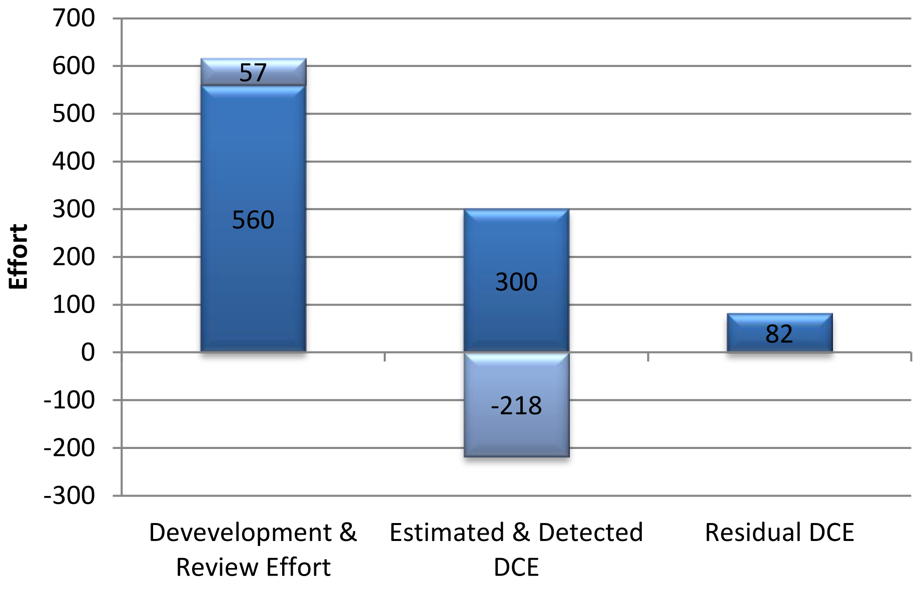


Figure 7.3: Review evaluation results

The overall development effort is 560 hours. Effort spent on QA activities is 57 hours. For this scenario, the DCFM estimates an overall DCE of 300 hours. With the process change suggested by the DCFM, additional defects could be detected and corrected. The identified defects were analyzed by engineering experts. Based on the experience of experts in combination with historical data, the potential DCE saved could be estimated to 218 hours resulting to 82 hours of residing DCE. This value served as a base for analysis of both the model and the process performance.

Thus, the overall defect detection rate is 73% (i.e. 218 hours / 300 hours). Including effort spent on review activities the overall effort is 699 hours (i.e. 560 hours + 57 hours + 82 hours). Finally, the overall effort saving is approximately 19% (i.e. 1 Ű 699 hours / 860 hours) thanks to performing additional reviews. This result proves the justification for these additional reviews. This ratio of 19% is close to the predicted reduction of 24% what proves the reasonable accuracy of the DCFM.

The evaluation results are based on a DCE estimated by experts which are treated as real empirical values. This is a validation limitation because experts might not have estimated the DCE correctly. However, it is not possible to assess the accuracy of this assessment because defects, for which the DCE was estimated were actually detected and fixed during reviews and not passed to the next phases to see how much it would really take to correct them.

8 Summary and Outlook

This chapter summarizes the results of this thesis and gives an outlook on future activities. First, the results from the *model analyses* are discussed in section 8.1. It is followed by the *novel contributions* of this thesis in section 8.2. This section also explains to what extent these contributions confirm the hypothesis of this thesis defined in section 1.2. After this, section 8.4 points out the most important *research activities* that have been carried out during this thesis. The consequential *process optimizations* resulting from this thesis for the development of automotive application within the Robert Bosch GmbH are described in section 8.4. Finally, the *perspectives* are presented in section 8.6.

