Software Development and Correction Estimation in the Automotive Domain

Dissertationsschrift zur Erlangung des akademischen Grades Doktor-Ingenieur (Dr.-Ing.)

vorgelegt der Fakultät Informatik und Automatisierung
der Technischen Universität Ilmenau

von M.Sc. Shahab Ahmed Nadir

Gutachter:

Dr.-Ing. Detlef Streitferdt, TU Ilmenau
Jun.-Prof. Patrick Mäder, TU Ilmenau
Prof. Dr. Christian Siemers, TU Clausthal

Eingereicht am 26.10.2016
Verteidigt am 20.10.2017

urn:nbn:de:gbv:ilm1-2017000761

ACKNOWLEDGEMENTS

Firstly, I would like to express my special appreciation and thanks to my advisor Professor Detlef Streitferdt for providing me with the opportunity to complete my Ph.D. thesis at Technical University of Ilmenau as an external researcher. I especially want to thank my first advisor Prof. Dr. Ilka Philippow, whose support and guidance made my research possible.

I want to thank the present and past colleagues who were all times available to support this work. I especially want to thank my first group manager Michael Rölleke who motivated me to start this research.

Last but not the least, I would like to thank my beloved wife, Carmen Wohlfrom and my children Haron and Hannah for their patience and understanding.

ZUSAMMENFASSUNG

Während der letzten Jahrzehnte hat sich Software in alle Lebensbereiche ausgebreitet. Die kontinuierlich steigenden Kundenanforderungen ließen auch die Komplexität steigen, bei gleichbleibender Produktqualität. Analysedaten und diverse Beispiele entstammen der Automobildomäne, die einen sicherheitskritischer Bereich darstellt, in dem Produkte mit speziellen Qualitätsanforderungen entwickelt werden. Qualitätsanforderungen müssen von diversen Prozessen und Standards bedient werden, bei gleichzeitiger Einhaltung enger Endtermine. Die Komplexität der Software und der Safety-Aspekt beeinflussen die Fehlerquote der Produkte stark. Viele Anforderungen werden während der Entwicklung hinzugefügt oder verändert und führen zu permanenten Änderungen in der Software und einer weiteren Steigerung der Komplexität. Änderungen müssen analysiert und getestet werden, um die Qualität des entstehenden Produktes zu gewährleisten. Die Vorhersage von Defekten und Änderungen in der Software sind ein wichtiger Anteil des Software Engineering.

Die industrielle Software-Entwicklung muss ihr Ziel innerhalb diverser Grenzen erreichen, ganz wichtige ist das Budget, wobei sich Änderungen an Projektparametern negativ auf das geplante Budget auswirken können. Solche Änderungen werden in zwei Klassen eingeteilt, durch Kunden verursachte neue oder veränderte Anforderungen, und die Korrekturen, die durch Systemverbesserungen oder Fehlerbehebungen entstehen, beide Klassen für das Projekt-Budget relevant. Die Aufwände für die neuen Kundenanforderungen können dem Budget einfach aufgeschlagen werden. Die Korrekturen verursachen ebenfalls große Aufwände, die zu einem negativen Budget führen können, was eine große Herausforderung für das Projektmanagement wie auch die automatisierte Schätzung der Aufwände über die gesamte Projektlaufzeit darstellt.

Diese Dissertation bietet dafür ein neues Modell an, um die Aufwandsschätzung mit unterschiedlichen Perspektiven zu verbessern. Dieses Modell integriert Folgeaufwände, die in späteren Entwicklungsphasen entstehen. Somit ist der Fehler-Kosten-Fluss Teil des Modells und ermöglicht die Verwaltung von Fehlern und Folgefehlern, die über alle Entwicklungsphasen verteilt sein können. Das neue Modell wurde erfolgreich innerhalb der Automobildomäne evaluiert. Die Verbesserung der Genauigkeit der Aufwandsschätzungen beträgt 80%.

ABSTRACT

Over the past decades, software has spread to most areas of our lives. The complexity increased due to steadily increasing customer demands and, at the same time, the high quality of the products had to be kept. The data for the analyses and many of the examples are taken out of the automotive software development domain. The automotive domain is a safety-critical area where products are developed with specific quality requirements. These quality requirements have to be met by many processes and by satisfying several standards within stipulated deadlines during the development lifecycle. The complexity of the software and the safety aspect have a strong influence on the product defect ratio. Many requirements will be added and adjusted during the development lifecycle leading to continuous changes in the software and increased complexity. All these changes need to be analyzed and tested to ensure the quality of the product. Predicting software defects and changes is a significant part of software engineering.

Industrial software development has to achieve its target within several boundaries. One of the important boundaries for an industrial project is the budget, where changes of any project parameters can easily lead to negative effects in the planned budget. Such changes are classified into two types, the changes pushed by the customer as new requirements or changed requirements, and the correction changes in the project because of improvements of the system and identified bugs with their fixes. This classification is important to control the project budget. The effort for the realization of new customer changes can be estimated and added to the budget. The correction changes also cause huge efforts, which can lead to a negative budget in the project which is a big challenge for the project management, the automated calculation of effort estimations for the complete development life-cycle.

This thesis offers a new model to improve the effort estimation from multiple perspectives. This model also integrates follow-up-defects in later process phases. Thus, the defect cost flow is part of the model and enables the management defects and follow-up defects which could spread throughout the development phases. The newly developed model was successfully evaluated in the automotive domain. The overall accuracy of the effort estimations was improved by 80%.

A

APP *Application layer*
ASIL *Automotive Software Process Improvement and Capability Determination, Automotive Safety Integrity Level*
Autosar *AUTomotive Open System ARchitecture*

C

CBR *Case Based Reasoning*
CEFMod *Flow mode of Correction Effort. Flow mode of Correction Effort*
CMMI *Capability Maturity Model Integration, Capability Maturity Model Integrated*
COCOMO *COnstructive COst MOdel, COnstructive COst MOdel*
CONC *Concentration*
CQM *Change Request Management*
CRM *Change Request Management*
CRQ *Change Request*
CUS *Customer phase*

D

DCF *Defect Cost Flow*
DES *Design phase*
DevE Development Effort
DOF *Degree of Focus*
DOS *Degree of Scattering*
DRBFM *Design Review Based on Failure Mode*
DSM *Design Structure Matrix*

E

ECU *Electronic Control Unit*
EoDC *Effort of development/ Defect correction*

F

FEP *Failure Effort Parameter*
FMEA *Failure Mode and Effects Analysis*
FPA *Function Point Analysis*
FSM *Functional Size Estimation*

G

GQM *Goal Question Metric*

H

HCC *HMI Control and Coordination*
HMI *Human Machine Interface*

I

IEC *International Electrotechnical Commission*
IMP *Implementation phase*
ISO *International Organization for Standardization, International Organization for Standardization*

K

KPI *Key Performance Indicators*
KPIs *Key Performance Indicator*

M

MCAL *Microcontroller layer*

O

OEM *Original Equipment Manufacturer*

P

PF *Productivity Factor*
PRM *Process Reference Model*

Q

QA *Quality assurance*

R

RCM *Required Change Management*
REQ *Requirement phase*
RM *Requirement Management system*
RTE *Run Time Environment*

S

SCE *Software Correction Effort*
SIG *Automotive Special Interest Group*
SLOC *Size Lineof Code*
SRV *Service layer*
SubDES *Sub-Design phase*
SWPM *software process model*

U

UML *Unified Modeling Language*

V

V&V *Verification & Validation phase*

Table of contents

List of Tables:

List of Figures:

1 Introduction

1.1 Motivation

1.1.1 Automotive Industry

For the next years, experts estimate the global registration rate of new cars to be over 70.000.000 per year [1]. At the same time, the European Commission requests a reduction of traffic deaths by 50% in the same period. Up to 70% of future innovation in the automotive industry will be based on software and thus, software projects will gain importance. A side effect of the massive number of software modules in an automobile is, that in recent years 50% of all vehicle recalls were software related [2].

The increasing of software size and complexity does not only lead to improved new functionality but also to increase the defect rate. Defect rate is a measure of the relative number of units that are defective per release. This affects the quality and has a strong influence in the business. Especially for safety critical use cases, which is characteristic for the automotive sector, reliability has to be guaranteed by systematic defect avoidance already during development [1]. Managing and optimizing the quality assurance of large scale software product is one of the major challenge in the automotive industry. This will strongly influence the product costs for the development of a high quality product with low in-field defect rates and still low costs and planed deadlines. Automotive products have a long lifecycle compared to the development lifecycle, which explains the development efforts put into such failsafe products.

Applying changes may introduce defects into software products. There are several software releases with hundreds of changes to be addressed in the development. Changes have their unique characteristics based on process and product factors. Such metrics allow for an expert estimation of the probability of the injection of defects into the product. The time, in relation to the development phases, of the defect detection leads to a different impact on the product cost. Furthermore, an early detection of a defect accounts for dramatically lower costs.

Figure 1.1 and Figure 1.2 show the software changes of an industrial software product. They show that the development of the product has a long development lifecycle with a high number of change requests because of customer requests and defect corrections. The left Y-axis presents the state of actual number of changes within the time span. The change starts with the state “Submitted” as new change request, analysing the change is the second state “Analyzing” where the project manager have to decide accept the change or not, by acceptance an estimation and schedule of the change is necessary with the state “Decided”. After the decision and accept the change the working tasks have to start in the state “Managing Act” to implement the change and deliver it. The right Y-axis presents the total number of changes from the start of the development.

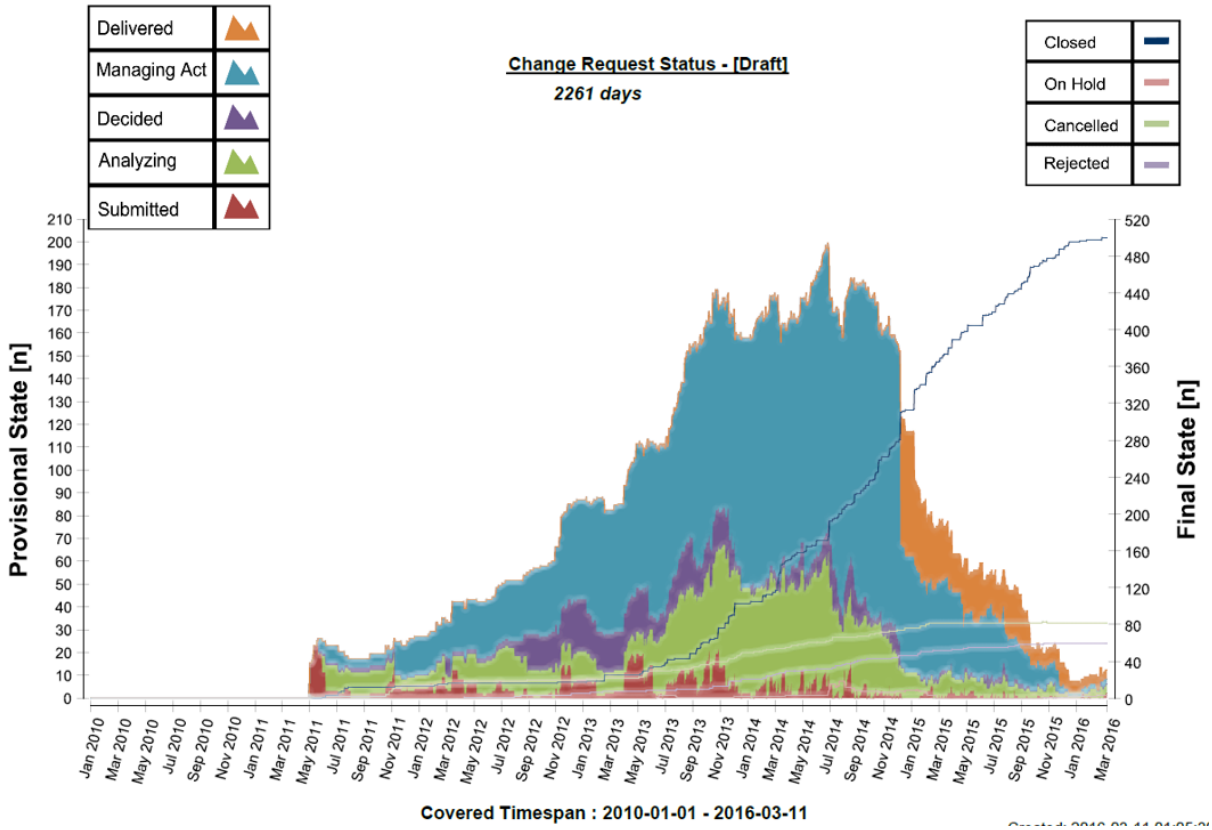


Figure 1.1 Change Request Distribution during Project Lifecycle

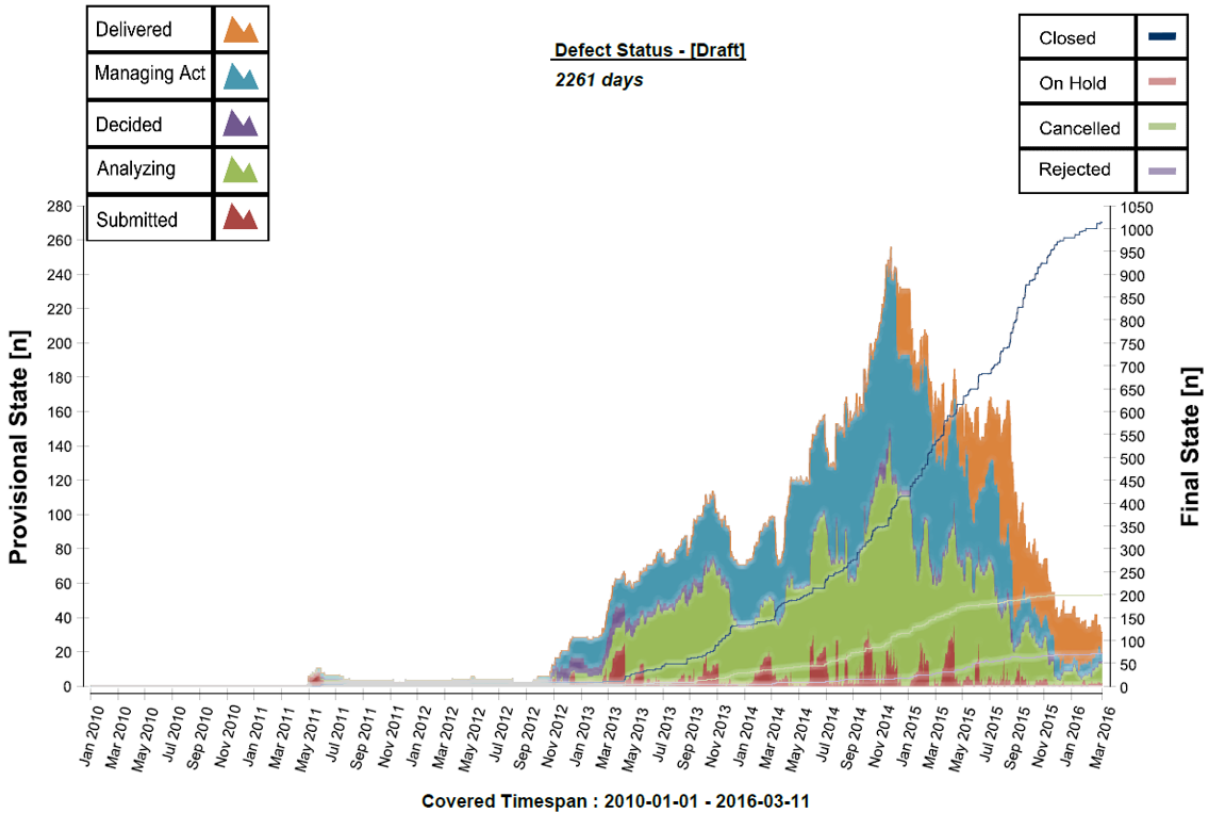


Figure 1.2 Change Request Distribution

1.1.2 Product Assessment

The product assessment consists of development, cost, sales and production assessment. Assessing the products defect rate is an important topic for developing safety critical products. Consider for example the automotive domain: power-train functionalities such as the combustion engine control and transmission management or electric motor control and battery management, comfort functions such as traffic sign recognition, lane keeping assistance, and collision avoidance systems are becoming more and more sophisticated and complex. A safety-critical system has to be dependable [3], i.e., it has "to avoid service failures that are more frequent and more severe than is acceptable" [4]. Note that in contrast to the real-time system definition, service failures here include incorrect results with a correct timing, i.e. produced before the deadline. Aside from the automotive domain, examples of safety-critical systems are nuclear reactors, medical devices, and vehicles.

One of the central topics when developing innovative safety critical products is the assessment of a product's defect rate before it is released. It is a big challenge and 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 have to be changed in the software?
- "How large are" (What is the complexity of) the changes to the software?
- How complex and difficult are these changes?
- What is the current test coverage and the future test coverage?

The defect rate has a central influence on the overall costs of the software product. Therefore, using defect rates improves the effort estimation. Especially the rework needed to fix defects is one of the uncertain variables. Several factors can influence the defect rate in software product, e.g. the pressure to meet a project's deadline. These factors have to be considered for estimating the project effort.

Due to the above described complexity it is difficult for project managers to keep track of these variables, factors and their impact on the product's defect rate. Thus, a deviation among the estimations of different project managers was found.

1.1.3 Process Improvement

The demand to develop software with lower defect rates leads to continuous process improvements with specific steps to identify weak areas where optimization is needed. Therefore, the efficiency of defect detection and correction activities are maximized.

For example, in order to reduce the overall defect rate there is a need to change the effort distribution of review and test activities. Before adapt these changes the development process in all process phases need to be analyzed concerning their effectiveness. E.g. the influence of every variable on the optimization criteria is of great interest. Based on this analysis, an evaluation of the improvement can be introduced. For project managers it is difficult to develop an overview of the impact of changing process variables in relation to the overall engineering process.

This thesis offers a model to support projects managers and decision-makers with their decisions regarding change activities and estimating the effort needed to make these changes. The complexity of the project management in the automotive domain together with the challenges have been briefly pointed out in the last paragraphs. A first analysis for this thesis was

based on extensive questionnaires to identify the influencing factors for the effort estimation of industrial software projects.

1.2 Research Hypothesis

The goal of this thesis is derived from the discussion of previous section. The project managers and decision makers in industrial domain face a big challenge in their daily work to estimate the effort of changes in software product. They are responsible for estimation and plan all activities for further non-functional tasks. They have to take in account the development process, quality assurance measures and effort reduction. These problems can be summarized to the following statement:

- P1 Missing an overview of all Key Performance Indicators (KPIs) of product quality and effort.
- P2 Inaccurate estimation of project effort because of different expert knowledge and process data.
- P3 Need for experts in different phases.

The influencing variables discussed in this thesis (KPIs) are essential to identify the optimization measures. The KPIs and the relationship among them influence the effort of development, and have a crucial effect on the defect rate which itself is an important factor to reduce the software development effort. Effort is used in this thesis to describe the cost of product.

Several extensive questionnaires were used systematically according to widely used estimation methods to identify the Influencing factors on the effort estimation in industrial software projects parallel to the expert's knowledge. These Influencing factors discusses the key questions and lessons learned from several interviews and questionnaires. These interviews were done with different groups of project managers in different software development domains. The respective project manager received the requirements as well as possible approaches to estimate the size and effort presented in detail. Finally, a free discussion took place at which the project managers could contribute their own ideas and suggestions for improvement. This information was shared with all project managers. Open questions were clarified after the interview took place. Fifty project managers took part in these questionnaires and interviews, and another twenty project managers and ten system architects took part in the subsequent analysis phase.

Most of ECU (Electronic Control Units) in the car are based on AUTOSAR[1], Bosch produces 72 different ECUs [2]. Bosch adapted AUTOSAR to an own CUBAS[2] architecture. For the requirements analysis of this thesis, 7 version update of CUBAS, including the initial AUTOSAR operating system for Linux v3.1, were analyzed. In addition, three variants of the engine control software for different car manufacturers, and four variants of car multimedia software were analyzed.

The degree of newly implemented functionality has been in the range of about 30% to 70%. The projects used different development processes like the V-Model and agile processes for

[1] AUTOSAR (Automotive Open Systems Architecture), https://www.autosar.org/, see section 5.1.1

[2] CUBAS (Common UBK Basic) is the complete AUTOSAR conforming Basic Software including the software drivers for controller peripherals, which are not specified in AUTOSAR, but excluding complex drivers and software drivers for ECU peripherals (ASICs). https://www.elektroniknet.de/elektronik-automotive/software-tools/autosar-basis-software-cubas-543.html

short development cycles with globally distributed development teams. The programming languages in use were C, C++, Java and script languages like Perl and Python. The projects had an average effort of 20 person years and an average duration of 6-9 months.

Priority (1=high ...8=low)	Criterion	Requirement
1	Process	The effort of development process management has to be included in the total development effort calculations.
2	Maintainability	The effort of defect correction and software maintenance has to be included in the total development effort calculations.
3	Flexibility	The estimation of efforts should be possible for different types of changes (in different development phases, small up to large changes).
4	Development	The development effort needed during product development lifecycle.
5	Quality Assurance	The effort of QA during development has to be included in the total development effort.
6	Test	The test execution effort (unit ~, component ~, integration ~, system ~, and delivery test)
7	Used Tool	The cost of tools used and the effort of using tools have to be included in the total development effort.
8	Simplicity of Use	Simplicity of using the method and the tool support this use

Table 1.1 Prioritized Criteria as Interview Result

The result of this initial analysis revealed the eight criteria summarized in Table 1.1 with decreasing priority. The requirements were identified as important and missing in the current project management activities and they are addressed in the new approach of this thesis. The main goals of this thesis can be formulated as follows:

- G1: The overall development effort shall be decreased by using a new model to identify the relevant quality assurance measures.
- G2: A new method is needed to identify hot spots in a software product (e. g., components or even single functions). These hot spots are most promising for a decrease of the defect rate and/or the development effort.
- G3: A development process has to be created to derive the KPIs for the reduction of the development effort as well as the effort of defect correction.

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

- H1: Current estimation models can be improved by the integration and feedback of process data. The estimation accuracy will increase.

- H2: With a new estimation model (see H1) project managers can identify the hot spots in a products architecture where the most effort reduction is possible.

- H3: A new development process with distributed efforts for quality assurance measures will reduce the effort of failure correction as well as the overall development effort.

1.3 Outline

Chapter 1 presents the problem definition of this thesis. The problem is identified as a result of analysing the development processes and the result obtained by several interviews and questionnaires for project managers and product architects in different automotive industries. Based on the result, the research hypotheses are constructed based on the defined problems motivated this thesis.

Chapter 2 provides information about this thesis' problem domain, describes software engineering in the fields of automotive, i.e. described the engineering process model as well as quality assurance topics. This is followed by an introduction of traceability and crosscutting approach. It describes the motivation behind selecting a crosscutting approach and introduces their main characteristics. The next section presents an overview on the estimation models and approaches used currently in the domain of software estimation. This chapter is closed with a state-of-the-art section, describing some activities tried to solve the problem statement of the thesis.