8.1 Model Analyses

8.1.1 Software Process Model

Even though many metric related effort estimation programs have been established for a long period of time, the final breakthrough and commitment to metric based software quality assessment is still missing [Ordonez and Haddad, 2008]. These metric programs form the basis for further effort estimation methods such as *Software Process Model (SPM).* The SPM shows that based on accurate data it is possible to handle complex structures as well as incomplete and changing data. To achieve the goal of building a model based on *Bayesian Networks (BNs)* to support decision makers, it is crucial to have data representing the real world. It is essential to have a positive way of managing errors so that employees as well as *Project Managers (PMs)* can use a system for tracking defects that potentially have negative influence on their reputation. Furthermore, employees need to be convinced that continuous improvement programs alleviate daily work. With the commitment to such programs the results of SPM show that it is possible to predict *Defect Correction Effort (DCE)* at high accuracy. The SPM demonstrates the potential of using a system based on BNs to support decisions in the context of software effort estimation. Even though it is very challenging to build such complex models, BNs include a wide range of benefits. They are based on objective process and product data as well as on expert knowledge and offer the possibility to support decision makers. BNs enable them to justify and document their decisions. Based on the graphical representation of cause and effect it is easy to discuss even complex problems and identify weaknesses or even new aspects in such complex domains as software effort estimation.

8.1.2 Defect Cost Flow Model

The *Defect Cost Flow Model (DCFM)* predicts the lowest DCE for a maximum effort spent on *Quality Assurance (QA)* measures in early development phases. Later phases are predicted to have their benefit cost-optimized. Typical cost-benefit optimization strategies regarding the optimal effort spent on quality measures tend to optimize locally. For example, every development phase is optimized separately in its own domain. In contrast to this, it is demonstrated that even cost intensive quality measures pay for themselves 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 *Defect Cost Factor (DCF)* for the feature to be developed. Furthermore, the DCE depends on the process activities involved per development phase. The more activities involved in the rework of an engineering artifact, the higher the overall DCE. The overview on the influence of all *Key Performance Indicators (KPIs)* of a software product is very complex. With the DCFM, PMs 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)* [CMMI Product Team, 2010], a process improvement approach also used as a reference to appraising a company's engineering processes.

8.2 Novel Contributions

8.2.1 Defect Cost Factor

In the course of this thesis, the need for a measure is identified to describe the amount of rework for a specific defect. The amount of rework is often expressed as the number of defects. It is defined as DCF, as an indicator for the rework of defects. The DCF is defined for every high level function, i.e. named features, of the software product to describe different defect rates for different features. Besides the potential DCE, the DCF incorporates the development effort as reference value to normalize the DCE over all features (see 4.4).

8.2.2 Software Process Model

The SPM was developed to represent all changes of a software release as part of a *Dynamic Bayesian Network (DBN)*. For each of these potentially hundreds of changes, a separate BN is created including its unique change characteristics and linked to the overall release respectively the DBN. The SPM enables PMs to assess process, product or project changes over time (see 5).

8.2.3 Defect Cost Flow Model

A different impact on the overall DCE could be recognized for defects originating from other development phases than the one they were detected in when compared to defects detected in the same development phases in which they originated. Therefore, to reduce the DCE of the software product, every development phase is assessed along with its specific DCE characteristic and focus on the shift of defects through the development phases.

These ideas were realized in the brand new DCFM. The main goal of the DCFM is the identification of process areas where optimization leads to the lowest defect rates and the 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 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 (see 6).

8.3 Hypothesis Confirmation

At the beginning of this thesis, the following hypothesis had been defined:

H1 It is possible to develop estimation models for development effort and DCE that incorporate process data and expert knowledge in the absence of significant data.

H2 It is possible to enable PMs to identify product areas where additional effort spent on defect rate reduction has a high cost-benefit ratio.

H3 It is possible to incorporate the supporting development process to distribute effort for QA measures most effectively.

Hypothesis H1 can be confirmed based on the results from SPM and DCFM. Both models showed their capabilities to incorporate both process data as well as expert knowledge within a single model. Furthermore, both models represent an industry based development process using real data from process databases as well as experts. Furthermore, their results could be used to estimate development effort in combination with its DCE at high accuracy.