Chapter 3 describes the development methodology used and the model supported in this thesis. The development steps are described from the problem definition to the validation of the model in the procedure.

Chapter 4 describes the dataset used in this thesis. Three datasets were used to analyse the problem statement and how the KPIs affect the effort. These datasets were extracted from an internal project system, expert knowledge as well as different research and literature. The analysis of these datasets led to the development of two models: one for software processes (SWPM) and another for the flow of change effort (CEFMod).

Chapters 5 describes these two models and analyzes the KPIs needed for them. These models were built, calibrated and validated, with their results compared with estimates based on expert knowledge and the actual effort required to complete the change requests. Using the model shows an improvement in effort prediction.

Chapter 6 presents the result of practical tests for the models presented in this thesis. The thesis has been carried out in an industrial environment, more than one hundred change requirements have been analyzed. The effort for each requirement and some other similar requirements were estimated by using the approach presented in this thesis and estimated with another approach that is available in the company for this goal. These results have been analyzed and compared as a validation phase for the approach suggested by this thesis.

Chapter 7 presents a summary and the outlook for the work in this thesis. The quality levels for automotive product during the development lifecycle and the quality in use in customer side are viewed. A perspective for the use of this approach described here too.

2 Background

This chapter gives a background information about the main aspects of this thesis. Effort estimation is a process or an approximation of the probable effort of a product, program, or a project, computed on the basis of available information. It is very important to have an accurate effort estimation for every kind of project. Project estimation in an improper way; lead to estimate a high effort of the project, sometimes it will be reached 150-200% more than the real effort needed [5]. Therefore, there is a need to have an improved estimation of the project. The effort for a project is a function of many parameters. Software developers need a simple approach for effort estimation with improved accuracy, although in the early phases of software development projects the estimation has a high variance [6].

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 software products and therefore methods describing how to reduce software defects. This section is followed by the principles of Traceability analysing using crosscutting theory used to develop the models for this thesis. The third section of this chapter presents an overview about different models and approaches used for software estimation. Some were analyzed and tested to meet the goal of 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

Failsafe products are a major motivation in Industrial software teams. This section describes the field of automotive product engineering to illustrate the boundary conditions for the development of fail-safe automotive products and describes the 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 demand for software products in the automotive industry has been increasing exponentially over recent decades, making software engineering one of its critical success factors [7]. The extensive use of Electronic Control Units (ECU) in vehicles today has already led to an average of 20 ECUs in smaller and even more than 70 ECUs in upper class cars [1]. Develop a Complex features within a short time to market and with minimal engineering costs at very high quality are the challenges that have to be met. Besides functional complexity, some other specific characteristics for the automotive field are needed, especially:

- Code efficiency for an improved performance due to limitations of processor and memory resources.
- Code portability and traceability to ensure further development and variant handling.
- High availability and reliability to support safety-critical applications.
- Real-time operation.
- Network operation.

An increased number of software products in the automobile, leads to complex customer requirements, which make the automobile product a complex product.

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” [2].

Project and process management are needed with a continuous development in respect to the availability of the corresponding processes, which are increasingly being used as a basis for order placement [8]. The development process has to describe all phases of the development lifecycle and describe all interactions of different engineering disciplines in the system. The process and the continuous improvement of the process are important parts of the software development. Figure 2.1 illustrate most important phases of development process in an industrial software project including the activities needed to manage and control the project.

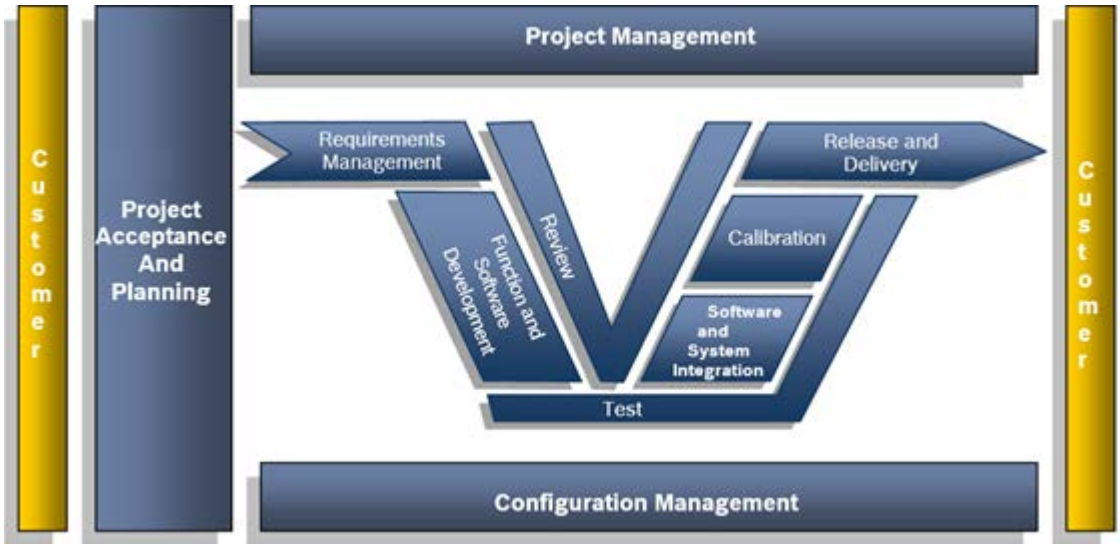


Figure 2.1 Software Development Process

2.1.1 Standards and Norms

Standards and norms help in improving products and processes and they 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 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 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 [9].

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 [9].

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 [9].

2.1.2 Quality Management

The quality is the degree to which a component, system or process meets specified requirements and customer needs and expectations. And software quality can be defined as: the totality of functionality and features of a software product that bear on its ability to satisfy stated or implied needs [10]. Different activities during the plan and development of the product help to achieve the desired quality. The quality in automotive is an important aspect for the software, whereby quality management is a support process for the software development and development lifecycle. The cost of product have to include the cost of quality. Achieving the development limited by time and budget is one of quality factor for industry. Standard and norms are used in development lifecycle to manage the quality. The standard ISO-26262 contains the guidelines and headlines for test procedures of the safety-critical application in the software, while some other norms serves to ensure the quality in other software areas. These standards and norms help to establish certain measures to fulfil the specific quality requirements. The quality assurance is a part of automotive processes, which includes all of these measures and leads to an increase in the quality of the software product [11]. The quality of the development process defines the software quality, whereby this assumption is the basis of the quality management. The individual’s capabilities and experiences in software development lead to better quality management. Different (e.g. FMEA, DRBFM) processes used in hardware quality management are specified as being used in the software sector to ensure the quality of the software product [12]. The requirement management also has a major effect on the quality assurance in the automotive sector. In order to ensure the high quality of software product, the definition

of standards and norms and the requirement management should be available as a parallel basis process.

The quality assurance deals with the quality of the project and product management. The project quality management addresses the management of the project and the product. While the project quality management is applied to all projects with no aspects specific to their product, product quality measures and techniques are specific to the particular type of product produced by the project. The project quality management takes care of the following issues [12]:

- Meeting customer requirements by overworking the project team may produce negative consequences in the form of increased employee attrition, unfounded errors, or rework
- Meeting project schedule objectives by rushing planned quality inspections may produce negative consequences when errors go undetected. Here a good traceability will aims to measures for tracing errors in the results which including the test and reviews.

The quality assurance of the product refers to the examination of the software products according to previously set quality features. Measures of quality assurance (QA) for software aim to examine and distinguish the following three categories:

- Organizational measures: measures for planning and examining.
- Constructive measures: measures for avoiding errors in software development, e.g. use of appropriate methods, training of employees to support the use of a process model.
- Analytic measures: measures for tracing errors in the results, e.g. reviews and software tests.

Software development in automotive domain is required to be of a high quality. Thus, there are a lot of activities dealing with the quality attributes of safety-critical software. Defect in an automotive system could lead to a damaged system; therefore, developers aim towards a defect-free software. Several test levels are needed to ensure the required software quality during the different phases of the development lifecycle. The test levels are classified into three levels as follows:

- Dynamic test: the product is tested at run-time 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, whereby the test cases should be previously defined. In order to ensure reliability and safety, other procedures are needed. It is similar to the situation of static and formal procedures.

2.1.3 Development Workflow

Nowadays, modern vehicles are known for their performance and quality, and they are valued for the money and innovative technology features that they include. Most of these current and future innovations are based on electronics and the use of software. The electronic control

units (ECUs) are becoming more dependent on the use of software. According to a study, the size of software in a modern automobile is between 40 to 90 Megabytes. Nonetheless, there is this ever-increasing demand on the software to guarantee the efficiency and responsiveness of the embedded functionality for the vehicle drivers, leading to the fact that 85% of the functionality of modern-day vehicles is controlled by software [13].

This raises a number of concerns related to hardware and software interactions, safety issues due to the systems malfunctioning and others. One of the major challenges is to manage the quality of this increasingly complex software on which vehicle functions are dependent. One approach to address these concerns is to use and reuse the standardized processes and best practices. The motor vehicle manufacturers are now focusing on the software capability of their suppliers and demand assessments of suppliers' software capability both before and during the projects.

Different OEMs (Original Equipment Manufacturers) have been using different approaches (as listed below) to assess the capability or maturity of their software suppliers [14]:

- Audi: both ISO/IEC TR 15504 and CMM
- BMW: Internal questionnaire
- DAG: ISO/IEC TR 15504 (since 1999)
- Porsche: Internal questionnaire
- VW: SW-CMM 1.1 and internal questionnaire

This created the need for developing a "common framework for assessing the suppliers in automotive industry" which resulted in the creation of Automotive SPICE (ASPICE).

"In a major initiative, the Automotive Special Interest Group (SIG), a joint special interest group of The SPICE User Group and The Procurement Forum, launched the Automotive SPICE® initiative together with the major motor vehicle manufacturers to develop a 'common framework for the assessment of suppliers in the Automotive Industry'. The result is the publication of the Automotive SPICE® Process Assessment Model (PAM) together with the associated Process Reference Model (PRM)" [15].

The goal was to create an international standard specific to automobile industry that could be used by the automobile manufacturers to assess their suppliers' software capability. Nothing new was created; instead, the existing SPICE standard (ISO/IEC 15504) was used and adapted to the automobile manufacturer requirements by tailoring it and selecting the most relevant subset of the processes from SPICE. It has published its own process reference model and process assessment model [8]. The members of the Automotive SIG include:

AUDI AG, BMW Group, Daimler AG, Fiat Auto S.p.A., Ford Werke GmbH, Jaguar, Land Rover, Dr. Ing. h.c. F. Porsche AG, Volkswagen AG and Volvo Car Corporation.

2.1.3.1 CMMI (Capability Maturity Model Integration)

CMMI is a process improvement approach for organizations that want to improve their process performance across a specific set of processes, a project, division or the entire organization [100]. It provides an organization with the fundamental and necessary elements of efficient processes that result in the improved performance of the organization's processes, as well as the improved performance of the organization eventually. These process improvement processes help in putting together traditionally separate organizational functions, setting goals

and priorities for improving those functions, suggesting quality improvement practices and providing a point of reference for appraising the current processes [16].

2.1.3.2 V-Model

V-model is a life cycle model influenced by Waterfall model in which processes follow a sequential path of execution. This means that each phase must be finished before the next phase starts. However, there exists a major difference between these two models that testing is much more focused throughout the development life cycle in V-model than in Waterfall model. The reason this model is called V-model is that testing plans and procedures are made for all the phases preceding and including implementation/coding phase. These testing plans and procedures are then executed after the coding is done to verify those phases which precede implementation phase [17]. Figure 2.2 below shows how these phases form a V shaped model.

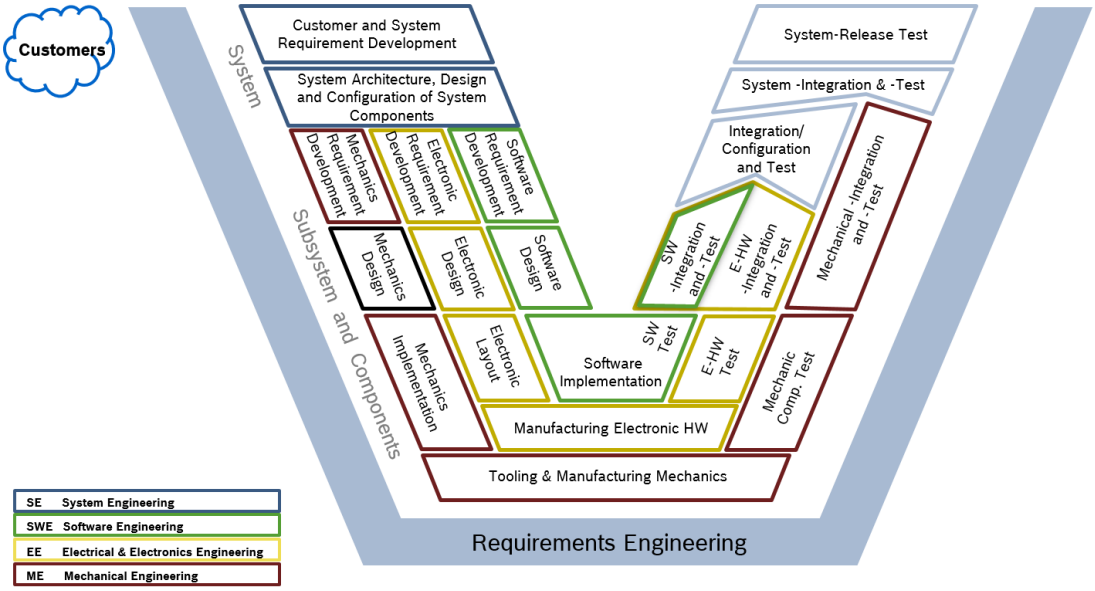


Figure 2.2 Product Development using V-model

2.2 Traceability & Crosscutting Concerns

With increased software complexity, software developers realized that the assessment of the impact of change in one requirement introduces in the rest of the system is a critical task [18]. The changes in a requirement may introduce a conflict with other requirements of a system. Undetected conflicts could lead to miss the project target, where the requirements of the software system might not reach its target and could not be released [19]. Therefore, it is necessary to detect the influence of requirements to each other and how they are implemented. Davis classifies traceability information in four types in his book “Software Requirements: Objects, Functions and States” [20]:

- Backward-from traceability links requirements to their sources in other documents or people.
- Forward-from traceability links requirements to the design and implementation components.
- Backward-to traceability links design and implementation components back to requirements.

- Forward-to traceability links other documents (which may have preceded the requirements document) to other relevant requirements.

The ability to define traceability between different requirements themselves has an important aspect in a complex software system. Kotonya and Sommerville [21] state that by extending the backward-from and forward-to traceability in order to have links between the same documents (the requirements document), it is possible to cover this concern.

To define traceability links, one of the most common methods used is by means of traceability tables. Table 2.1 shows a traceability table example for a system with six requirements. Each requirement is listed in the vertical and horizontal axes of the table, and the cells are used to mark the relationships between them. These tables can also be used to mark any other kind of traceability relationship. This would require adding the traceable artifact to both axes.

The proper way to read this table is by navigating its cells and interpreting the information depending on whether we are reading a row or a column. A mark in the row of a requirement indicates that this requirement is depending on the marked requirements. For instance, R3 in Table 2.1 is dependent on R4 and R5. On the other hand, by navigating the column we can find which requirements are depending on a specific requirement. We can also see that requirement R2 has R4 as dependent. Using this approach, it is possible to perform change impact analysis simply by navigating the column of the requirement that was changed. If a change was introduced to R5, by going down the R5 column, we could find that R2 and R3 are dependent requirements, and therefore the impact on R2 and R3 of the change introduced on R5 can be assessed.

	R1	R2	R3	R4	R5	R6
R1			*	*		
R2					*	*
R3				*	*	
R4		*				
R5						*
R6						

Table 2.1 Traceability Table Overview [21]

These tables however, are not scalable. For a small number of requirements, it is possible to navigate the table with relative ease to perform the necessary analysis, but for a large number of requirements (hundreds or thousands) the table becomes unmanageable. To address this problem, an alternative to traceability tables can be used. Instead of using a table, it is possible to use two traceability lists to provide the same information. One list traces the “depends-on” traceability, while the other provides the “dependents-off” traceability. Table 2.2 shows an example of these lists.

Traceability lists have the advantage of being more compact and easy to read than traceability tables. If one wishes to find the requirements that are depending on a given requirement it is a matter of searching the “dependents-off” list. For instance it is easy to see that R3 as R1 as dependent. The inverse is also straightforward. By navigating the “depends-on” list, we can see that R5 is dependent on R6. The only drawback of using traceability lists is that the information is duplicated, leading to problems of maintaining the information in a coherent state in both lists.

	Depends-on
R1	R3, R4
R2	R5, R6
R3	R4, R5
R4	R2
R5	R6

	Dependents-off
R2	R4
R3	R1
R4	R1, R3
R5	R2, R3
R6	R2, R5

Table 2.2 “Depends-on” and “Dependents-off” Traceability Lists [21]

Finally another important aspect of traceability is the ability to perform queries on traceability information and viewing the results returned. The great majority of the authors do not provide any implementation of the approaches proposed. This is usually left for the developer of the system to either implement the traceability himself, or to be performed by traceability tools [22]. Many different solutions can be used to achieve the same goal. Traceability tables can be implemented using a simple spreadsheet, or a relational database. Traceability lists can also be implemented using a relational database, a spreadsheet, a simple text file, or implemented in many programming languages (e.g., the standard Java Class Library [23] already contains a Hash table implementation that could be used to implement this solution).

Depending on the solution, the desired information can either be extracted manually (e.g., going through a spreadsheet) or automatically using predefined or custom queries (e.g., using SQL with a relational database). There are several kinds of traceability queries that can be performed. Traceability queries can occur at two different levels:

- Inter-level Queries traceability between artifacts at different development levels (e.g., requirement with design artifact or design artifact with source-code).

- Intra-level Queries traceability between artifacts at the same level of development (e.g., requirements models with requirements documents).

The type of queries can also be divided between simple and complex types. Simple queries usually involve choosing an artifact or set of artifacts and viewing the related artifacts. The type of information returned should also be chosen (which kind of artifacts, etc.). The results can also be viewed in a variety of formats. It is possible to see it represented in a simple textual report, a graph or a table. Figure 2.3 and Figure 2.4 show an overview of the concepts that address traceability.

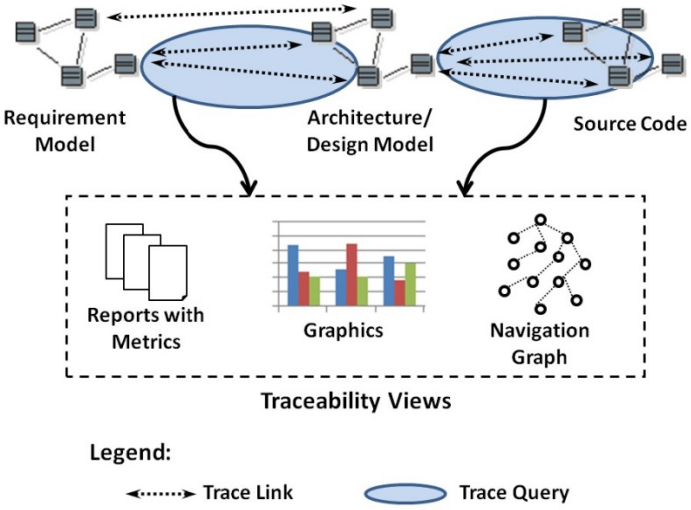


Figure 2.3 Traceability Basic Concepts [24]

More complex and sophisticated traceability queries can also be provided to the developers. These might include requirements coverage analysis, which aims to ensure that architecture, design and implementation artifacts cover specific requirements. This could be useful in satisfying the need of showing that the resulting system met the contractual agreements. Another important query might be the change impact analysis, mentioned previously, which allows the discovery of artifacts that are affected by a change introduced in a requirement.

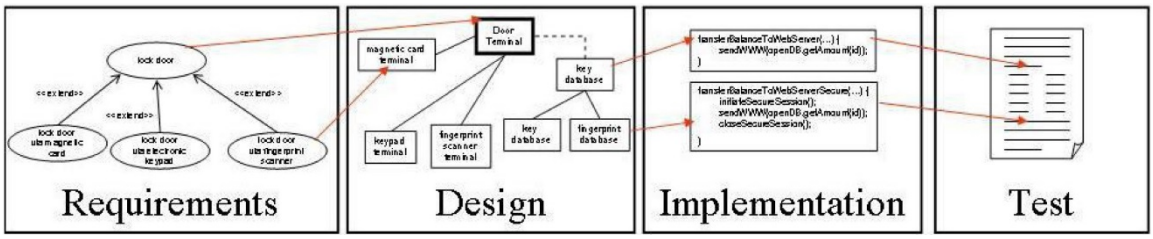


Figure 2.4 Fine-grained Traceability Approach [59]

2.2.1 Crosscutting Concerns Matrix Representation

The crosscutting can be represented in matrices, using the concept of a Design Structure Matrix (DSM) [23]. This matrix shows the relation between source and target. These relations can have different types, such as usage and abstraction dependencies (e.g. realization, refinement and tracing [21].

Different phases of development life cycle could be represent as a source and target system. Each phase will be a source for the following phase which represents the target. The source and target elements should be sort such that the sources are represented by the row of the matrix, and the target will be represented by the columns. Each cell of this matrix will be denoted with 1 if the source element (in the row) has a relation to the target element (in the column). It will be set to zero if there is no relation. Table 2.3 illustrates the dependency matrix of the system shown in Figure 2.5.

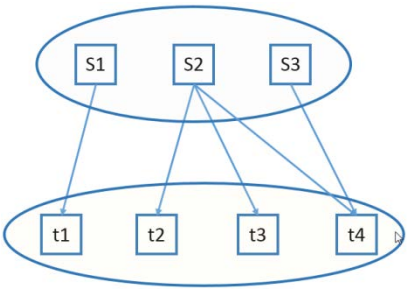


Figure 2.5 Mapping between Elements at Two Levels of Abstraction

Dependency matrix

	t1	t2	t3	t4
S1	1	0	0	0
S2	0	1	1	1
S3	0	0	0	1

Crosscutting matrix

	S1	S2	S3
S1	0	0	0
S2	0	0	1
S3	0	0	0

Table 2.3 Example Dependency Matrix and Crosscutting Matrix

To represent the development phases, a cascading pattern system will be used. For a system show in Figure 2.6 there are two patterns: between domains A and B, and between domains B and C.