With the help of the DCF, it is possible to confirm hypothesis H2. Models like SPM and DCFM using the DCF enable PMs to define DCF thresholds and classify error-prone features, independent of the development effort used to realize these feature. This allows the focus of QA measures on product areas where optimization leads to high benefits in terms of a reduction of rework.

Hypothesis H3 can be confirmed based on the change of suggested process areas identified by the DCFM. The results of the process change evaluation depict the capabilities of DCFM to identify development phases where optimization leads to an overall cost reduction. Classical optimization strategies tend to optimize with focus on a single only. In contrast to that, DCFM is capable to optimize single phases with focus on the overall effort.

8.4 Research Activities

Several research activities have been carried out during this thesis. Their main goal is to illuminate different aspects and assure state-of-the-art research in the domain of this thesis, i.e. the potential of BNs and how they can be used to assess our software products, -projects and -development processes.

We had our first publication in [Schulz et al., 2008] with a presentation on how BNs can be used to describe automotive software engineering processes [Schulz et al., 2008].

At the same time, we carried out a workshop for the assessment of risks coming along with the development of safety critical functions, e.g. the autonomous emergency brake for vehicles. Therefore, we consulted Prof. Norman Fenton from the Queen Mary University of London, one of the leading experts in this field. The results of this workshop are documented as a part of a separate bachelor thesis [Weibert, 2009].

Furthermore, a cooperation with Dr. Lukasz Radlinski was started, who obtained his Ph.D. as a part of the research team around Prof. Norman Fenton [Radlinski, 2008]. Our part of this cooperation was to supply data and expert knowledge along with the backgrounds behind the development of fail-safe automotive applications. On the other side, Dr. Radlinski assured state-of-the-art research in our fields. Two publications resulted from this cooperation: [Schulz et al., 2010a,b]. After presenting the DCFM in [Schulz et al., 2010b], we had been invited to publish further details and analysis in [Schulz et al., 2011].

In [König, 2008] we analyzed the potential of the *Constructive Cost Model (CO-COMO)*, a widely known software cost estimation model first published in 1981 by Barry Boehm. The results of this work showed that it is difficult to calibrate generic cost estimation models such as COCOMO. Furthermore, it is difficult to adapt the structure of these models to company specific characteristics. As a consequence, the goal of this thesis was to pursue the approach to build completely new models reflecting the main characteristics of our domain.

The topic of how to calibrate those parts of the models for which there was no data available in [Wuchenauer, 2009] was discussed. With this work we wanted to understand how data can be collected with the help of a *Design of Experiment (DOE)* and use it to calibrate our models. Therefore, a DOE was set up with the main focus on identifying the relation between a project's deadline pressure and resulting software defects. The DOE was partly conducted by students of the University of Stuttgart and engineers of the Robert Bosch GmbH. The results showed that in general it is possible to calibrate specific parts of our models with the help of a DOE. However, it is very expensive to calibrate models for an industrial application in their own context solely based on DOEs. For this thesis we could use the influence of deadline pressure on software defects as one of the main indicators for the SPM.

8.5 Process Optimization

The results of this thesis are used to support several process improvement activities. One conclusion from the results of the SPM is to focus on QA measures not only at the end of a project but throughout all development phases. Furthermore, deadline pressure especially at the end of a projects leads to higher defect rates. The resulting DCE demands even more effort to finalize the project. These results support our *Quality for Feature* initiative focusing on low defect rates instead of extra functionality.

The results of the DCFM are used to focus the QA measures in early phases of a project. New methods for requirements-, design- as well as code reviews have been established in a first project. Here, every work package is reviewed by the *Integration & Test (I&T)* engineers before it is closed. First analyses of review protocols show similar results as predicted by the DCFM. In a second step, these new methods will be established in every future project.