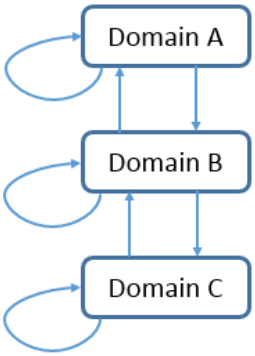


Figure 2.6 Cascading of Traceability Pattern

The target of the first pattern serves as source in the second pattern. Then, the transitivity dependency relation rel for source A, intermediate level B and target C, and #B is the number of elements in B is defined as follows [25]:

$$\exists\, k \,\varepsilon\, (1 \ldots \#B) : (A[i]\ rel\ B[k]) \wedge (B[k]\ rel\ C[m]) \Rightarrow (A[i]\ rel\ C[m])$$

The dependency between A and B is represented by the term of AxB, and between B and C with BxC. The trace dependencies between level A and C based on the transitivity of dependencies can be obtained by means of the Boolean matrix multiplication of the matrices AxB and BxC as shown in Table 2.4.

Domain	**A**	**B**	**C**
A	AxA	AxB Forward trace dependency	AxC ((AxB)(BxC))T Forward trace dependency
B	BxA Backward trace dependency	BxB	BxC Forward trace dependency
C	CxA ((AxB)(BxC))T Backward trace dependency	CxB Backward trace dependency	CxC

Table 2.4 Analysis Matrix for Different Domains

The transposed matrix (AxB)T illustrates the transposed dependencies between level B and A as a backward trace. In the coupling matrices with their relations lead to capture the dependencies of traceability of different elements in the same level. The transitive closure as example (AxA)* should be determined at each level. The dependencies between two levels could be found with the help of the transitive closure of the levels, as example between A and B levels:

(AxA)*(AxB)(BxB)*.

Change impact in the pattern of traceability operates in terms of the elements involved in the change of a source element. The set of these elements is called the impact set. To illustrate the different mapping between the components, Figure 2.7 illustrates a system with two levels of impact components.

Injection: impact s1 => change (s1,t1) Scattering: impact s2 => change ((s2,t2), (s2,t3)) Tangling: impact s3 => change (s3,t4) + preserve (s4,t4) Crosscutting: impact s5 => change ((s5,t5), (s5,t6)) + preserve ((s6,t6), (s7,t6)) [25]

The change required in a source like s1 means there is a change needed in t1, or there could be a change which would have to be analyzed. In the same way, a change in s2 means that there is a change in t2 and in t3. These changes needed an analysis to set the effect of s2. The change in s3 means that there is a change in t4 which need to be analyzed. The effect of this change in t4 need to be analyzed if there is an effect leads to other change needed in s4, because t4 dependent on s4 and this change need to be addressed as well. Changing in s5 for crosscutting leads to changes in t5 and t6, and there are preserving the dependencies of s6 and s7 onto t6.

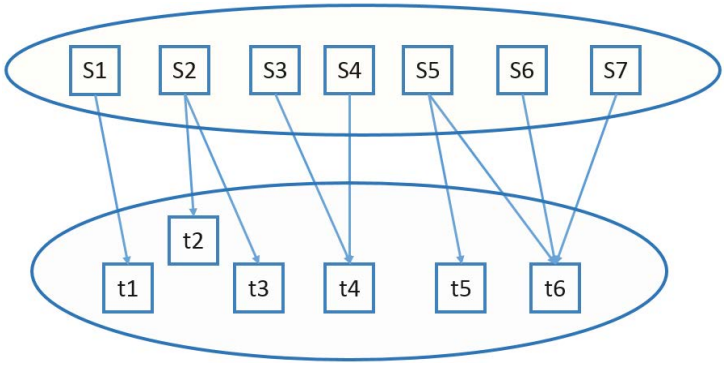


Figure 2.7 Example of Change Impact [26]

For a multiple levels dependencies, a change element v in the mapping (x,v) as shown below is used.

$$\text{impact}(x,v) = \text{change}(x,v) + (\text{preserve}(u,v) \mid u \leftarrow \text{preds}(v); x \neq v) + (\text{impact}(v,w) \mid w \leftarrow \text{succs}(v))$$

A software development life cycle is a multi layer model which represents the development life cycle phases. A change in a component y in a system mapping (x,y), may change or preserve other mappings. This can be represent by the function:

$$\text{impact}(x,y) = \text{change}(x,y) + (\text{preserve}(u,y) \mid u \leftarrow \text{preds}(y); x \neq y) + (\text{impact}(y,z) \mid z \leftarrow \text{succs}(y))$$

Where preds(v) represents the predecessors of an element in backward trace, and the function succs(v) represents the successors of an element in forward trace [27]. The result of this function contains a list of mappings to be changed based on forward traceability, and a list of mappings to be preserved based on backward traceability.

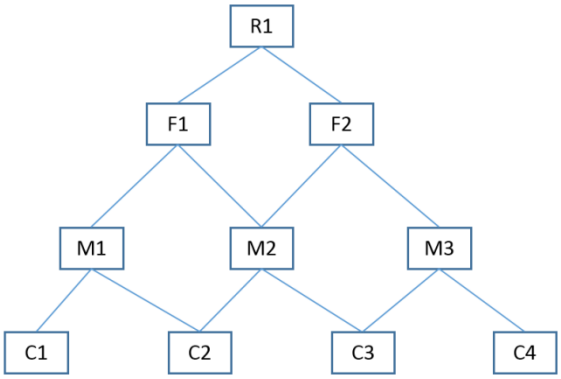


Figure 2.8 DSM Illustration of Software Development Phases

Figure 2.8 represents a part of a DSM for different levels of development life cycle. A defect in element M2 in the mapping (F2,M2) could be represented as:

$$\text{impact}(F2,M2) = \text{change}((F2,M2), (M2,C2), (M2,C3)) + \text{preserve}\ ((F1,M1), (M1,C2), (M3,C3))$$

The six mappings represented in the equation could involve changes in higher layers. In our example it goes up to R1. The analysis of impact (F1,M1), impact(M1,C2) and impact(M3,C3) will involve the total change effect in the model.

A change in the requirement level will include more impact changes.

impact(R1,R1) = change((R1,R1),(R1,F1),(F1,M1),(M1,C1),(M1,C2),(F1,M2),(M2,C2),

(M2,C3), (R1,F2), (F2,M2),(M2,C2),(M2,C3),(F2,M3), (M3,C3),(M3,C4))

+ preserve ((R1,F2),(F1,M2))

This example illustrates the possibility of changes. Such changes could include several levels up to the development level. Some changes will be difficult to analyze like (M1,C2) and (M2,C2) and will be presented as a conflict. Such a conflict only occurs in case of a combination of scattering at one level and tangling in a subsequent level, resulting in multiple paths from initial source to final target elements.

The conflicts could be calculated in a way:

#conflicts = #changed + #preserved - #involved

Table 2.5 illustrates the dependency matrix for the example of Figure 2.8. There are 4 levels R, F, M, and C, the cells value are shaded dependent on the relations among the system components.

	R	F1	F2	M1	M2	M3	C1	C2	C3	C4
R	1	1	1	1	2	1	1	4	4	1
F1	1	1		1	2		1	4	2	
F2	1		1		2	1		4	2	1
M1	1	1		1			1	2		
M2	2	2	2		1			2	2	
M3	1		1			1			2	1
C1	1	1		1			1			
C2	4	4	4	2	2			1		
C3	4	2	2		2	2			1	
C4	1		1			1				1

Table 2.5 Matrix Representation of Software System

The dependency matrix in Table 2.5 represents the dependency matrix of the system. This matrix represents the dependency between pairs of elements in the system. To estimate the effort need for the change, the size of a change should be estimated. The size of a change is the effect of the dependency to the system. A concern model was established by [28] [29]. It proposes a suite of aspect oriented metrics to complement the traditional metrics (coupling and cohesion) and is used to estimate a dependency value among the system elements, which helps to estimate the effort needed for each phase of development life cycle, and it also estimates the effort to shift changes from one phase to another. Two main metrics are defined:

- Degree of Scattering (DOS)

- Degree of Focus (DOF)

Concentration (CONC) is defined as a measure of the number of statements related to a concern within a specific component [29] [30]. Thus, CONC focuses on a particular source (s) and target element (t) and illustrates how a target element contributes to the realization of the source element:

$$CONC(s,t) = \frac{SLOCs\ in\ component\ t\ related\ to\ concern\ s}{SLOCs\ related\ to\ concern\ s}$$

To have a measure of the variance of the concentration of a concern over all components of the system. A second metric described the Degree of Scattering (DOS). A value of DOS close to 1 indicates that this concern(s) has a high degree of distributing while a value close to 0 indicates that this concern is not distributed and could be defined as

$$DOS(s) = 1 - \frac{|T| \sum_t^T \left(CONC(s,t) - \frac{1}{|T|}\right)^2}{|T|-1}$$

Where T is the number of components. A high DOS indicates the implementation of a concern is highly distributed (crosscutting). That's lead to have a risk of shifting some effort from one to other phases of the lifecycle.

This metrics are focused on the relation from source to target elements. To provide some metrics focused on the relation from target to source elements. A second definition of the Dedication metric which assesses the number of statements of a component (t) related to a particular concern (s) are needed.

$$DEDI(t,s) = \frac{SLOCs\ in\ component\ t\ related\ to\ concern\ s}{SLOCs\ related\ to\ concern\ t}$$

The metric of Degree of Constrain (DOF) and is a variance of the dedication of a component to every concern with respect to the worse case is illustrated:

$$DOF(t) = 1 - \frac{|S| \sum_s^S \left(CONC(t,s) - \frac{1}{|S|}\right)^2}{|T|-1}$$

Where S is the number of components.

With experts knowledge and the effect of KPIs discussed in this thesis, a dependency matrix from Table 2.5 was developed and presented in Table 2.6. The element value presents the dependency of the row on the column variable. Each cell represents the relationship between two components of the system i.e. r_1f_2 represents the dependency between requirement (R1) and feature (F1) and c_3c_4 represents the dependency between the artefact (c_3) and artefact (c_4) in the system shown in Figure 2.8.

	R	F1	F2	M1	M2	M3	C1	C2	C3	C4
R	r_1r_1	r_1f_1	r_1f_2	r_1m_1	r_1m_2	r_1m_3	r_1c_1	r_1c_2	r_1c_3	r_1c_4
F1	f_1r_1	f_1f_1	f_1f_2	f_1m_1	f_1m_2	f_1m_3	f_1c_1	f_1c_2	f_1c_3	f_1c_4
F2	f_2r_1	f_2f_1	f_2f_2	f_2m_1	f_2m_2	f_2m_3	f_2c_1	f_2c_2	f_2c_3	f_2c_4
M1	m_1r_1	m_1f_1	m_1f_2	m_1m_1	m_1m_2	m_1m_3	m_1c_1	m_1c_2	m_1c_3	m_1c_4
M2	m_2r_1	m_2f_1	m_2f_2	m_2m_1	m_2m_2	m_2m_3	m_2c_1	m_2c_2	m_2c_3	m_2c_4
M3	m_3r_1	m_2f_1	m_2f_2	m_2m_1	m_2m_2	m_2m_3	m_2c_1	m_2c_2	m_2c_3	m_2c_4
C1	c_1r_1	c_1f_1	c_1f_2	c_1m_1	c_1m_2	c_1m_3	c_1c_1	c_1c_2	c_1c_3	c_1c_4
C2	c_2r_1	c_2f_1	c_2f_2	c_2m_1	c_2m_2	c_2m_3	c_2c_1	c_2c_2	c_2c_3	c_2c_4
C3	c_3r_1	c_3f_1	c_3f_2	c_3m_1	c_3m_2	c_3m_3	c_3c_1	c_3c_2	c_3c_3	c_3c_4
C4	c_4r_1	c_4f_1	c_4f_2	c_4m_1	c_4m_2	c_4m_3	c_4c_1	c_4c_2	c_4c_3	c_4c_4

Table 2.6 Traceability Matrix with Change Effect

2.3 Software Estimation Technique

Complex software controlling is a very important part of software development. Project planning, control, risk analysis, and effort estimation all affect production costs and overall quality.

Much research has been put into software cost modelling. Since 1965, a number of studies and research have focused on the attributes of software projects. This led to the development of different models, starting in the late 1960s and early 1970s [31].

This section provides an overview of mainly used models and methods of software estimation. Most were developed in the early 1980s and 1990s, and are still being used and updated in response to software developments [5]. The widespread use of software has led to increase the complexity and production risks. Improved estimation techniques make use of software metrics to help project managers with their decisions. These estimation models help managers in different phases of the development lifecycle, starting with requirement analysis, architecture, implementation, testing, and maintenance.

Here, based on own experience, the commonly used estimation techniques. These techniques are presented. These techniques can be classified into:

- Model-based techniques
- Expertise-based techniques
- Learning-oriented techniques
- Dynamic techniques
- Regression-based techniques
- Composite techniques

2.3.1 Model Based Techniques

Many models were developed for software estimation, most of them were developed for specific use cases, and so their structures cannot be analyzed, compared, or contrasted in different developments domains. Other models are mostly used in software development management. These models can be analyzed in terms of software cost estimation and product quality. Some of these techniques do not cover early development activities and can only be used for quality management at the end of the development process; therefore, they cannot really support process quality management. This disadvantage make their use in process development within the automotive industry difficult. Some of the most commonly used model-based techniques are [32]:

- The Putnam Software Lifecycle Model (SLIM)
- The Jensen Model
- The Bailey-Basili Model
- PRICE-S
- Checkpoint
- Software Reliability Growth Models
- SOFTCOST
- SEER-SEM
- ESTIMACS
- COCOMO II

The most commonly used and evaluated techniques are the COCOMO II, SLIM, and SEER-SEM parametric models [31]. COCOMO II is a public domain model that has been implemented in several commercial tools and is continually updated. SLIM and SEER-SEM have been implemented in commercial tools and shared some features with COCOMO [31]. Other industrial effort models such as Checkpoint and Estimacs have been implemented but are not used for automotive software, instead focusing on business applications [33]. An earlier comparative survey of software cost models can be found in [11].

2.3.1.1 The Putnam Software Life-cycle Model (SLIM)

One of earliest software engineering techniques was developed in the late of 1970s by Larry Putnam of Quantitative Software Measurement, the Software Lifecycle Model (SLIM) [34].

SLIM assumes that effort for software projects is distributed similar to a collection of Rayleih curves. SLIM explained that staffing increased during the project and then decreased sharply during acceptance testing [34]. The effort required for the project is estimated using Rayleigh curves in terms of a manpower build up index (MBI) as described in Figure 2.9, productivity factor (PF), schedule and defect rate and various other productivity parameters this model used.

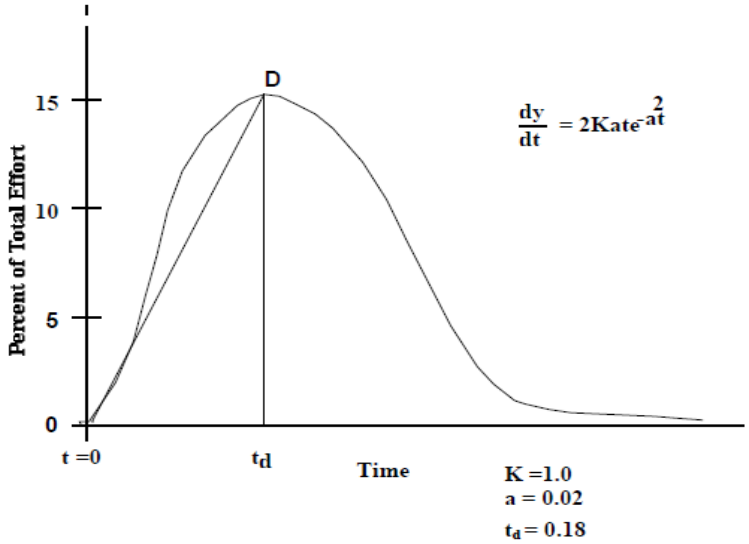


Figure 2.9 Rayleigh Model [34]

SLIM contains a software maintenance phase following the main development phase. The maintenance and development phases are very important and can be estimated independently during the development lifecycle. SLIM uses effective lines of code, SLOC, as a unit of size for software projects. The size of new and modified code can be found using SLIM, but not deleted code. SLIM deals with new and modified code in the same way as maintenance effort.

Analyzing two projects used Slim model shows some weak points in use for different types of project in automotive domain. Slim model works well on large project with no nodes to other project. Slim model is sensitive to size of code and technology factors. Slim model works for waterfall lifecycle, and need huge effort to adapt to agile or other life cycle process model.

2.3.1.2 COCOMO II

The COCOMO (COnstructive COst MOdel) cost and schedule estimation model was published in [35]. The definition of COCOMO II is published in [36]. COCOMO II employs three sub-models to deal with different software environments:

- Applications Composition,
- Early Design,
- Post Architecture.

These sub-models can be combined and used in different ways depending on the

- APPLICATION COMPOSITION MODEL: Used in the earliest development phases.
- EARLY DESIGN MODEL: Used in the next phases of the development lifecycle. It addresses the system architecture and operating concept. All information needs to be available in this phase – any missing information will make estimation difficult. This model is based on function points, lines of code, and some other factors and effort parameters.
- POST ARCHITECTURE MODEL: Used once the system architecture and project concept are available. The source code and function points are used in this model as sizing

parameters, with the ability to adjusting these parameters for future use. Different sets of effort multipliers and scale factors can be used to estimate development effort.

Cost factors are evaluated and weighted within COCOMO II based on application complexity and software reliability, execution, memory, and environmental constraints, development skill levels, tools and technologies, and a variety of other considerations [11].

The estimated effort required to develop a project is the most fundamental calculation made by COCOMO II. The effort (in person/months) is calculated using COCOMO II's effort equation. Most other COCOMO II results, including requirements and maintenance estimates, are derived from this quantity [11] [36].

One important factor for making good management decisions is the cost estimation for the project. This is connected to determining how much effort and time a software project requires [37].

- **COCOMO II Scale Drivers**

Scale drivers in the COCOMO II model are the most important factors contributing to project's duration and cost. Each scale driver need to be described and established in the project. These scale drivers determine the exponent used in the effort equation [11] [36]. COCOMO II defines each of the cost drivers, and the effort multiplier associated with each rating [38].

The scale drivers include:

- Precedents
- Development flexibility
- Architecture / Risk resolution
- Team cohesion
- Process maturity

- **COCOMO II Effort Equation**

The COCOMO II model makes its estimates of required effort based primarily on the estimate of the software project's size [31]:

$$Effort = 2.94 * EAF * (KSLOC)E \quad \text{equ. 2.6}$$

Where

EAF: Effort Adjustment Factor derived from the Cost Drivers.

E : exponent derived from the five Scale Drivers

2.3.1.3 Reuse Model

Devnani Chulani's [11] broad research into software engineering reuse costs indicates that the amount of modification required to reuse code to the resulting costs are nonlinear in three ways:

- Reuse effort does not start at zero.
- Analyzing the software to be able to modify

- The relative cost of analyzing module interfaces

Three modification factors can be derived from the nonlinearity of software cost estimation. These factors relate to the different development phases, the percentage level of design and implementation modification, and the percentage of integration effort required for software development. Detailed ratings of the reuse model could be found in [11].

2.3.2 Expertise Based Techniques

Estimation models based on expert knowledge are useful for projects lacking quantified empirical data [37]. An expert team will have the skills and understanding of project complexity to estimate the effort needed. They can estimate the effort needed for development and quality assurance activities needed for each development phase. Knowledge of previous projects and the differences between them could prove useful for the new project and improve estimates. The ongoing demand for expert knowledge is a major disadvantage of such techniques. Expert knowledge would be needed for different development phases to allow an estimate for the whole product. Some of most widely-used models using expertise-based techniques are the Delphi Technique, Rule-based Systems and the Work Breakdown Structure.

2.3.2.1 The Delphi Approach

The Delphi Technique is named after the legend of the Oracle of Delphi [39]. The technique draws on knowledge from senior expert developers to make predictions about software development projects.

Only a few expert developers are needed for this. The precise number of experts needed depends on their availability. A minimum of three experts is usually chosen to ensure a range of values. Farquhar performed an experiment where he gave four groups the same software specification and asked them to estimate the effort needed to develop the software product [13] [22]. The estimates were performed in two ways: two groups used the Delphi technique, while the other two had regular meetings. These meetings produced far better estimates than the Delphi technique [99]. These different methods of project prediction led to improvements in the Delphi technique. Boehm and Farquhar formulated an alternative method, the wideband Delphi technique [36], which involves different steps:

1. The coordinator provides the experts with the requirements for review
2. The coordinator sets up a group meeting to discuss related issues
3. The participants complete the forms separately and return them to the coordinator.
4. The coordinator feeds back the results of participants' responses.
5. The coordinator conducts another group meeting to discuss differences between the participants' responses to achieve a possible consensus
6. The coordinator asks the experts to refine their estimates, again separately, and steps 4-6 are repeated as many times as appropriate.

2.3.3 Learning Oriented Techniques

Learning-oriented techniques include both some of the "manual" as well as "automate" techniques applied to estimation activities by building models that "learn" from previous experience. The learning oriented techniques based on the knowledge from the prior and current software

development. There are several models which based on this technique, like neural networks and Case Based Reasoning (CBR) [40].

2.3.3.1 Neural Network

According to Gray and MacDonell [41], neural networks is the most common software estimation model-building technique used as an alternative to mean least squares regression. They realized the disadvantages of multiple regression methods and start with neural networks [42].

The Neural Network is a network that is comprised of nodes or neurons that imitate the biological neurons [14]. Neural networks map an input to an output using linear or non-linear functions based on the nature of the problem and can be used as classifiers or regression. The mainly used neural network models are used to solve a regression problem. Feed-forward networks are very popular in neural network where the flow of information is always from input to output. A neural network is usually composed of an input layer, hidden layers, and output layer. When a neural network model is used to predict the effort of a software project, the input layer is composed of nodes such as software size, programming language, programmer experience, etc. The middle layer contains a certain number of neurons, and each has a weight. The output layer contains one node that is the predicted effort. The output of a node is defined based on an activation function. This is represented mathematical algorithm. The model then iterates on its training algorithm, automatically adjusting the parameters of its estimation functions until the model estimate and the actual values are within some pre-specified delta. The specification of a delta value is important to avoid an over-trained to the known historical data. The model adjusting its estimation algorithms to get a good predicting results for the training data set [42] [43].

Neural networks have been extensively used to model software reliability. Cheng [44] and Mozolin [45] discuss the use of neural networks for reliability growth modelling and to identify high defect modules in software products. Koshgoftaar demonstrated a classification of fault fragile software modules [42].

Ch. Satyananda demonstrated a model designed to improve network performance based on the COCOMO model. The model uses a multi-layer feed forward neural network to estimate software development effort [42].