Based on the measurement concept introduced in this thesis it is possible to assess the benefit of QA measures. Development effort, DCE and the DCF are the main variables. Hence, the benefit of reviews is expressed based on their average DCE reduction. An example: for a specific feature we spend 80 hours on requirements engineering and take 12 hours for review activities. Three defects are detected by the review. Based on the DCF for the feature, we estimate a potential DCE of 47.2 hours.

The average DCE for one of these defects is 7.07 hours. For the three detected defects, it means a DCE reduction of 21.2 hours. Taking the effort spent into account, the effort saved can be expressed as 9.2 hours or 19% compared to the performance of the previous QA activities.

8.6 Perspectives

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 a part of the continuous improvement process. One major part of this, is the establishment of a long term measurement framework. Based on the collected data, we could further enhance our models and thereby provide precise decision support to process optimization and project estimation.

Furthermore, if we consider effort instead of number of defects, changes or review findings, this would allow a consistent assessment of the software product, projects and development processes. Effort as a metric is more accurate and therefore leads to a better estimation performance.

The collection of data not only for the overall development process but also for specific features enables the focus on product areas where optimization leads to highest benefit-cost ratios.

Bibliography

Central directive quality - cdq0302 software development. Robert Bosch GmbH Central Directive (Internal), May 2010.

M. Abouelela and L. Benedicenti. Bayesian network based xp process modelling. *International Journal of Software Engineering & Applications*, 1:1–15, 2010.

AgenaRisk. URL http://www.agenarisk.com/.

V. R. Basili and H. D. Rombach. The tame project. towards improvement-oriented software environments. *IEEE Transactions on Software Engineering*, 14(6):758–773, 1988.

T. Bayes. An essay towards solving a problem in the doctrine of chances, 1763.

S. Bibi and I. Stamelos. Software process modeling with bayesian belief networks. In *Proc. Int. Software Metrics Symposium*, 2004.

E. Castillo, Guti, J. M. Gutiérrez, J. M. Gutîaerrez, and A. S. Hadi. *Expert Systems and Probabilistic Network Models*. Springer, 1997.

Center for Systems and Software Engineering. Cocomo2, 1995. URL http://sunset.usc.edu/csse/research/COCOMOII/cocomo_main.html.

R. Chillarege, I. Bhandari, J. Chaar, M. Halliday, D. Moebus, B. Ray, and M.-Y. Wong. Orthogonal defect classification-a concept for in-process measurements. *IEEE Transactions on Software Engineering*, 18:943–956, 1992. ISSN 0098-5589. doi: http://doi.ieeecomputersociety.org/10.1109/32.177364.

CMMI Product Team. Cmmi for development version 1.3. Technical Report CMU/SEI-2010-TR-033, Software Engineering Institute (SEI), 2010.

J. Dabney, G. Barber, and D. Ohi. Predicting software defect function point ratios using a bayesian belief network. In *Proc. 2nd Int. Workshop on Predictor Models in Software Engineering*, 2006.

A. Darwiche. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press, 1 edition, 4 2009. ISBN 9780521884389.

A. C. V. de Melo and A. J. Sanchez. Software maintenance project delays prediction using bayesian networks. *Expert Syst. Appl.*, 34(2):908–919, 2008. ISSN 0957-4174. doi: http://dx.doi.org/10.1016/j.eswa.2006.10.040.

J. del Sagrado and I. M. d'Aguila. *Artificial Intelligence Applications for Improved Software Engineering Development: New Prospects*, chapter A Bayesian Network for Predicting the Need for a Requirements Review, pages 106–128. IGI Global, 2010.

N. Fenton, W. Marsh, M. Neil, P. Cates, S. Forey, and M. Tailor. Making resource decisions for software projects. In *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, pages 397–406, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2163-0.

N. Fenton, M. Neil, W. Marsh, P. Hearty, D. Marquez, P. Krause, and R. Mishra. Predicting software defects in varying development lifecycles using bayesian nets. *Inf. Softw. Technol.*, 49(1):32–43, 2007a. ISSN 0950-5849. doi: http://dx.doi.org/10.1016/j.infsof.2006.09.001.

N. Fenton, M. Neil, W. Marsh, P. Hearty, L. Radlinski, and P. Krause. On the effectiveness of early life cycle defect prediction with bayesian nets. *Empirical Softw. Engg.*, 13(5):499–537, 2008. ISSN 1382-3256. doi: http://dx.doi.org/10.1007/s10664-008-9072-x.

N. Fenton, P. Hearty, M. Neil, and L. Radlinski. *Artificial Intelligence Applications for Improved Software Engineering Development: New Prospects*, chapter Software Project and Quality Modelling Using Bayesian Networks, pages 1–25. IGI Global, 2010.

N. E. Fenton, M. Neil, and J. G. Caballero. Using ranked nodes to model qualitative judgments in bayesian networks. *IEEE Trans. on Knowl. and Data Eng.*, 19(10):

1420–1432, 2007b. ISSN 1041-4347. doi: http://dx.doi.org/10.1109/TKDE.2007.1073.

GeNIe. URL http://genie.sis.pitt.edu/.

P. Hearty, N. Fenton, D. Marquez, and M. Neil. Predicting project velocity in xp using a learning dynamic bayesian network model. *Software Engineering, IEEE Transactions on*, 35(1):124 –137, jan.-feb. 2009. ISSN 0098-5589. doi: 10.1109/TSE.2008.76.

Hugin. URL http://www.hugin.com/.

IABG. The v-model - general directive 250. software development standard for the german federal armed forces., 1992. URL http://v-modell.iabg.de/.

F. V. Jensen. *Introduction to Bayesian Networks*. Springer, 1 edition, 8 1997. ISBN 9780387915029.

C. Jones. *Software Quality: Analysis and Guidelines for Success*. International Thomson Computer Press, 6 2000. ISBN 9781850328674.

M. Klaes, T. Beletski, and A. Sarishvili. Ap 3.1: Effektivität von qs-maSSnahmen stand der wissenschaft. *Fraunhofer IESE-Report*, 096.07/D:40, 2007. doi: urn:nbn:de:0011-n-692908. URL http://publica.fraunhofer.de/.

B. König. Aufwandschätzung mit bayes'schen netzen. Master's thesis, Eberhard Karls Universität Tübingen, 2008.

T. J. O. Leary, M. Goul, K. E. Moffitt, and A. E. Radwan. Validating expert systems. *IEEE Expert: Intelligent Systems and Their Applications*, 05(3):51–58, 1990. ISSN 0885-9000. doi: http://doi.ieeecomputersociety.org/10.1109/64.54673.

I. Lee, J. Y.-T. Leung, and S. H. Son. *Handbook of Real-Time and Embedded Systems*. Chapman & Hall/CRC, 2007. ISBN 1584886781, 9781584886785.

R. E. Neapolitan. *Learning Bayesian Networks*. Prentice Hall, illustrated edition edition, 4 2003. ISBN 9780130125347.

M. J. Ordonez and H. M. Haddad. The state of metrics in software industry. In *Information Technology: New Generations*, 2008. doi: 10.1109/ITNG.2008.106. URL `http://ieeexplore.ieee.org/iel5/4492437/4492438/04492521.pdf`.

J. Pearl. *Bayesian networks: A model of self-activated memory for evidential reasoning (Report. University of California, Los Angeles. Computer Science Dept)*. UCLA, Computer Science Dept, 1985.

J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1 edition, 9 1988. ISBN 9781558604797.

Promisedata. URL `http://promisedata.org/`.

L. Radlinski. *Improved Software Project Risk Assessment Using Bayesian Nets*. PhD thesis, Queen Mary University, London (unpublished), 2008.

L. Radlinski, N. Fenton, M. Neil, and D. Marquez. Improved decision-making for software managers using bayesian networks. Technical report, School of Electronic Engineering and Computer Science Queen Mary University of London, 2007.

L. Radlinski, N. Fenton, and M. Neil. A learning bayesian net for predicting number of software defects found in a sequence of testing. *Polish Journal of Environmental Studies*, 17:359–364, 2008.