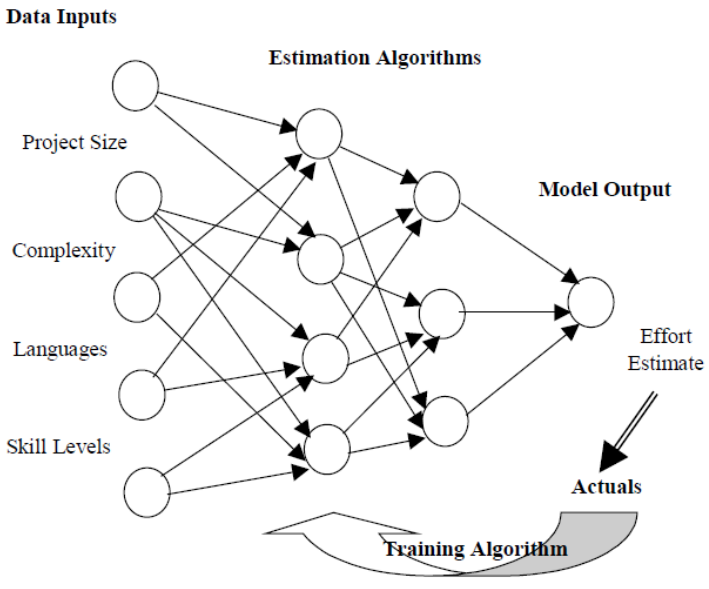


Figure 2.10 Neural Network Estimation Model

Analysing the Neural network model provide a good solution for industrial estimation, but the missing of information and tools that support the model make the use of this model not easy and not possible in a real project. Figure 2.10 shows a simple view for Neural Network.

2.3.3.2 Dynamic Based Techniques

Dynamic models estimate project activities and distribution of effort and schedule based on project characteristics. Price S [31] [32] and detailed COCOMO [6] estimate effort using phase-sensitive effort multipliers. The detailed COCOMO model provides different phases multipliers for every cost driver in the system to achieve this goal. This model is closer to the simpler intermediate COCOMO in terms of estimation accuracy, with better estimation of phase distribution effort. The most commonly used model based on this technique was created by Abdel-Hamid and Stuart Madnick [46]. They developed a model that estimates the time distribution of effort, schedule develop and defect rates as a function of staffing rates, experience mix, training rates, personnel turnover, defect introduction rates and other factors [32]. Madachy [15] developed a dynamic model of an inspection-based software lifecycle process to support quantitative evaluation of the process.

Dynamic system simulation models are represented mathematically by differential equations [15]:

$$x'(t) = f(x,p) \qquad \text{equ. 2.7}$$

Where x is vector describing the levels (states) in the model

p is set of model parameters f is a nonlinear vector function and t is the time.

2.3.3.3 Regression Based Techniques

Regression based techniques are popular technique of building models. These techniques include the Standard or Ordinary Least Squares regression and the robust regression approaches, etc. [47].

2.3.3.4 Composite Techniques

Each of models discussed above have a point of view to predict the effort of software which give some advantages for the model, but each of them still lack some data to have an accurate estimation. Composite techniques offer a model by combining two or more techniques to formulate the most appropriate functional form for estimation, in order to avoid missing data, which could be important for the estimation. The Bayesian approach is a composite techniques model combines two of important techniques: The expertise based and model based techniques. Dealing with the most important project data, make this technique suitable to be used in industrial field, and developing the safe critical software.

2.3.3.4.1 The Bayesian Approach

The Bayesian approach to calibrating the COCOMO II model was introduced by Chulani [40]. Chulani study produced a set of constants and cost drivers officially published in the COCOMO II book [6]. Since this study, the Bayesian approach has been used in a number of calibrations of the model using multiple COCOMO data sets [48]. An attractive estimating approach that has been used for the development of the COCOMO II model is Bayesian analysis [49].

Bayesian analysis is a well-defined and rigorous process of inductive reasoning that has been used in many scientific disciplines [48] [50]. The important advantage of using Bayesian approach in compare with other approach is that this model used the data set accumulated from

different completed projects and the experts' judgement from a developer team in order to estimate the effort of the project. The model used Baye's theorem to represent the data and the relationship among these data. This approach presents a formal process where a data sample could combined with the experts judgement. Using Bayes' theorem allowed to combine the two information sources as follows:

$$f(\beta \mid Y) = \frac{f(Y \mid \beta) f(\beta)}{f(Y)} \qquad \text{equ. 2.9}$$

Where ß is the vector of interested parameters, and Y is the vector of sample observations from the joint density function.

In equ. 2.9 is the posterior density function for ß summarizing all the information about ß, f(Y|ß) is the sample information and is algebraically equivalent to the likelihood function for ß, and f(ß) is the prior information summarizing the expert judgement information about ß [27]. The concept of the standard regression will be used here, and in addition the experts' knowledge will be included in the model. The different development phases will be deal with to reduce the risks during the development life cycle. This approach offers an ability to deals with data which missing in other estimations models. These data and the experts knowledge helps to get the decision in the different phases of development life cycle. Most of statistical estimation model describe the cost driver based on data. This data will be not enough in the complex system because of missing the interfaces among these data. Bayesian approach described the data and their relations, this information in the model leads to better estimation. The experts' knowledge will be involved in the model from the developing team members and the project managers. Information described the development process, the different parameters which affected the effort and parameters affected the quality of the software are important parameters to estimate a software. Bayesian techniques make best use of relevant prior information along with collected sample data in the decision making process to develop a stronger model [27].

2.4 Maintenance Cost Estimation Models

Increased software use has led to increased software complexity. This, in turn, means an increased probability of defects. The error ratio depends on the complexity of the product. Therefore, software maintenance estimation has become a crucial part of the software development lifecycle. Software domains which require a high quality product and low defect ratio need a process to handle changes and defects during the development lifecycle. Several models have been developed to estimate and handle software maintenance. The maintenance effort includes every software change necessitated by a change in requirements, defect or failure, or to fix various bugs. These models can be divided into three categories depending on their focus and estimation approach: phase level, release level, and task level maintenance estimation models.

2.4.1 Phase-Level Models

Product development are divided into different phases during the development lifecycle to ensure that quality requirements are satisfied. Software maintenance can be included in each phase to ensure software quality. Several estimation models have been developed to estimate the effort required for maintenance activities, which covers all activities performed during the operation of the software, including functional changes and improvements, defect correction and all other changes needed to ensure that all technical requirements of the system are met. The COCOMOM, SEER-SEM, PRICE-S, and SLIM models estimate this effort as a part of the total development cost of the software. These models use the number of lines of code as the main driver of estimated cost. Most of these models use additional cost drivers to estimate

maintenance effort. For example, COCOMO uses two drivers, software understanding and programmers' degree of unfamiliarity; SEER-SEM uses parameters such as maintenance size growth over time and maintenance rigor [51]. Any accurate estimation of system development should include the maintenance effort. The large number of assumptions required about the final system makes the accurately estimating maintenance effort one of major challenge for most industrial software.

2.4.2 Release Level Models

Some estimation models handle maintenance effort in a more finely grained way. They included the previous project data into estimates for the future releases.

Basili developed a regression model to estimate the effort needed for different types of maintenance, such as error correction and enhancement [18] by using a single variable, SLOC. Indicating the size of line code is a good predicator of the effort. Ramil [52] divide maintenance effort into sequences of operational releases after the initial delivery of the system, and developed a linear regression model to estimate the effort required to evolve the system between releases. They used size (measured as number of added or changed modules and subsystems) and number of changes to modules and in addition to all changed modules as model predictors. The model was not very accurate, indicating a model's evaluation and ensuing inferences are contingent upon the estimation accuracy indicators used [53]. The model was evaluated and recalibrated to be more accurate.

Caivano [54] described a model using engineering and restoration and developed a tool to dynamically calibrate the effort estimation model for renewal projects. The change information was accumulated during project execution, allowing for dynamic calibration. The model used number of lines of source code, complexity, and number of modules obtained after process update as parameter. The model analysis showed that fine-grained estimation and model dynamic recalibration improve accuracy and that estimation models are process dependent. A later empirical study verified these conclusions [55]. Other studies have also reported some success in improving prediction accuracies by recalibrating estimation models in an iterative development environment [56] [57].

Sneed developed a model called ManCost to estimate software maintenance effort by applying different sub-models to different types of maintenance tasks. Sneed classified maintenance into four different categories [58]:

- Error correction: effort of error correction for a release.
- Routine changes: maintenance costs of implementing routine change requests.
- Functional enhancement: effort of editing and improving software functionality. The SLOC or function point and associated effort is estimated using adjusted size, complexity, quality, and other influence factors considered here.
- Technical enhancement: effort of technical improvements, such as performance and optimization.

Sneed illustrates different characteristics of these types of tasks, each requiring an appropriate estimation sub-model. These sub-models must measure adjusted size and productivity indices, taking into account the effects of complexity and quality factors. Estimating maintenance effort lacks the validation reported for evaluating the estimation performance of the sub-models. Sneed proposed several extensions to account for reengineering tasks and maintaining web applications [59].

Functional Size Estimation (FSM) methods have inspired a number of studies to develop maintenance estimation models using FSM metrics. Most of these focus on the effort required for adaptive maintenance tasks by adding, modifying, and deleting functions. Focusing on function means FPA does not accurately reflect the size of small changes. To improve the model, Abran and Maya suggested an extension to the FPA method by using several complexity levels of the function to handle finer intervals [45].

Another study predicted the effort required to implement non-corrective change requests [60]. The FPA procedure to determine the FP count used in this model uses a multiplicative factor called the maintenance impact ratio (MIR) to adjust the FP count to account for the relative impact of a change. The result of the study shows that the size of the changed component has a higher impact on the effort than the size of the change itself. This is because those responsible for maintenance might have needed to spend investigating other functions tangentially related to the change [61].

2.4.3 Task Level Models

Some estimation models define maintenance tasks as defect reports or change requests as part of total maintenance. These models deal with small effort estimation in the range from a few hours to a month.

Sneed developed a model with a seven step process and developed a tool called SoftCalc to estimate the size and costs required to implement maintenance tasks [59]. The model used size measurements based on SLOC, function points, object points, and data points. Sneed determined the size of the affected domain and the affected software. These were then adjusted based on complexity, quality, and project influence factors. This adjusted size was used in combination with a productivity index to compute maintenance effort.

Fioravanti developed a model to build classification models for change request effort. The model supposed a procedure included four high level steps: identifying predictable metrics, identifying significant predictable metrics, generating a classification function, and validating the model [62].

The model represented the predictable variables and effort using a number of so-called difficulty indices. The model defined the effort range according to five intervals (less than an hour, between an hour and a day, between a day and a week, between a week and a month, and longer than a month) and indexed these from 1 to 5 respectively. Briand and Basili used a dataset of 163 change requests from different projects at the NASA Goddard Space Flight Center to create a model that could dynamically construct models according to specific environments [63].

Fioravanti presented and evaluated a model and metrics for estimating the adaptive maintenance effort of object-oriented systems [62]. Lucia proposed a model to build effort estimation models for corrective maintenance by using multiple linear regression [64]. Lucia showed that using a linear rather than nonlinear model produces more accurate estimates with the same variables. He also showed that using different types of corrective maintenance tasks can affect model performance.

2.5 State of the art

Several research projects relate to effort estimation for industrial software products. Moløkken Østvold and Jørgensen [65] reviewed the most relevant software effort estimation surveys. Expert judgment based estimation methods are the most used in the software industry. The reason for that is that the project teams believe in the team knowledge and they claimed that there is no evidence that formal estimation models lead to better estimates. The estimation accuracy when using formal methods did not show improvements over expert estimations [33]. They also found that managers tend to think that the accuracy of their estimates is better than it in fact is. Effort estimation accuracy in different development models has been analyzed. The results from those studies indicated that projects that are developed with an incremental model may be less likely to experience effort overruns than projects that implement sequential development models.

Yang [66] conducted a survey on software cost estimation in the Chinese software industry. The goal of the research was to investigate the state of software cost estimation in China, identify areas of potential improvement, and try to provide suggestions on how to improve the software cost estimation process and methods. Their study showed that large projects were more likely to experience cost and schedule overruns. When asked about satisfaction with the estimation techniques, most of the participants were neither satisfied nor dissatisfied. This study also showed that few organizations, a mere 15%, were using algorithmic model-based methods. The main causes for the low use of the previous techniques appeared to be the high adoption cost and the non-significant benefit once these are applied.

Trendowicz [67] performed surveys into the software industry with the purpose of analyzing industrial practices related to estimating software development effort. Multiple expert based techniques seemed to be the most accepted practice in the software industry. Collaboration with the customer generates better results in software projects, both in agile and traditional development environments [68]. Furulund [69] analyzes and discusses the effect customer collaboration has on estimation accuracy. It shows that the projects with daily contact between contractors and customers have lesser effort overruns than other projects do. Table 2.7 gives an overview on further related research activities in the software change estimation.

This thesis offers a model to support project managers in their decisions which are affected by many criteria. Several meetings have been held and questionnaires based on [70], have been used with project managers and estimation responsibles in Autosar groups in order to elaborate on and define these criteria which are presented in table 3. The questionnaire results show that several criteria affect the cost of software projects in automotive industry. In addition the criteria have been prioritized.

The mainly used estimation approaches were analyzed based on these criteria, in order to focus the strong and weak points in each approach depending on the industries requirement and needs. Comparative methods are based on comparison with similar projects already implemented in the same domain. The similarity is defined by previously determined equal influencing factors of the comparison projects, and the current project has to be estimated. Influencing factors can be both "results-related" and "processing-related". The most important result-related influencing factors are the quantity (scope of the project), the complexity (complexity of the project) and the quality (reliability, maintainability of the project result) of the project. The most significant development-related influencing factors include the knowledge and experience of project staff, the tools used, the modelling and programming languages used. [47] Another major development-related influencing factor is the use of personnel, whose marginal utility is falling, however. The coordination and communication effort increases disproportionately with the number of people [20].

Source	Main problem investigated
S.G. MacDonell, M.J. Shepperd (2003) [71]	Model based approaches in general needed too much inputs parameters to get a good estimation.
	The result is interpretable with some difficulty, Interpretability requires the knowledge which productivity factors have a significant impact and how they relate to effort.
X. Huang, D. Ho, J. Ren, L.F. Capretz (2007) [72]	High assumptions, an unrealistic assumptions could lead to high risk.
E. Stensrud, I. Myrtveit (1998) [73]	Late availability of estimates, i.e. the applicability of an estimation method during the various stages of software development.
M. Jørgensen (2004) [74]	Expertise approaches required experts in several domains.
	Low Repeatability, the experts needed for each new request.
E.S. Jun, J.K. Lee (2001) [75]	Low Transparency, i.e. experts for each domain needed.
S. Bibi, I. Stamelos (2006) [76]	Learning oriented based has a weak explanatory ability
I.Tronto, J. Silva, N.S. Anna (2008) [77]	The capability of dealing with noisy data shows that its prone to run into overfitting to the training data
A. Idri, A. Zahi, E. Mendes, A. Zakrani (2007) [78]	Require plentiful data for training Sensitive to network architecture and parameter setting
S. Berlin, T. Raz, C. Glezer, M. Zviran (2009) [79]	Requires project magnitude/size, project application area, the level of new design and code, experience and skill levels of project members, hardware constraints, customer specification, reliability requirements, development environment
L.C. Briand, T. Langley, I. Wieczorek (2000) [80]	usage in automation field is difficult
M.O. Elish (2009) [81]	The Regression tree requires building the tree only with information that can be estimated at the beginning of a project.
A. Heiat (2002) [82]	Using different error thresholds to build the tree lead to provide mixed results
Y.F. Li, M. Xie, T.N. Goh (2009) [83]	Low Interpretability need for big knowledge about all factors have a significant impact and how they relate to effort. Low Automation of Method and missing tools with high usability
E. Mendes, N. Mosley (2008) [84]	Bayesian inference statistics need for good probability knowledge
	Difficult to handle categorical features
I. Stamelos, L. Angelis (2008) [85]	Probability estimation needed, that leads to complex tool and use

Table 2.7 Summary of Recent Models of Software Change Estimation

Most estimation approaches deals with the size of software as an important criteria affecting the effort of the project. SLOC is used as measure for the complexity of change and all process needs. Figure 2.12 presents the estimation accuracy dependent on the SLOC for the model developed in this thesis. The 100% accuracy is the real effort known after project completion. The data was taken from the change request system. The expertise based estimation is a good approach but it requires costly experts in different phases of development lifecycle. The other approaches divert in their results depending on their use. This analysis lead to define the strong

and weak points for each estimation approach, and help to classify and evaluate different characteristics in the project reaching to describe the criteria of the project that have an effect on project effort and cost.

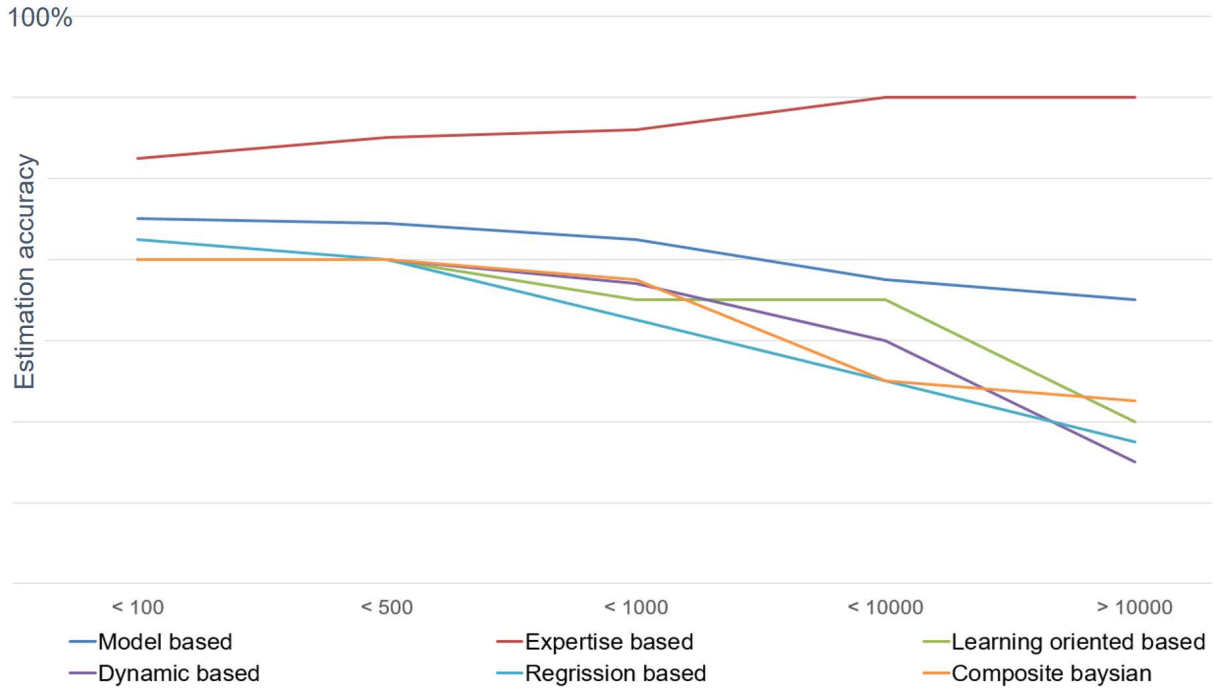


Figure 2.11 Estimation of Software Projects with Different Sizes

Figure 2.13 shows the missing criteria, as defined in Table 1.1, for the industries development, and the need for an approach that helps the project managers in their decision in one or more sector of estimation criteria.

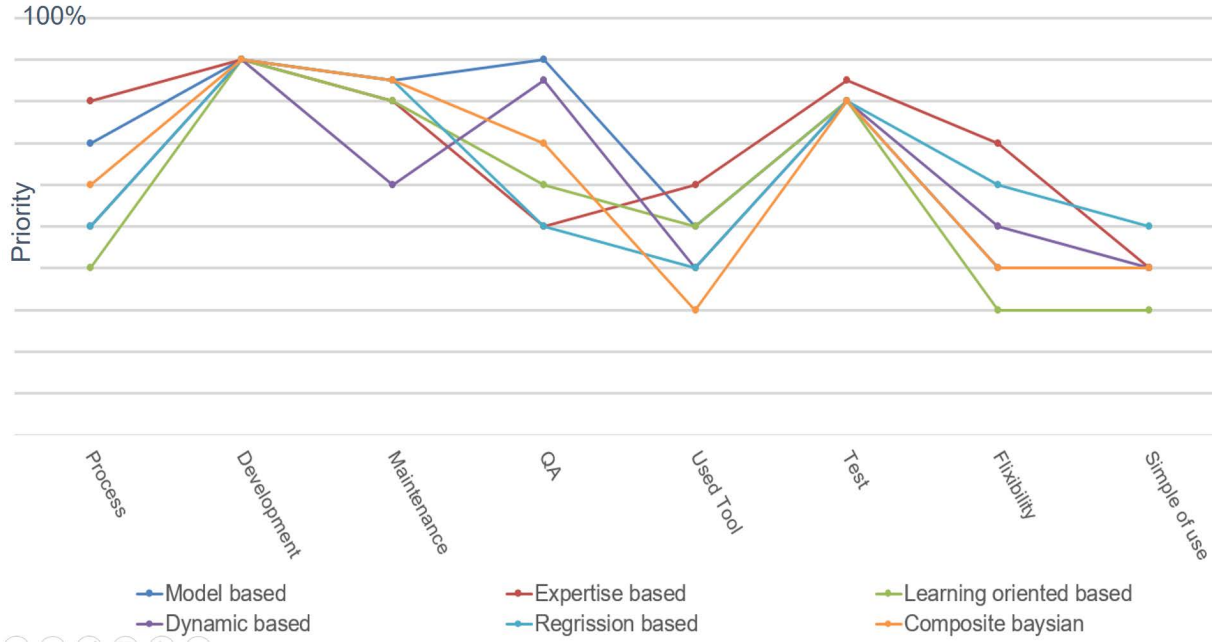


Figure 2.12 Estimation Techniques Test in Automotive Domain

Table 2.8 presents this need of industry for an approach satisfies all these criteria. It shows the main used approaches against the satisfaction criteria of industrial development demands. The classification used in five steps, with (++) as full satisfaction down to (- -) as totally non-satisfaction. This thesis offers an approach which satisfied all these criteria with (++).

Estimation Technique	Process	Development	Maintenance	QA	Used Tool	Test	Flixibility	Simple of use
Model based	**0**	++	++	+	**0**	++	-	-
Expertise based	+	+	+	+	--	-	+	+
Learning oriented based	--	++	+	+	-	+	**0**	-
Dynamic based	-	+	-	**0**	-	+	**0**	-
Regression based	-	+	-	+	-	+	+	**0**
Composite bayesian	-	+	+	**0**	+	+	+	-
Thesis approach	++	++	++	++	++	++	++	++

Table 2.8 Estimation Metrics in Automotive Domain

Table 2.9 summarizes the state-of-the-art knowledge of section 2 and points out the gaps to be closed by this thesis in relation to the research hypotheses and goals stated in section **Fehler! Verweisquelle konnte nicht gefunden werden.**.