S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach (2nd Edition)*. Prentice Hall, 2 edition, 12 2002. ISBN 9780137903955.

R. G. Sargent. Verification and validation of simulation models. In *WSC '05: Proceedings of the 37th conference on Winter simulation*, pages 130–143. Winter Simulation Conference, 2005. ISBN 0-7803-9519-0. URL `http://portal.acm.org/citation.cfm?id=1162708.1162736#`.

T. Schulz, T. Gorges, and W. Rosenstiel. Bayesian networks modelling automotive software engineering processes. In *IP08*, 2008.

T. Schulz, L. Radlinski, T. Gorges, and W. Rosenstiel. *Knowledge Engineering for Software Development Life Cycles: Support Technologies and Applications*, chapter Software Process Model using Dynamic Bayesian Networks. IGI Global (in press), 2010a.

T. Schulz, L. Radlinski, T. Gorges, and W. Rosenstiel. Defect cost flow model - a bayesian network for predicting defect correction effort. In *PROMISE '10*, 2010b.

T. Schulz, L. Radlinski, T. Gorges, and W. Rosenstiel. Predicting the flow of defect correction effort using a bayesian network model. *Empirical Software Engineering*, 2011.

R. E. Shannon. *Systems Simulation: The Art and Science*. Prentice Hall, 1975.

B. Stewart. Predicting project delivery rates using the naive-bayes classifier. *Journal of Software Maintenance*, 14(3):161–179, 2002. ISSN 1040-550X.

W. Stolz and P. Wagner. Introduction of a quantitative quality management for the ecu software development at gasoline systems. Technical report, Robert Bosch GmbH, 2005.

R. Torkar, N. Adnan, A. Alvi, and W. Afzal. Predicting software test effort in iterative development using a dynamic bayesian network. In *Proc. of 21st IEEE International Symposium on Software Reliability Engineering*, 2010.

S. Wagner. A bayesian network approach to assess and predict software quality using activity-based quality models. In *PROMISE '09: Proceedings of the 5th International Conference on Predictor Models in Software Engineering*, pages 1–9, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-634-2. doi: http://doi.acm.org/10.1145/1540438.1540447.

X. Wang, C. Wu, and L. Ma. Software project schedule variance prediction using bayesian network. In *Advanced Management Science (ICAMS), 2010 IEEE International Conference on*, volume 2, pages 26 –30, 2010. doi: 10.1109/ICAMS.2010.5552847.

M. Weibert. Risikobewertung von sicherheitskritischen funktionen auf basis von bayes'schen netzen. Master's thesis, Hochschule Furtwangen, 2009.

S. M. Weiss. *A Practical Guide to Designing Expert Systems*. Rowman & Littlefield Publishers, Inc., 1 1984. ISBN 9780865981089. URL `http://amazon.com/o/ASIN/0865981086/`.

R. Winkler. *An Introduction to Bayesian Inference and Decision, Second Edition*. Probabilistic Publishing, 2nd edition, 1 2003. ISBN 9780964793842.

D. A. Wooff, M. Goldstein, and F. P. A. Coolen. Bayesian graphical models for software testing. *IEEE Trans. Softw. Eng.*, 28(5):510–525, 2002. ISSN 0098-5589. doi: http://dx.doi.org/10.1109/TSE.2002.1000453.

V. Wuchenauer. Ein empirisches vorgehen zur prognostizierung von softwarefehlern auf basis ausgewählter entwicklungsprozessgröSSen eingebetteter software im automobilbereich. Master's thesis, University of Stuttgart, 2009.

K. W. Zimmermann and H.-M. Hauser. The growing importance of embedded software: Managing hybrid hardware-software business. Technical report, The Boston Consulting Group, Inc., 2004. URL http://www.bcg.com/impact_expertise/publications/files/The_Growing_Importance_of_Embedded_Software_Sep04_rpt.pdf.