Hypothesis (H), Goals / Requirement (G)	Refinement
G1: The overall development effort shall be decreased by using a new model to identify the relevant quality assurance measures.	Method/ approach have to enable the project managers to estimate the overall development effort and to identify the quality assurance measures where optimization leads to optimize the product costs. DSM, crosscutting and halstead approaches from the state-of-the-art are used, crosscutting and halstead have refined and adapted for this use. KPIs als factors for the method have to be defined and described.
G2: A new method is needed to identify hot spots in a software product (e. g., components or even single functions). These hot spots are most promising for a decrease of the defect rate and/ or the development effort.	Provide a method and tool to represent all product areas and identify the areas that need optimization to have optimal effort benefit ratio regarding to reduce the defect rate.
G3: A development process has to be created to derive the KPIs for the reduction of the development effort as well as the effort of defect correction.	Software development process model (SWPM) have to be developed in order to describe the KPIs needed for the effort of the development and defect correction.
H1: Current estimation models can be improved by the integration and feedback of process data. The estimation accuracy will increase.	The accuracy shall be improved by the new model and shall contain all development and process activities that affect the effort.
H2: With a new estimation model (see H1) project managers can identify the hot spots in a products architecture where the most effort reduction is possible.	The new model represents the product by using DSM and dependability matrix including all necessary artifacts in software process and development. That helps to identify the component in product where an improvement needed to deduce the effort.
H3: A new development process with distributed efforts for quality assurance measures will reduce the effort of defect correction as well as the overall development effort.	The model have to represent all software artifacts which affect the quality assurance and helps to reduce the defect correction effort and total development effort.

Table 2.9 State-of-the-Art Knowledge

3 New Estimation Approach

This section presents an approach for project managers and the process owner to support in their decisions of software effort estimation and project schedule. This approach is developed from the first step by defining the problem until the last step where the model valuated and tested. The developed approach as shown in Figure 3.1 offers a huge knowledge about the project structure and software process and architecture. The system model presents all artifacts of the project in a graphical view. The relationships among these artifacts are presented in a matrix which describes the dependency of each artifact on the anothers in the system. The dependability matrix focuses on the affected effort in each artifact related to its relationship with other artifacts in the system. The weighting vector is the effort of change in the own artifact.

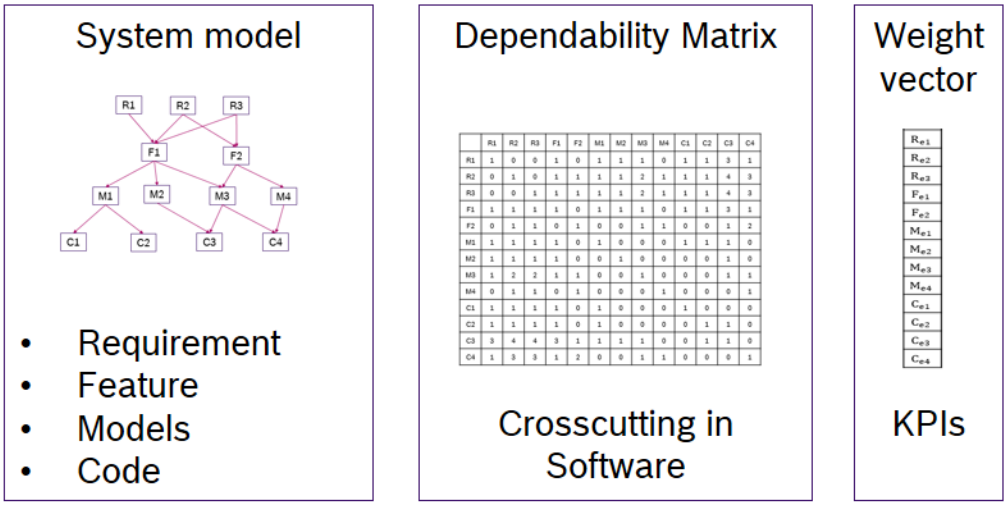


Figure 3.1 Own Approach

3.1 Approach description

This approach has to support project managers with their decision from multiple domains; therefore, a specification of multiple domain related and specification are needed. The approach has to trace all changes in the system throughout the project development lifecycle and document the reason for all changes in terms of feasibility, impact on other components, risk, effort, and schedule. The approach is classified into three knowledge sectors:

3.1.1 System model

The software has to be structured in a system model, where all artifacts in the software have to be presented in one or more graphical views depending on the domains focused in the system. All artifacts i.e. requirement, architecture, code, test and all non-functional artifacts (e.g. risk management and legacy standards and norms) are included and represents as nodes. The relationships among the nodes are presented as edges connected these nodes to each other.

The traceability and crosscutting concern were defined in section 2.2. A DSM have to represent the system elements from requirements down to implementation activities as shown in Figure 3.2. The DSM presents the relationships between these elements. Matrices can be constructed either “across rows” or “down columns,” depending on which direction is identified as forward process flow [86]. This overview presents a total view for thy system. This view helps the decision makers to have better understanding for the system and make their decision to estimate the effort needed for changing any element in the system. The model is built in a hierarchical design; therefore, changing a component in the system will affect other

components which are directly related, and other components which are not directly linked to the changed component may also be affected.

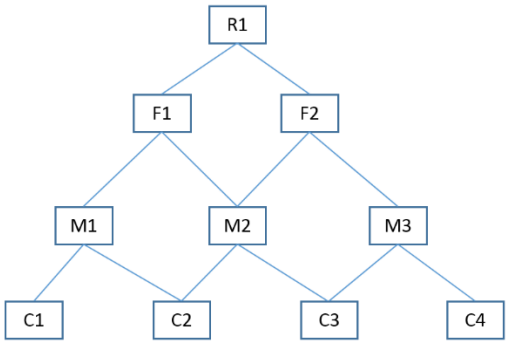


Figure 3.2 GSM Illustration of Software Development Phases

3.1.2 Dependability matrix

The crosscutting approach is used to describe the system model as a matrix. Each node has a value described the relationship between this node and other nodes in the system. This matrix described the dependability and how strong is the relationships among the system nodes. High node value means that the component in the row of the matrix is depended strongly on the component presented in the column of the matrix, where low value described a weak dependability of the component.

Change request in an element of the matrix could lead to high effort on other elements in the system. The node value is a description of the effeteness of a change effort in all affected elements, e.g. the node C_2R has a value four which presents a high node value and means that C_2 is strongly dependable on R, any change in R could lead to a change in C_2. In other side the value in node C_2M_3 is zero that means that C_2 is not dependent on F_3 change in F_3 zero effort for C_2.

The DSM will be constructed in an “across rows” fashion [67]. Non-zero entries indicate a relationship between two elements that the row element depends on the column element. The relationships among the different nodes in the system and the strength of this relation are analyzed using crosscutting principles, as illustrated in Table 3.1.

	R	**F1**	**F2**	**M1**	**M2**	**M3**	**C1**	**C2**	**C3**	**C4**
R	1	1	1	1	2	1	1	4	4	1
F1	1	1	0	1	2	0	1	4	2	0
F2	1	0	1	0	2	1	0	4	2	1
M1	1	1	0	1	0	0	1	2	0	0
M2	2	2	2	0	1	0	0	2	2	0
M3	1	0	1	0	0	1	0	0	2	1
C1	1	1	0	1	0	0	1	0	0	0
C2	4	4	4	2	2	0	0	1	0	0
C3	4	2	2	0	2	2	0	0	1	0
C4	1	0	1	0	0	1	0	0	0	1

Table 3.1 Traceability Matrix for System Described in Figure 6.2

3.1.3 Weight vector

The weighting value is a measure for the effort needed for the change in an element of the system as element own effort. This weighting value is depending on the complexity of the element. To find this complexity, an adapted Halstead method is used. The software development and defect correction models support the estimation approach by parameters that affected directly the effort of change. This effort is depending on the process and quality assurance used throughout the development lifecycle.

3.1.3.1 Complexity factor of change

Effort of changes dependent on the complexity of the system and the change needed. In order to estimate the effort of change needed in a system, the complexity of change included in a system have to be estimated. Halstead model [87] [88] is used in this thesis to estimate the complexity. Halstead model is in use since years to estimate the complexity of software code [88]. Halstead´s metrics is based on interpreting the source code as a sequence of tokens and classifying each token to be an operator or an operand.

The effort to implement (*E*) or understand a program is proportional to the volume (*V*) and to the difficulty (*D*) level of the program is presents in *equ.* 3.1

$E = V * D$ *equ.* 3.1

Equ. 3.1 could be used to estimate the effort for development of system contains several models. The effort of change in a module is estimated as in *equ.* 3.2:

Effort = Complexity x Weight [hour] *equ.* 3.2

In order to use Halstead model in estimation of software effort in a project, the metrics could be obtained as *equ.* 3.3

$Complexity = (N_1+N_2) \cdot log2\ (N_3+N_4)$ *equ.* 3.3

Where:

N_1: Number of component

N_2: Number of input for Module

N_3: Number of different component types

N_4: Number of output for Modules

Change request and defect in a software depending on the system complexity, the number of modules in the system and the interfaces among the modules in system to be changed. Therefore, a change in module leads to different changes in other module because of their dependability on each other and the interfaces among the module. The DSM matrix present the complexity and the connectivity among the different modules in system. The complexity of each module have to be estimated with Halstead model as in *equ.* 3.3.

The effort of change in a module is estimated as:

$Effort = \sum Complexity \times Weight\ [hour]$ *equ. 3.4*

3.1.3.2 Weighing of change

Industrial software development have to ensure the quality and more different metrics in software before release the software and the product. Several processes have to be used to cover all-important properties of the product. The development could be done in different phases by

different teams. Adapt new change request in a software system could have huge effort more than the effort need for code implementation. Estimate the effort needed for adapt a change or fix a defect in software system have to focus on all activities described in the process. Chapter 5 presents and analyses the development process in automotive industrial domain, and presents a model to estimate the weight of change considering all activities needed as standard process. This model supports the Project manager to estimate the effort of change in a complex software. The effort of change in a module is estimated as in *equ.* 3.4.

This thesis used some standard methods e.g. crosscutting concern and halstead, to build the estimation system, they were be adapted to this use. A new KPIs needed for the system were be analyzed and described as a new technique in the thesis. Figure 3.3 described the steps to estimate the effort of changes in software project, the own technique are bordered with a red square.

3.2 Development approach in this thesis

Procedure model

The guidelines for designing expert systems [89] is followed in this thesis as a development procedure. The goal of the thesis is to design an approach whose main application is to support project managers and process owners in making decisions related to the effort and cost estimation of software change. Figure 3.3 shows the procedure used in this thesis to build the model.

The definition of the problem is the first step toward having the model. Analyzing the problem helps to have the key performance indicators (KPIs) which need to define the problem. The problem definition of the models described in this thesis are derived from the research goal defined earlier in section 1.2. The Goal Question Metric (GQM) approach [90] and crosscutting concern described in section 2.2 have been used to systematically identify relevant KPIs of the model and their relations. According to that 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 structure of the model.

Internal sources within the specific automotive engineering environment has been used to obtain the model data. The main data source used is the change and defect management system which used to trace all changes and their specific information. A change request have to summarize all developed artifacts needed to realize this request of a feature. Process documentation and expert knowledge have been used as a further internal sources. Because of confidentiality of data used for build the model, the data has been made anonymous.

The Model is created and simulated with expertise in multiple fields, in order to establish a deep understanding of the model's domain, as first. Second, the discussed problem has analyzed and understood including its Key Performance Indicators (KPIs) and the relationships among these KPIs. A deep automotive expertise is demanded here, to be able to map the problem definition from theory to practice.

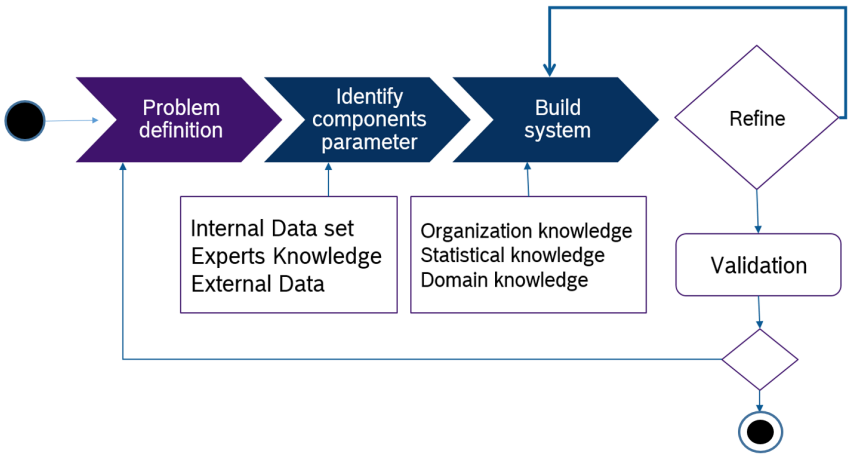


Figure 3.3 Model Development Procedure

To generate the KPIs and the relationship among them, a huge dataset is needed. These data are mostly gathered from:

- Internal sources: In the automotive industry, the data management system, including the change and defect management system, stores all change-specific information. Different change management system for three different automotive projects were analyzed in order to get the specific KPIs used in real life project.
- Expert knowledge: project managers and developers are the best experts working on the project. Expert knowledge is accumulated and collated via multiple discussions and discussion matrices filled out by the experts to generate a dataset of expert knowledge which can be used in the model.
- External sources: several studies and publications provide theoretical knowledge about the system.

Finally, to understand the consequences of data combination and the effect of data on the results, a statistical knowledge is required. Multiple iterations were required to design the models and fulfil all requirements from the fields mentioned above. And multiple scenarios were defined and analyzed to reflect different aspects of the problem definition. A model validation has been done as a last step.

3.2.1 Build and simulate the approach

Safety and quality are important requirements for software development in the automotive domain. To ensure these requirements are met, different norms and standards must be satisfied during the development lifecycle. The goal of this thesis is to build a system to support and help the project managers in their decisions. To achieve this, a large amount of data about the software changes are needed, these were be obtained by analysing different database of software changes systems and several experts in their specific areas. The system should be able to support the decision from multiple domains; therefore, a specification of multiple domain-related are needed. To exchange information among these domains, a domain specification is needed. The system should be able to trace every change in the system throughout the project development lifecycle and document the reason for every changes in terms of feasibility, impact on other components, risk, effort, and schedule. The system should be testable. A verification of the system was described and taken into account during its design. To test the system, all KPIs and their relationships should be verifiable, meaningful, and measurable.

Automotive software engineering is complex. Any model requires the following features:

- The ability to combine different data to reach a single decision.
- The ability to describe the interfaces among the system components.
- The ability to estimate incomplete data.

These requirements are not typically supported by traditional modelling techniques.

3.2.2 Model validation

The evaluation criteria in the thesis are based on existing studies [91] [92] and the available experience in the company with resource estimation methods. A general definitions has helped to structure the team discussion and evaluation of existing resource estimation models and methods. Table 3.2 gives an overview of three categories of evaluation criteria [70]:

- The first category is related to criteria for assessing models and their estimates (Model and Estimate Criteria).
- The second category of criteria deals with the process to build the final estimate (Estimation Method Criteria).
- The third group considers the applications of an estimation method (Application Criteria) to obtain a final resource expenditure estimate.

Model and Estimate Criteria	**Estimation Method Criteria**	**Application Criteria**
Quality of model and estimate	Assumptions	Application Coverage
Inputs required	Repeatability	Generalizability
Completeness	Complexity	Comprehensiveness
Type of estimates	Automation (Modeling)	Availability of estimates
Calibration	Transparency	Automation (Method Usage)
Interpretability		

Table 3.2 Evaluation Criteria for Cost Estimation Methods

- **Model and Estimate Criteria:** The category consists of criteria that are important when trying to assess an estimation model or technique, and its estimates. The main related questions are the following:
  - How good is the model's predictive accuracy?
  - What are the required inputs?
  - What kinds of resource expenditures are estimated?
  - What kinds of estimates are produced?
  - Can the model be calibrated?
  - Is the model interpretable by software practitioners?
- **Estimation Method Criteria:** This category deals with the estimation method as a whole, producing a final estimate, possibly based on a combination of estimation models and techniques. The main questions covered by this category are the following.

- What are the main underlying assumptions made by the underlying estimation model(s) and how realistic are they?
- How repeatable is the process to derive an estimate?
- How complex is the method to apply?
- To which extent is the model development supported by a tool?
- How transparent is the estimation method?

- **Application Criteria:** This category focuses on the applicability of a model with the help of a model application method. The main related questions are the following:
  - For which purposes can an estimation model be applied?
  - In which context can the estimation method be applied and was applied to date?
  - For what purposes can the model be applied?
  - How early in the lifecycle can a model be applied?

These validation techniques were considered to test the prototype to cover different aspects of validity. Different finding which do not affect the system specification were fixed directly. Other finding were covered as High Level Validity, Subsystem Validity and Input-Output Comparison described in the following subsections. With this validation we ensure that the structure of the model allows to support the decisions which have been specified during Design of the Expert System in order to ensure that the model supports an expert as he expects it to be. The internal validity of the model focused in the next step. The Model is divided into sub-models. Each sub-model is defined as a function of its input, which is the output of other sub-model. It is defined how a specific group should respond as part of the system.

Finally, the model is tested as a system. Comparable to the subsystem validity, the overall decision support system can be seen as an output based on a function of its input. For different input and output data, the system's decision and the expert's response to the input data will be compared. This validation will employ different technical knowledge [93]. A large amount of input data will be used to ensure the goodness of fit of the system by comparing the output with available expert data.

3.2.3 Refinement and generalization

After building the prototype, it was necessary to refine the data. Some features used in particular subsystems proved helpful in other parts of the system. Generalization and characterization techniques were used here. In this phase we found it necessary to refine and calibrate some parts of the system and fix various defects. After the refining phase, the model was rebuilt. Some refining was conducted following expert and user feedback. Major corrections and enhancements were integrated during this phase. Several iterations were run to build the expert system.

4 Data Model Analysis of Real World Automotive Software Projects

The model for software cost estimation as presented in the last section needs to be qualified and parameterized. The data base for the qualification of the model has been collected from different resources. Therefore, the data presented here is focused on industrial projects with their defect and failure rates. The analysis of this data is considered depending on the development process and development phases. The analysis of data leads to the classification of the features of the developed software and the failure effort factor. The following three paragraphs briefly describe the data that was collected and analyzed.

- **Internal data**

One important parameter of automotive products is the quality. The automotive software is developed with different processes to obtain the desired quality. To guarantee the required quality level, there is a demand for a process of the development lifecycle dealing with change requests.

Change requests are captured and managed with software change systems. Here, three databases of software change systems were analyzed with different projects. This database contains all information data about the changes in the software and the phase of development where these defects were generated. They describe the development chronology of each requirement in different phases of the development lifecycle and contains information about every change in this requirement depending on a defect or customer change request. This database contains full information about each software part from start with the requirement management to deliver/ realize this requirement including all test levels and all other formal and non-formal documentation required in the development and release processes of the software to customers. The above mentioned data has been anonymized for reasons of confidentiality and will be referred to as “internal data”.

- **Expert Knowledge Data**

A lot of important data is knowledge experience of the software developers and the project managers in each company. Such data could not be found as an internal database in a change request. The difficulty of rework and how it is estimated is one kind of data that cannot be found in a change request process. Therefore, this data has been accumulated from the project managers and stakeholders as expert knowledge data. This information was collected and will be analyzed in this section accordingly.

- **Experiments and External Data**

Different research studies and experiments were analyzed to identify the key performance indicators (KPIs) concerning the effort of software development and their effect on the changes. Some experiments were conducted to analyze and classify the KPIs used to build the system. A lot of other data was collected by a literature analysis and some closed research studies.

4.1 Data Analysis Model

Automotive software development projects use change request processes. As result of such a process many data items are stored. This “internal data” describes the data types and the

relation among this data throughout the system. Collecting and analyzing this data leads to illustrating different characteristics of the software process. The four most important data item are as follows:

- Feature change rate
- Change category
- Change effort for each task
- Type and priority of defect and defect correction to be described as defect analysing

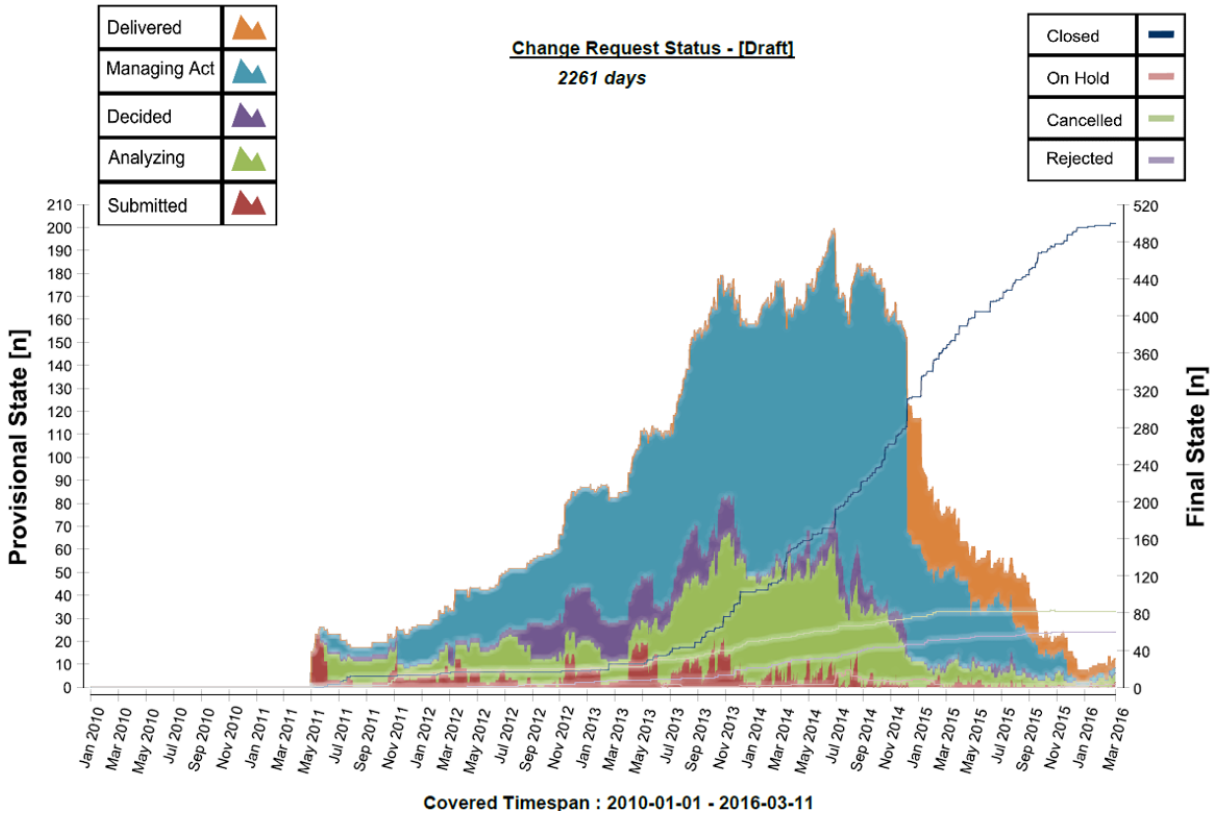


Figure 4.1 Change Request Status

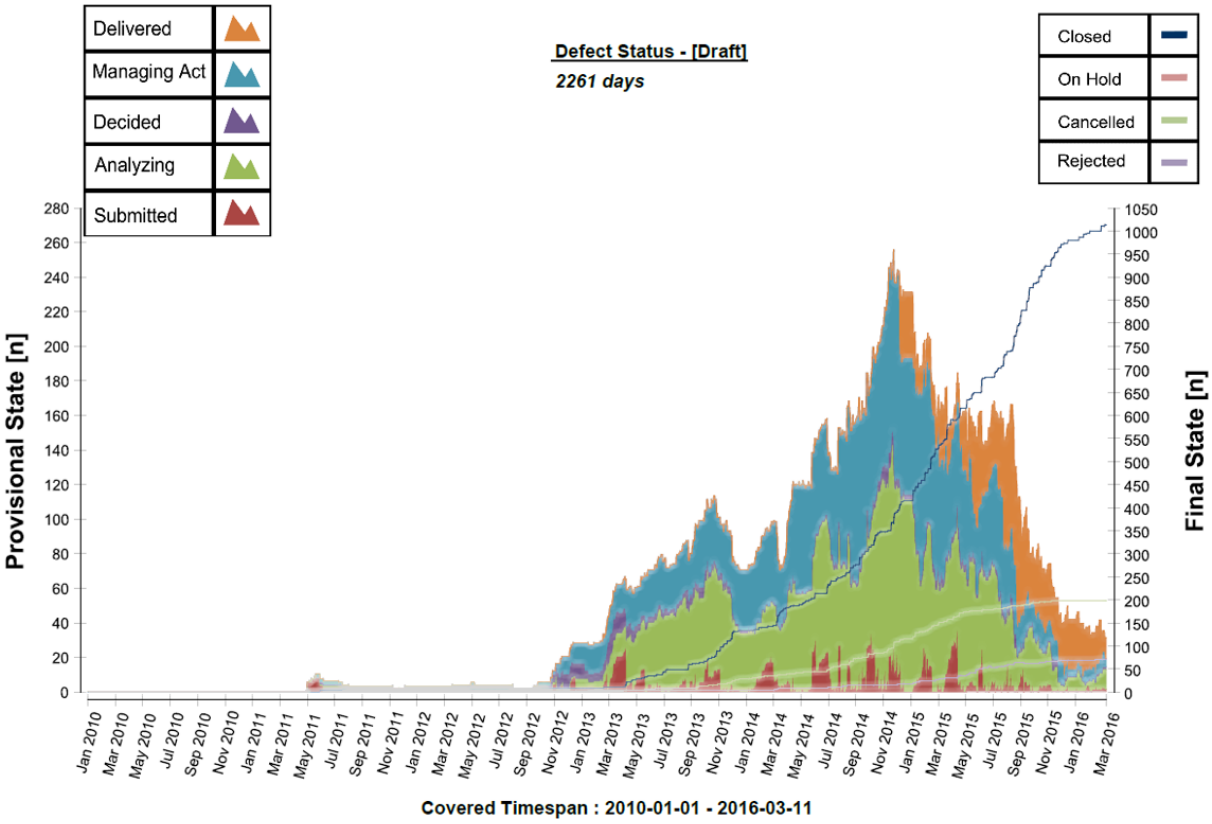


Figure 4.2 Defect Status

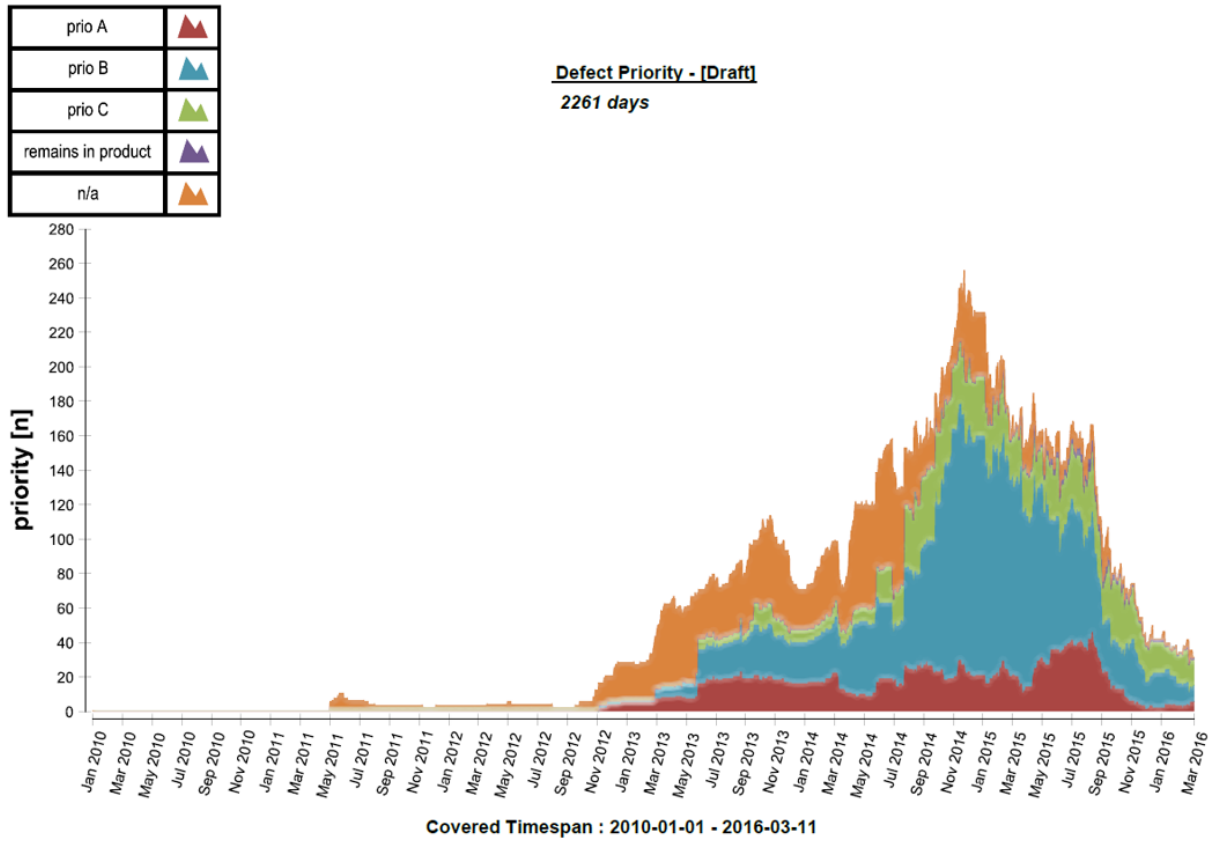


Figure 4.3 Priority Status

To ensure the quality of the product (e.g. automobile component), a large number of management systems and tools are used. The projects analyzed here were developed with different development process lifecycles. Agile development was used in some of these projects and traditional waterfall in others. The requirement management system (RM) was used for requirements representation and their status. The change request management system (CQM) was used for change management in system. The CRM is a system to register every change request (CRQ), bug fixing and defect change in the system. The CRM system contains all information about the development and the changes of each feature in the software project. CRM contains a lot of information about the defects in software, where the defect is originated and where it was detected, the deadline for fixing it and the effort needed to correct the defect.

It is important to support full traceability in the forward and backward direction for the development system from the requirement analysis phase to the validation phase, including all needed documentation. The risk management and quality are very important metrics in the development lifecycle and should be included in the traceability matrix of the data. CRM data can be classified into three categories depending on their required change management (CRM):

- Change request status as shown in Figure 4.1: describes the development states of change in requirement, as adding a new requirement, modifying or deleting an existing requirement. This information is needed for getting the development effort of the changes.

- Defect status as shown in Figure 4.2: describes the Failures or bugs in the Feature. These data contains important information about the failure effort, the phase where the failure detected, and the phase where the defect would be corrected (phase of correction).

- Change request priority as shown in Figure 4.3: describes the priority of each request and set a measure for the project deadline.

- Defect correction: describes the change of software to correct the defect. This contains the effort of correction.

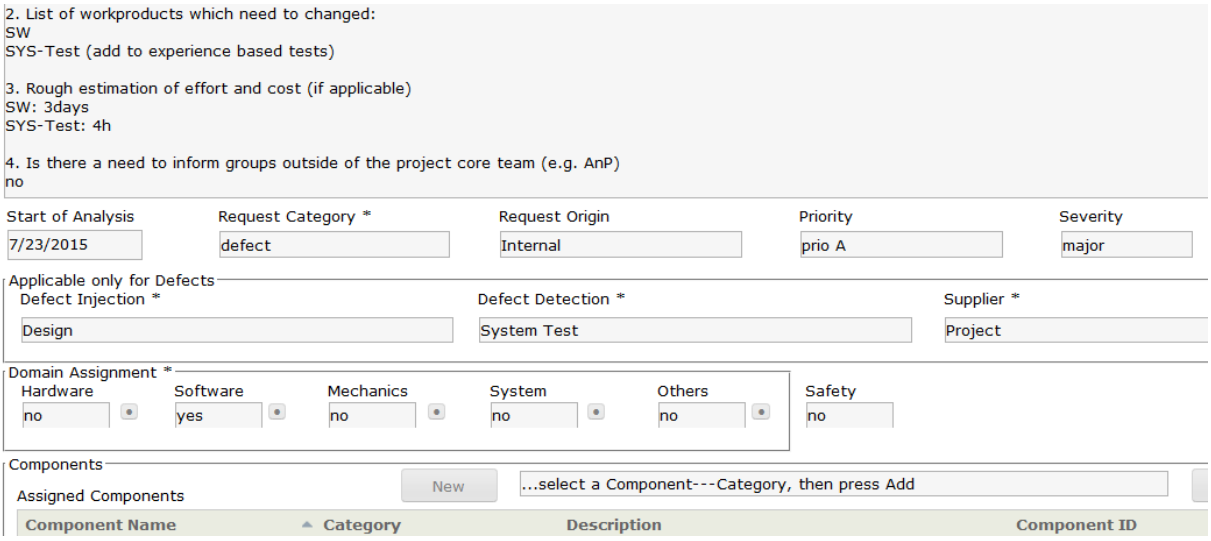


Figure 4.4 Defect CRM System Example

Figure 4.4 illustrates a defect change management system in CRM. Different information have to be fulfilled here. The effort for correction and test have to be estimated (2. and 3. in Figure 4.2). The defect injection (middle left in Figure 4.2) represents the phase where the defect is

originated, and the defect detection represents the phase where the defect is detected. The software component that includes this defect is declared as a component name and its description.

Figure 4.5 illustrates the data measurement concept in the model. Every node has a relation to the feature it describes. A feature is related to one or more requirements, whereas every requirement is related to exactly one feature. The nodes change, defect analysis and defect fixing 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 fixing 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 fixing node describing what needs to be fixed to remove the defect.

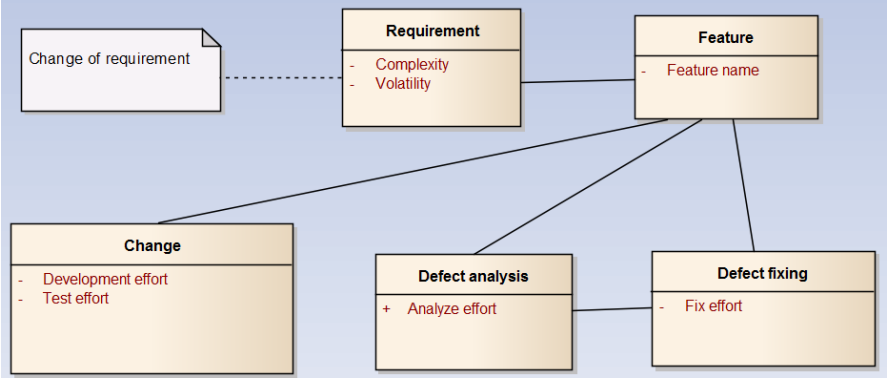


Figure 4.5 Measurement Concept

A change could be required because of development activities or defect correction activities which is an important information to link the effort to different budgets of the project. The data set includes all information about the effort as an estimation prior to executing the change request, and the effort needed to develop or correct each change after the change request was closed. This effort of development or correcting a change request is described in this thesis as Effort of Development/ Defect Correction (EoDC) including the relation to the released feature and the change effort.

4.1.1 Change Effort Categories

Software change requests have different scopes and categories depending on the importance and size of the changes. The CRM categorized according to the importance and size and technically realized with attributes in the data set.

To ensure the quality of the software part in the RCM different activities have to be planned depending on the category of the change. The categorization used in this model depends on the purpose of the change and their relation to the development feature, either the change is related to functionality or not, and the priority of this change which is the main input for the dead line of it.

Changes are categorized by measurement categories as follows:

- small: effort less than to 8 hours.
- medium: effort between 8 - 32 hours.
- large: effort more than 32 hours.
- other: unclassified changes like architectural changes.

- bugfix: The change needed for correct a defect.

The priority of the changes are used to indicate safety critical aspects in the software as well as their urgency. The priority levels are set to A, B, and C, where A represents the highest priority and C the lowest as shown in Figure 4.1c.

4.1.2 Change Distribution

Different projects data sets were analyzed for failures in different phases of the software lifecycle. Figure 4.6 illustrates different change requests in one of the projects with its development and bug-fixing effort used. The project development change request effort was 2147 hours and the correction change request effort during the project lifecycle was 1262 hours, the complete change effort was 3409 hours. The number of changes request were 254 development changes and 178 defect changes with cumulated number of change request of 432.

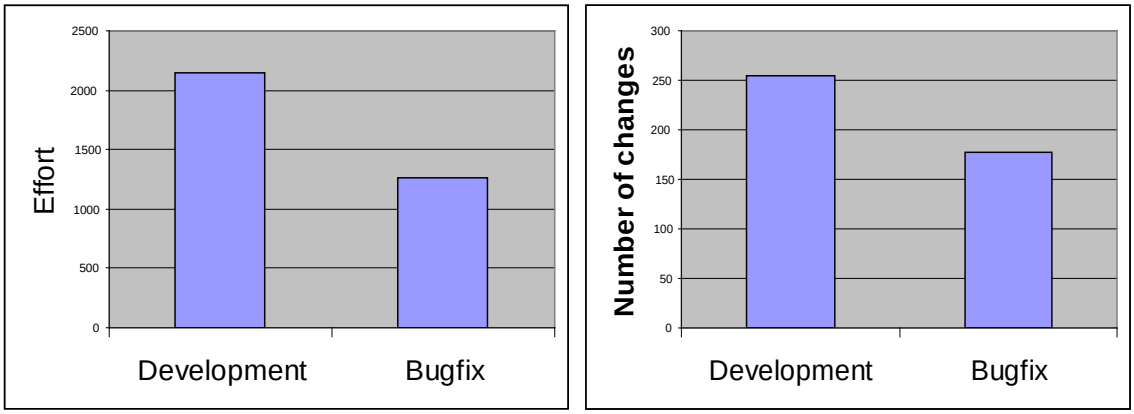


Figure 4.6 Defect Distribution over Effort and Number of Changes

The results of this project sample is an average overall change effort is 7.89 hours per change. The average development effort of each change is 8.45 hours per change, and the average correction request effort is 7.09 hours per change. Table 4.1 shows the calculated result ranges. Figure 4.5 presents the distribution of CRQs from the full changes, and the effort for the data-set respectively.

No. of total changes	3409
Total effort	432
Average effort/change	7.89
No. of Development changes	2147
Development effort	254
Average development effort/change	8.45
No. of defect changes	1262
Defect effort	178
Average defect effort/change	7.09

Table 4.1 Development and Correction Effort

The category of each change in the data set used were 172 small changes, 37 medium changes, 9 large, 36 other and 178 defect changes. The change dispersal could be classified to 40% ~small, 9% ~medium, 2% ~large, 8% ~other and 41% correction-changes.

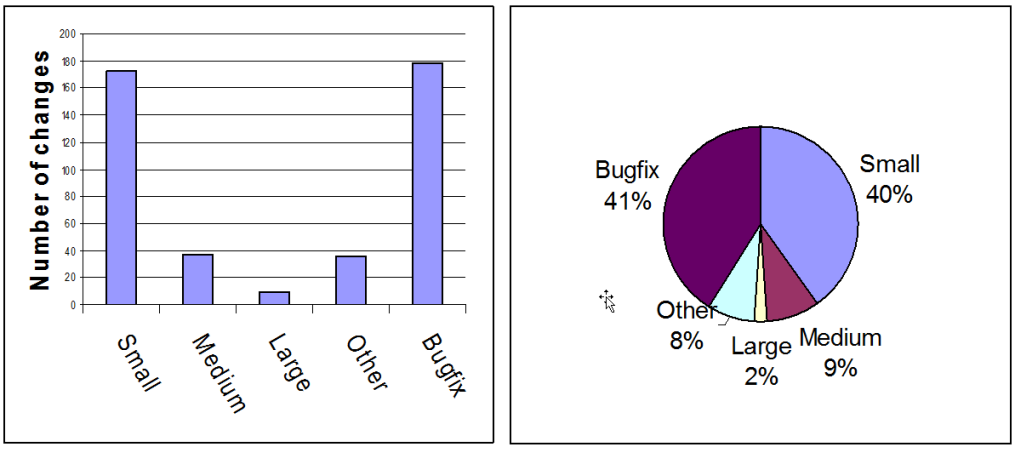


Figure 4.7 Change Request Distribution Based on Number of Changes

The resulting efforts from the data-set analysis are ~805 hours for small, ~587 hours for medium and ~377 hours for large changes. “Other changes” demanded 276 hours, and the correction effort demands 1202 hours.

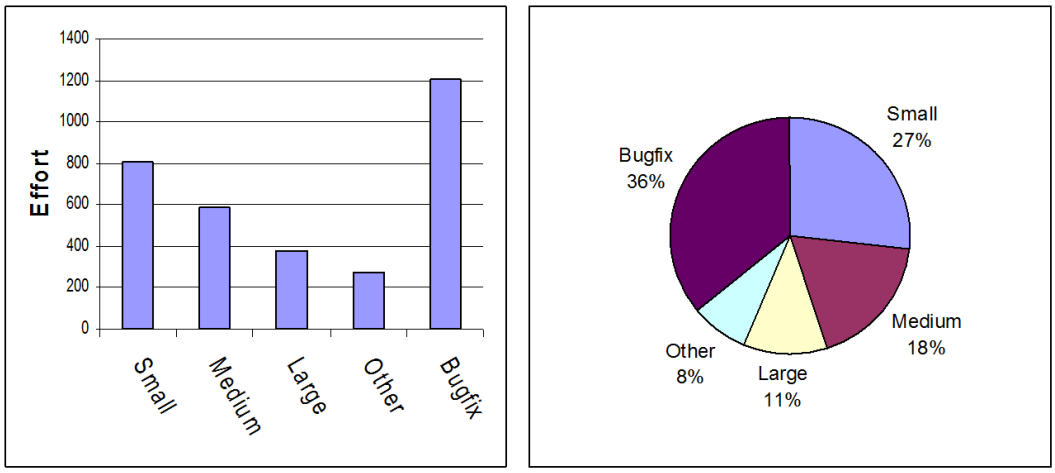


Figure 4.8 Change Request Distribution Based on Effort needed for Change

Table 4.2 illustrates the size of change requests and the effort needed in order to complete them with respect to their categories and to the average of each change category. The changes of the class small represent 40% of the data-set entries and 27% of total effort result in the deviation of 13%. The class medium shows 9% of total changes and 18% of effort obtained in the deviation of 9%. The large changes shows 2% of total changes and 11% of total effort obtained in the deviation of 9%. The other changes have 0% deviation, and the number of change and their effort have the same percentages of 8%. The correction category in the data-set has 41% of the entries, and shows 36% of total effort, 15% deviation.

The priority of the change request represents the deadline and pressure of this request to be completed. Figure 4.8 illustrates the change requests distribution based on their priorities.

Type	No. of changes	Effort	Deviation
Small	40%	27%	13%
Medium	9%	18%	9%
Large	2%	11%	9%
Other	8%	8%	0%
Correction	41%	36%	15%

Table 4.2 Type Distribution Error

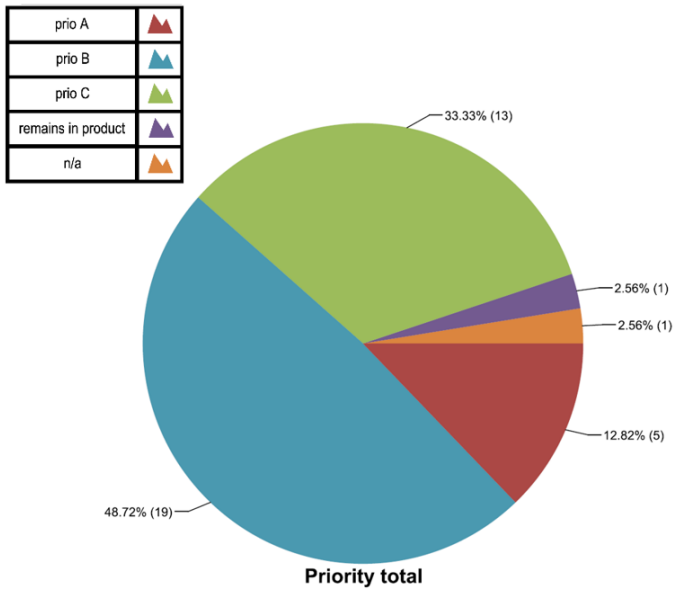


Figure 4.9 Change Request Distribution Based on their Priorities

The effort will be used instead of the number of changes to describe the models in next sections.

4.2 Feature Classification

The features of software are represented by different functionalities. Each software change is related to a specific functionality of the software and a feature. The functionalities of the software are grouped in different features, each with its data. Automotive software is a multi-layer structure software, with the architecture based on AUTOSAR architect [94]. The commonly features (here laers and components) are:

- MCAL. Microcontroller layer, deals with the vehicle's busses and hardware.
- SRV: Service layer, present the Interface from MCAL and an upper layer of software (RTE and HMI), this layer included the Bus configuration drivers.
- HMI. Human machine interface, Interface from MCAL/ SRV to upper software layer.
- RTE. Run Time Environment, Interface between Application components to HMI.
- APP: Application component.

Analyzing the data set shows that the different features of software have different rates of failures and defects. These differences are due to different aspects of the features. One important aspect is the requirement complexity, as well as interfaces to other features, implemented code complexity and the safety-criticality level of the feature. Due to these differences, an overview on the whole project is needed. A good traceability matrix includes most parameters, which has a major effect on the software that should be built and used in the project. To optimize the change of software process and quality, an assessment of the effect of correction of the defect and their effort is needed.

4.3 Failure Effort Parameter (FEP)

Automotive project is a long-life development product with a high safety-critical condition software; therefore, a high-level quality is required. Due to the dynamic demand of automotive domain, there is a continuous need to rework the requirement of the software owing to some change in the requirement itself or a change in different feature, which has an interface to the feature of a specific requirement, or alternatively due to fix a bug in the product that needs a change in feature and requirement. In order to ensure the functionality of the system with a high quality in a complex system like the automotive systems there is a need to deal with a large number of CRQs. The effort needed to complete these CRQs will be included in the total development effort of the system.

The effort needed for correction of the failure and defect in software could be derived from the rework of the requirements and features in the software, whereby a defect and failure rate could be analyzed and calculated from the total rework of the software project. Automotive software is developed by using a develop process and a developed framework to satisfy their requirements with specific quality needed. V-model and agile developing techniques are mainly used with automotive software development. The system and software will be developed in several phases of the software lifecycle. A software defect deals with each failure in each phase during the development of the software. Some failures could be detected in the same phase of development where the failure is generated by having the quality assurance activities in this phase, whereby the effort to fix such a failure will be small compared with other failures detected in different phases of the development phase lifecycle to where they are generated. These failures will need more activities to be fixed, such as an analysis and rework of other levels where the failure is detected. The effort of correction of software is increased when they are detected later due to the analysing effort, tools handling, test and integration. In order to avoid an extra effort of correction, different quality assurance activities are needed and a good traceability system should be supported during the development. The effort needed for software development can be classified into two types of effort, namely the effort needed to develop the software (DevE) and the effort needed to correct the software defects and failures (SCE). These parameters help to calculate the failure effort parameter (FEP) as in equ. 4.1.

$$FEP = SCE / DevE \qquad \text{equ. 4.1}$$

Where:

SCE is Software correction effort, is defined as correction.

DevE is effort needed for software development, applies to changes of categories (“small”, “medium”, “large” and “other”).

The Size of software, the complexity of software components and the interface among the different parts of software affect the effort needed to correct the defects and failure in software. The automotive software project have high development efforts because of their safety critical characteristics, and they have a high complexity of the features. This increases the effort of

software defects and failure corrections. FEP represents this relation. For data set used in this chapter, according to equ. 4.1 the *FEP* results to:

FEP = 1262 / 2147 = 0,587 equ. 4.2

An *FEP* of 0.587 indicates less correction effort than development effort. With a FEP value of greater than one, a company spent more effort for correction than for the development. Based on the experiences with the industrial projects a FEP in between 0,2 and 0,6 is acceptable. Values of greater than one are a string indicator for a re-structuring or re-development of a software.

Category	MCAL	HCC	RTE[h]	APP[h]	SRV[h]
Development effort [h]	496	694	173	657	127
Defect correction effort [h]	436	707	67	44	8
DCF	0.88	1.01	0.38	0.07	0.06

Table 4.3 Defect Effort Parameter

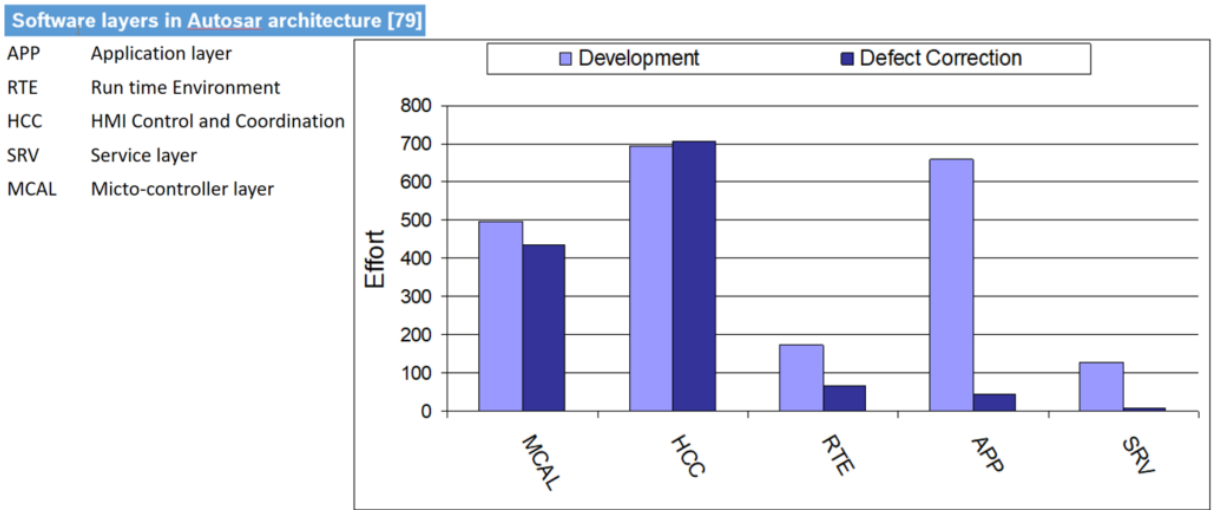


Figure 4.10 Change Request Effort Distribution over Features

Figure 4.10 illustrates the distribution of SCE based on the different features in the software projects analyzed. The different features represent different relations between the effort needed for software correction and the effort needed for software development. Different features show different values of SCE compared to the effort needed for their development. The microcontroller layer (MCAL) with a development effort of 496 hours and an SCE of 436 hours has, an FEP of 0.88. HCC presents the development effort of 694 hours and an SCE of 707 hours, leading to an FEP equal to 1.01. The complexity of HCC development is higher than its maintenance effort. The service layer (SRV) presents a lower FEP of 0.06 obtained from the development effort of 127 hours and the effort for correcting the defection of 8 hours. The SRV features used a generator tools, the defects were injected because of a wrong understanding of requirement or defect in the requirement analyzing phase.

Category	MCAL	HCC	RTE	APP	SRV
Development effort	496	694	173	657	127
Correction effort	436	707	67	44	8
FEP	0.88	1.01	0.38	0.07	0.06

Table 4.4 Effort Distribution over Different Features with Different Complexities

In [95] a system has been developed and used to identify the KPIs needed for SCE prognosis. The experiments were conducted with different 20 participants out of the development team. The main point discussed in the experiment was to analyse the effect of time pressure on the average rate of software defect, as well as studying the effect of the code generator tool on the cost of software. The experiments were conducted in an automotive domain, using the automotive tools for requirement and quality management.

	Deadline pressure		
Measure	High	Medium	LOW
Min.	1	0	1
25%	3	5	1
Medium	4	6	1
Mean	4	5	2
75%	5	7	4
Max.	6	7	5
Standard deviation	2	3	2
Variance	3	11	4

Table 4.5 Number of Defects over Deadline Pressure

5 Development and defect correction KPIs

As shown in the last section, the effort of software development in automotive domain includes the effort of software development itself and the effort of software defects correction. This chapter describes the Key Performance Indicators (KPIs) needed to estimate the development effort needed and the effort of defect correction during the development lifecycle. After identifying the KPIs that affect the rate of defects in a software product, a software process model (SWPM) and a flow of correct cost (DCF) model are presented to assist PMs in their daily work. These models are be built and tested. The use of the models are described and analyzed with the results presented in the results section.

5.1 Process model for software development

Automotive software release include several changes, as illustrated in Figure 5.1. Each of these changes will affect product and process factors and has its own probability of causing a new bug or defect. The effort needed to manage software defects and bugs during the development lifecycle is an important parameter that affects the total software effort and cost. The SWPM helps to optimize the estimation of effort needed to correct software defects in a project, as well as helps project managers to finally make their decisions.

The model of this thesis is built with a dynamic structure, whereby this structure offers a possibility to deal with and analyze the effect of changes on the product and the process of development over time and helps to improve them. This can help project managers to estimate the effort needed for the current and future phases of the development lifecycle. Having an overview of all changes to the system and the effects of each change made during the different phases and releases of the software will give managers a better overview of the system's complexity and help them to make better estimates [96].

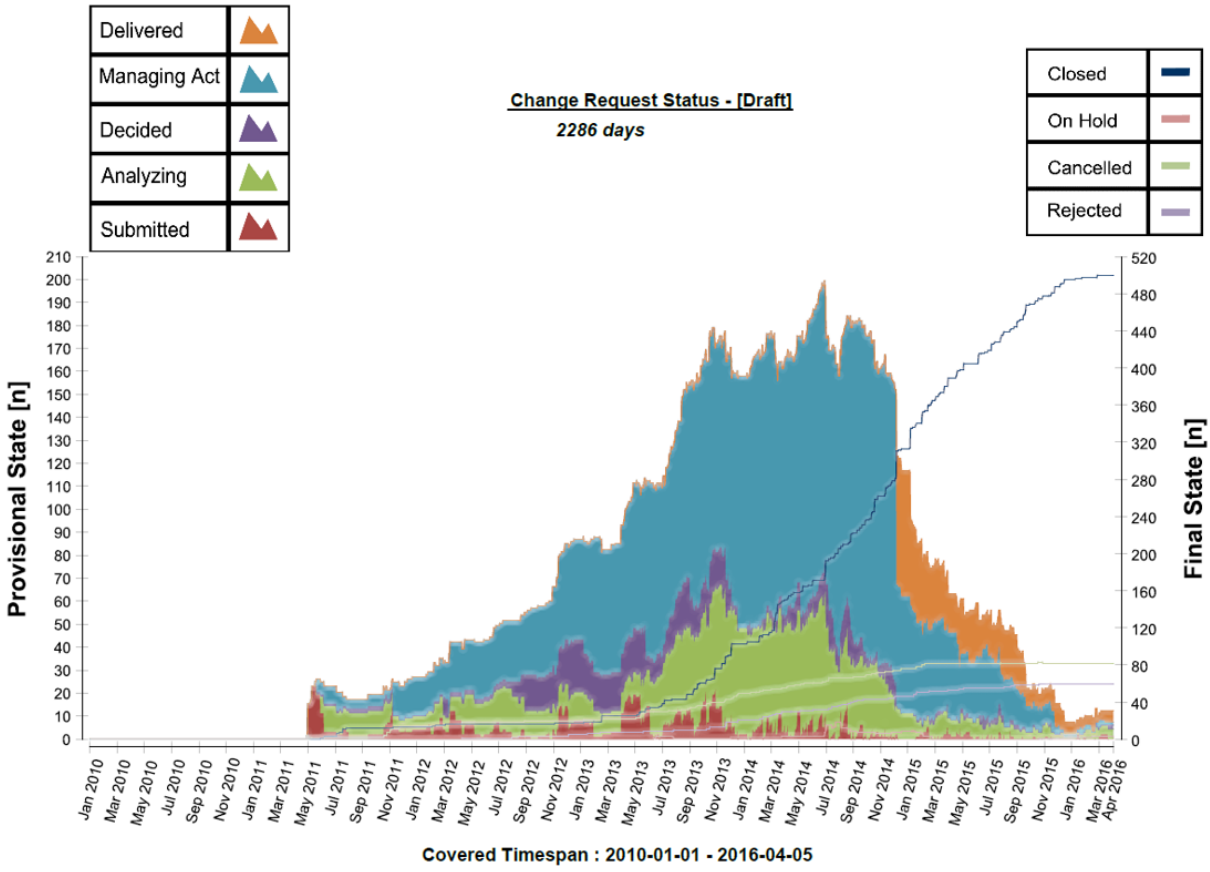


Figure 5.1 SW Releases

Figure 5.1shows the change request in a real software project during the development lifecycle. 512 changes were fixed and 147 changes were cancelled and rejected after analyzed during this time. These activities affect the total cost of the product and need to be estimated carefully.

5.1.1 Problem Definition

The model of this thesis has been developed to estimate the effort of defect correction in relation to the development effort for a specific feature of a software product, e.g. the Human Machine Interface (HMI) based on Autosar [24]. SWPM is intended to be used in a number of analyses to support the project managers to plan or asses their software project. The major advantage of SWPM is its capability to take KPIs into account having a significant influence on the estimation variables, i.e. effort of development and defect correction. 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.

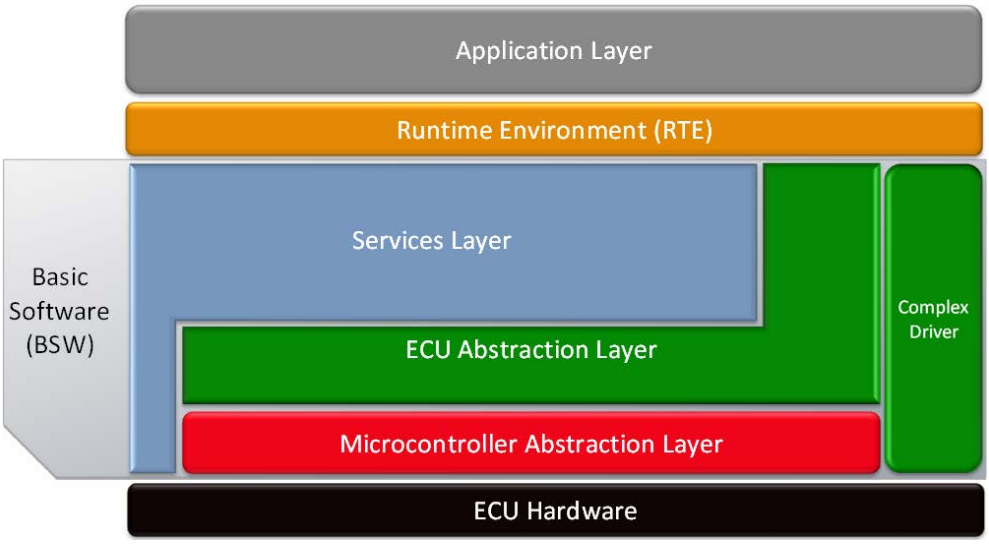


Figure 5.2 Autosar Software Layers [24]

The new estimation model introduces parameters and factors that affect the effort needed to finish a project. The parameters and factors in the estimation model can be divided into five categories:

1. Process effort: the effort needed for process activities, and the maturity level used.
2. Product effort: the effort needed for documenting quality and other different engineering artifacts.
3. Project effort: the effort needed for project.
4. Change effort: the effort needed for changes to be developed.
5. Test effort: the effort needed to integrate and test changes.

The analysis of the effort needed to correct the software defects (SCE) in last chapter showed the effects of defects on the total software effort. Thus, the total engineering effort of a product is a combination of development and change efforts, refined the above five categories. Figure 5.3 illustrates the process of release planning, including effort estimation, which the SWPM structure is based on.

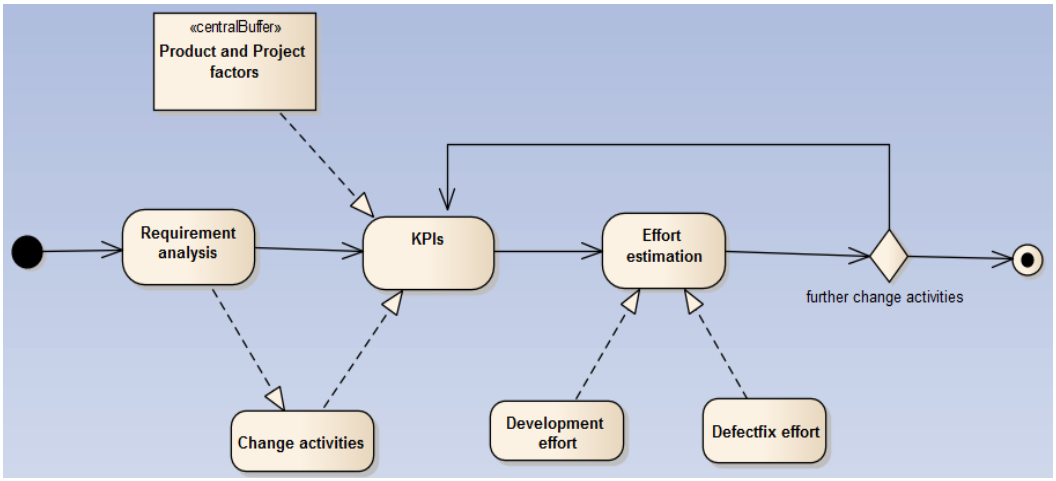


Figure 5.3 Effort Estimation Process

The overall product requirements are divided into number of features, where each feature has a unique characteristic. The process begins by analyzing the requirements for each feature. The changes were be planned to define the release packages and trace it. To predict the effort needed for each software change, we have to consider all KPIs for each change and analyze the effort needed to address the change by adapting the software. Only then can we estimate the total development cost with a high accuracy.

5.1.2 KPIs and Relationships among them

The software metrics approach using the goal question metric (GQM) has been studied by Viktor Basili [90] [97]. The GQM approach was used to identify the traceability of the system components and their relation. The GQM approach was used because a similar matrix already existed for the system from the failure mode and effects analysis (FMEA) [96], which is a mandatory part of the development process for automotive software, as illustrated in Figure 5.4.

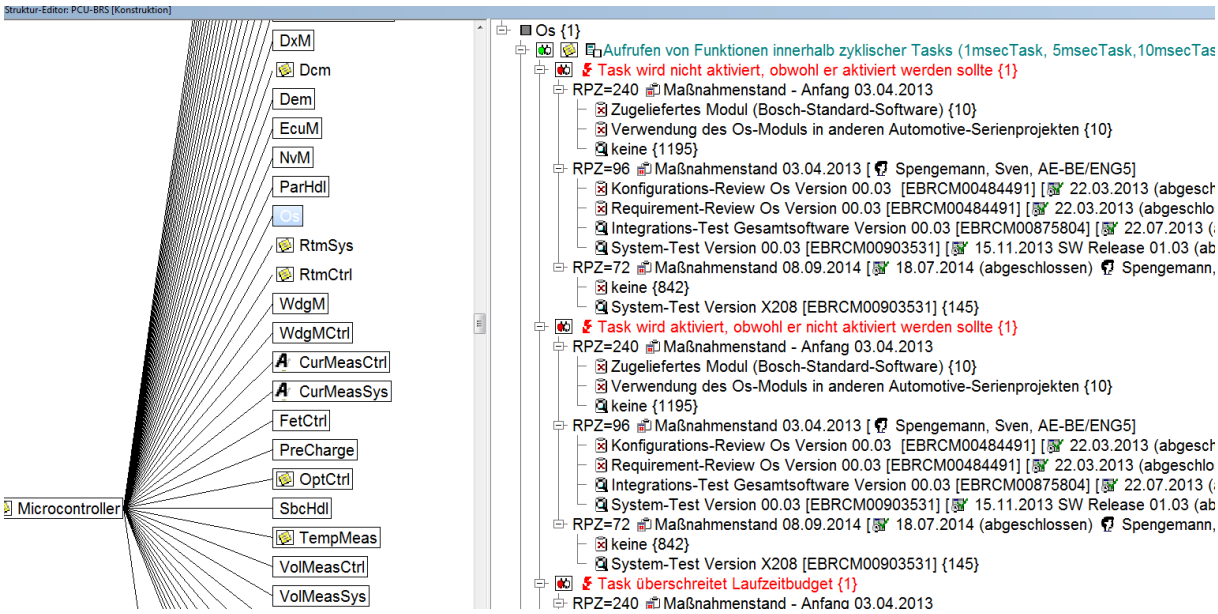


Figure 5.4 Example of FMEA

The Goal Question Metric (GQM) approach was used to identify relevant KPIs of the model and the relations among these KPIs. These KPIs were derived from the problem definition

which is broken down into its major components for further characterization. The questions were used for the refinement including the questions of project managers to understand the problem. There is a set of metrics used to characterize every question. Metrics represent a model to measure properties of any engineering artifact, KPIs in SWPM respectively. The following enumeration holds a set of questions (Q) along with their metrics (M) characterizing the goal of SWPM, as shown in Figure 5.5.

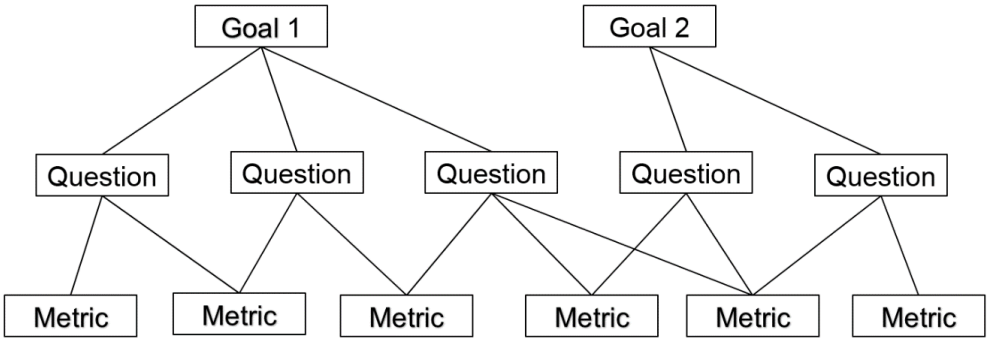


Figure 5.5 GQM Model Hierarchical Structure [90]

The most important Qs with their M and G are represented in Table 5.1:

Q	M
Feature difficulty?	M1 feature difficulty. Difficulty of feature needed different effort? M2 feature volatility. The possibility of requirement change? M3 feature complexity. Difficulty of requirements of feature?
Engineering effort?	M1 engineering effort. The effort needed for developing a single change including all test activities M2 type of change. Change level describing its complexity.
Potential effort for correction?	M1 effort of correction. What's the potential-effort needed to fix requirements aberration? M2 development effort. What's the needed effort to complete a change? M3 defect cost parameter. What's the indicator of a feature-error? M4 effort to correct a defect after testing. What's the residing defect correction effort need after complete a change.
Potential failure correction-effort?	M1 testing effectiveness. Defects found by test. M2 effort of testing. Test effort needed after implementation. M3 test ratio. Test-effort comparing with development-effort. M4 test payoff. Defect detection potential. The quality of test activities? M5 defects found. Reduce the correction effort of software defects.

Product factors?	M1 product factor. Using the existing product as a base for new development. M2 documentation quality. Understandability of the product developed. M3 code quality. Understanding, Changing, Enhancing the existing code base.
Project activities?	M1 deadline pressure. Different kind of pressure during the development cycle, shouldn't effect fixing change within a specific time frame.
Process activities?	M1 process activities. Measurement of less error-prone engineering. M2 process organization. Maturity-level of the organizational process. M3 process application. Implementation.

Table 5.1 GQM for Software Change

5.1.3 Estimation Model

Building a model to help project managers and decision-makers to make their decisions requires wide knowledge of the development phases to be included in the system. Including this huge amount of information in a system increases the complexity of the model. The requirement and design of the software are mandatory information for the model. The goal of the model is to describe the system as a network. It should be represented as a cause-and-effect chain, with all system components and their interrelationships. The crosscutting concerns network illustrated in section 2.2 is used to model the system, and the continuous iteration of the network allows different parameters to be calibrated. Representing the system as such a network requires system design knowledge.

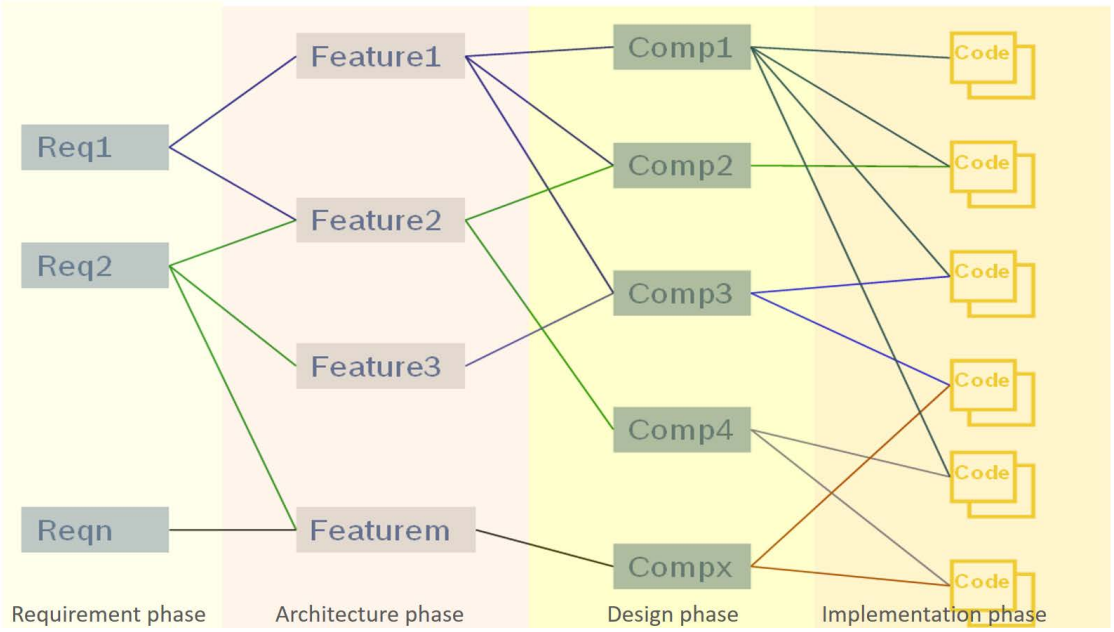


Figure 5.6 System Architecture

Knowledge about the development process must also be used to develop the network, due to the effect of the development process on development effort. This knowledge of the develop-

ment process can be supplied by the Project manager. The different phases in the development lifecycle, the requirements, feature design, module design, implementation and validation and verification phases should all be described in the system, with sufficient traceability to track each change both forwards and backwards. A traceability matrix will be built to include this knowledge and all other documentation for the tools, risk management and other documentation needed. Figure 5.6 provides an overview of the system architecture and the development phases from the requirement level to the implementation and documentation level.

Figure 5.7 shows the main part of the estimation model. The necessary KPIs are represented in the model as nodes, with the edges between the nodes representing the cause-and-effect approach [98]. The model is divided into five categories to represent all KPIs in the change model, namely change parameters, test activities, product parameters, process activities and project parameters.

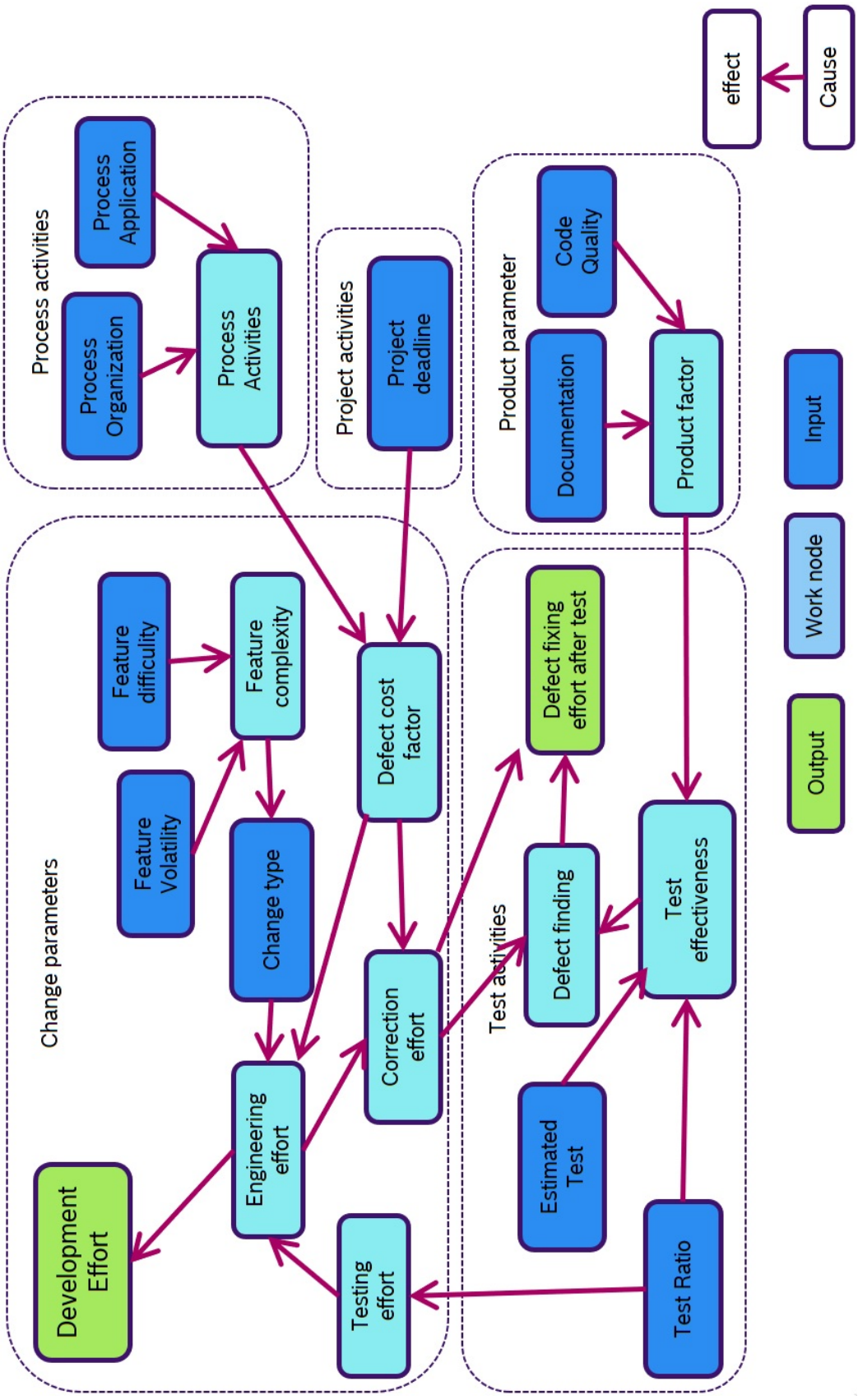


Figure 5.7 Software Process Model

5.1.3.1 Model calibration

After gathering the information needed, the SWPM was built. To set the different values for the model parameters, calibration nodes were added to the estimation model to ensure the expected result. Figure 5.8 shows the setup of the model process. Different KPIs needed to be calibrated to achieve an accurate estimate of the software development effort.

Requirement analysis is the first phase of the development lifecycle, where software requirements are analyzed and classified as input for the next phase. The requirement analysis is represented as an entry node in the model. The requirement classification leads to the features analysis, where the features are categorized and classified. In this phase, the complexity of the features is assessed, which requires expert knowledge. This information can be gathered from old projects with similar development environments, and some other knowledge will need expert knowledge in the first model setup step. The old project data and experiences help to illustrate the change distribution of different system features. These experiences can be used as a basis for estimating the degree of the crosscutting network. Other features could share similarities with previously-developed features. The experience from these features could be helpful here when considering the new characteristics of the project requirements.

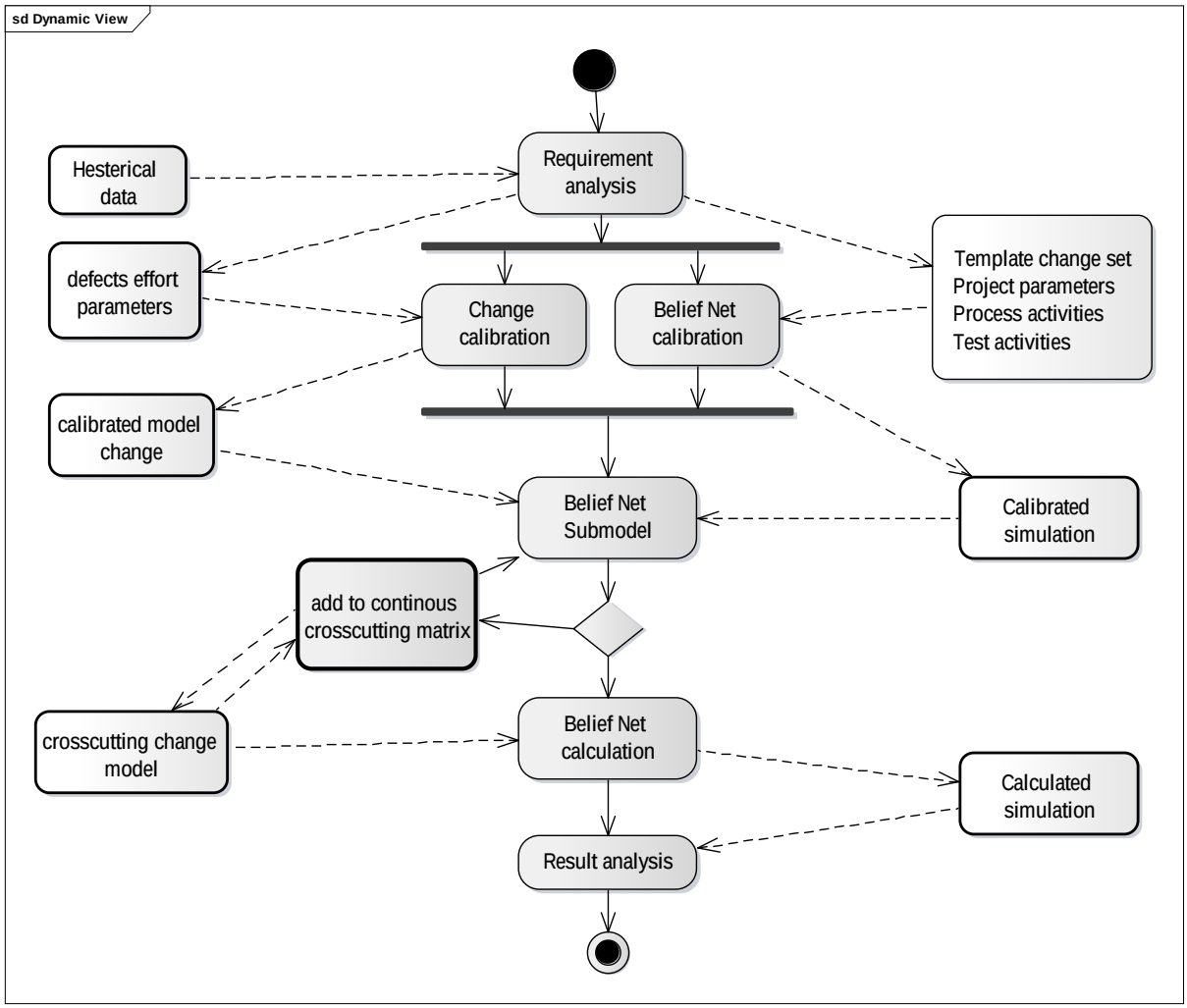


Figure 5.8 Software Process Model Setup Overview

A lot of information about the system architecture is needed to build the model; therefore, the system needs a traceability matrix to provide an overview of the software architecture and helps to establish the nodes and their connections. To establish the change model, the change

distribution parameters, project parameters, process and test activities were used. The parameters were initially defined via template values defined by a change distribution list. This template list was used for small, low complexity changes to features. The project parameters describe the effect of source code and documentation on the project as artifacts of engineering quality. The environment of the project is represented as process activities and test activities.

After establishing a network for each model in the system, a continuous network was created for the whole project. This network was improved by iterating several times over the defined changes. After establishing the continuous network for the system, we started to run our calculations. These calculations were performed for every model in the system and for system as a whole. The result of this calculation was stored in the simulation environment for every model network and every change for every KPI. This will allow managers to see how the estimated values evolved over time.

5.1.3.2 Change parameters

The relationships between different changes to specific KPIs are known as change parameters. One of the most important parameters is the complexity of the feature, which is influenced by the feature characteristics. This complexity of the feature depends on the difficulty in meeting the associated requirements; therefore, it is necessary to complete requirement analysis carefully to obtain better features. Feature complexity must be classified into different levels of complexity to secure a better estimation of change. This should be done via the requirement management system. Feature complexity was classified into three levels, namely low, medium and high.

Feature complexity is a very important parameter as it affects the software correction parameter. Low complexity features have low correction parameters, while features with high complexity have high correction parameters. The process activities and project engineering environment quality will both affect feature complexity. Figure 5.9 illustrates the change parameter and its influencing parameters.

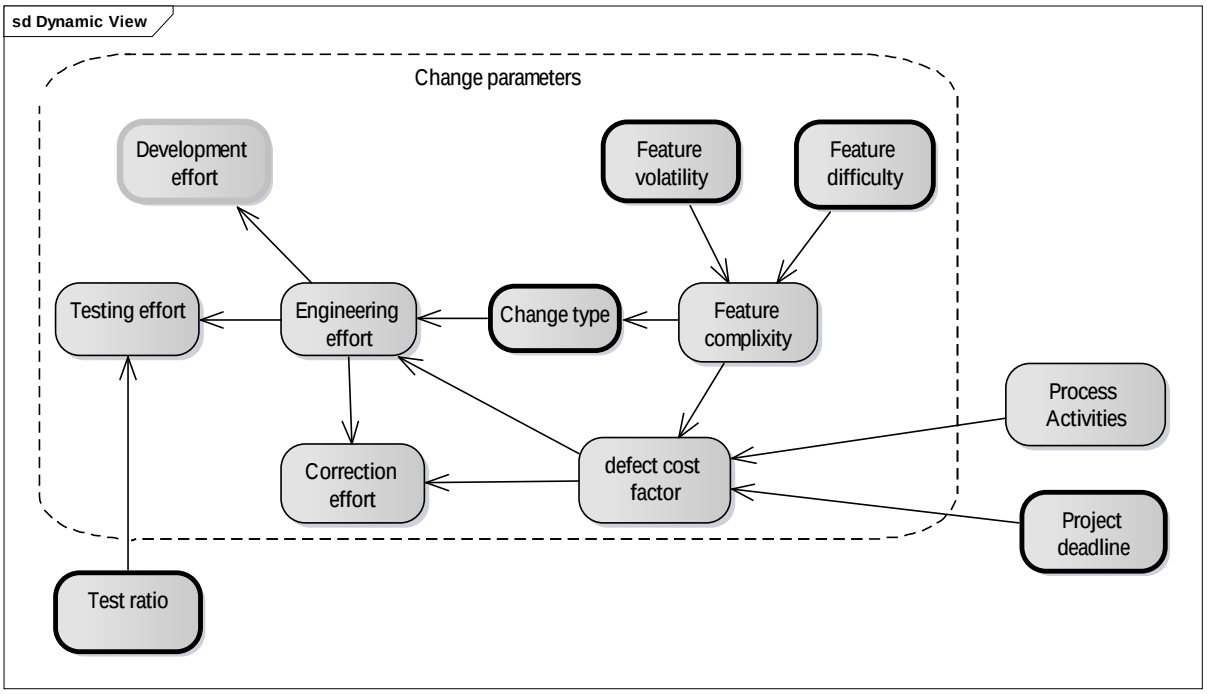


Figure 5.9 Change Parameters in Software Process Model

The main affected change parameter is the feature complexity which depended on the feature difficulty and volatility, these are used as calibrated nodes for this parameter. The second parameter is represented as engineering effort which included all activities because of the change, i.e. change type, defect cost factor, test and correction effort. The model was calibrated using different datasets. Some of this data was based on statistical data derived from project data from old projects, while others was based on expert knowledge and literature. Feature volatility and complexity were calibrated using the expert knowledge available within the organization. Table 5.2 illustrates a data analysis to describe the complexity of features used. Defect and bugs parameters were found from internal data.

Table 5.2 illustrates that HCC development has the highest FEP due to its high volatility and complexity. MCAL follows HCC with medium volatility and high complexity. RTE has a complexity factor but low FEP, due to the use of a high quality code generator tool. APP and SRV exhibit low FEP. The type of change can be derived from feature complexity. Low complexity features affected low effort maintenance parameters. Higher complexity features caused more development effort requiring greater change effort.

	MCAL	HCC	RTE	APP	SRV
Volatility	Medium	High	High	Low	Low
Complexity	High	High	Low	Low	Low
FEP	0.88	1.01	0.38	0.07	0.06

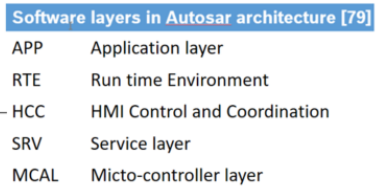


Table 5.2 Features and FEP

The type of change is an important parameter for deriving average engineering effort. This involves all change activities, including requirement management, design, implementation, and testing. Table 5.3 illustrates the effect of change type on engineering effort for different components included in our dataset.

The change within a feature's FEP need for information about the ability to have the effort needed to correct the software from the engineering effort. The estimation presents the effort needed to correct the software by changing implementation. This is the basis for the potential average number of defects found by the test activities.

	Feature complexity			
Type of change	Low	Medium	High	Engineering effort
Small	63.27%	28.50%	19.00%	6.44
Medium	6.18%	17.85%	12.00%	21.50
Large	7.42%	1.00%	3.50%	56.00
Other	11.87%	6.24%	9.00%	9.72
Bugfix	11.26%	46.41%	56.50%	20.00

Table 5.3 Effort Change Distribution

5.1.3.3 Process Activities and Project Factors

Process organization and process application represent the maturity level of an organization and its application [99]. It is a measure described the capability of the process activities to ensure the continuous development of high quality products. Figure 5.10 illustrates the part of the process model that describes the influence of process activities on the effort needed for software correction.

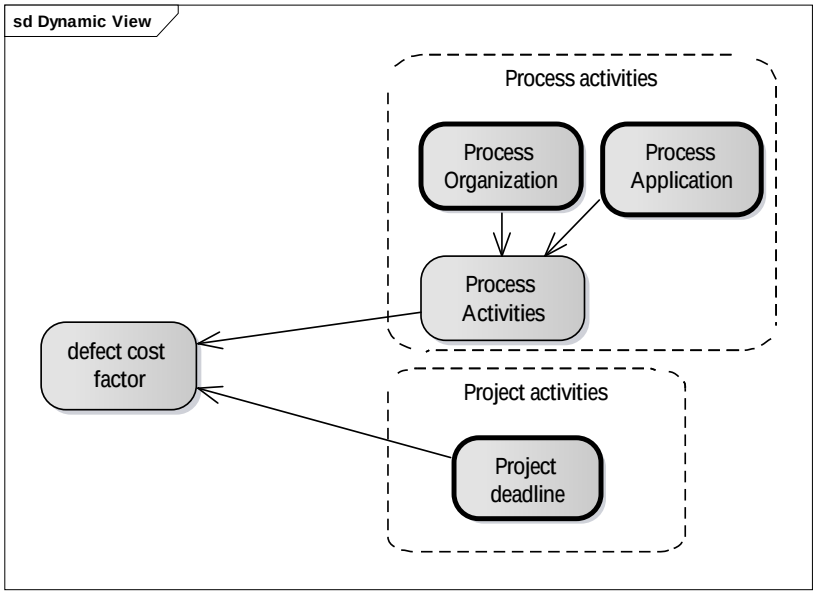


Figure 5.10 Process and Project Activities in Software Process Model

The process deals with the organization norms and standards, which have an impact on software effort. Due to the continuous demand for shorter development cycles, product deadlines strongly influence software costs. This important effect is described in the process model in Figure 5.10 as the “project activity”.

Various data from real projects with different parameters was analyzed to set the value of these parameters. Table 5.4 shows the results of the analysis, highlighting that the high deadline pressure to complete a CRQ leads to a reduced effort needed for process activities, whereby the average engineering effort could be reduced by 50% to achieve a desired goal. The priority parameter for the change request includes the deadline pressure of the project within the company system. Figure 6.11 present a data set of CRQs from a real project with the effect of project pressure on the total effort needed for software development.

	Process activities and project deadline		
Type of effort	Low	Medium	High
Correction effort	100%	300%	400%
Engineering effort	100%	50%	35%

Table 5.4 Deadline Pressure

The results revealed up to 300% greater effort needed for software correction compared to ideal engineering conditions. Deadline pressure in a project with low process activities could reduce the average engineering effort by 35%, although correction costs increased up to 400%.

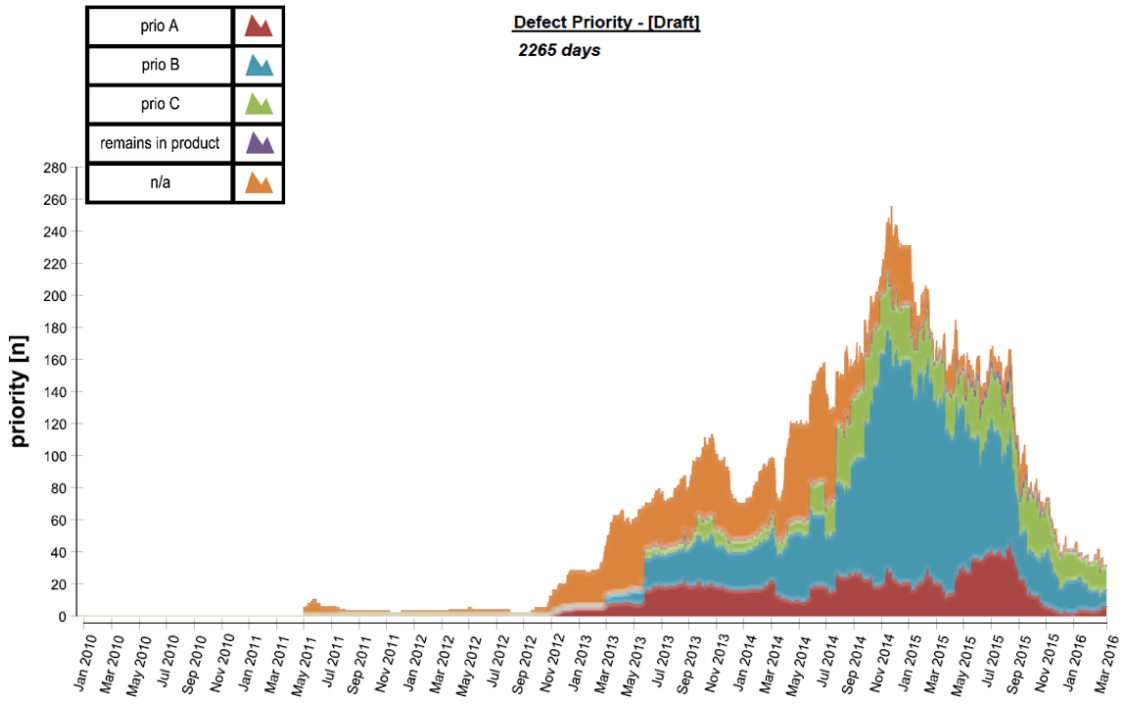


Figure 5.11 Priority Distribution during Development Lifecycle

5.1.3.4 Product parameters

The product parameters reflect the testability of the software. These will have a main effect on the error rate after the phase of testing. High documentation and code quality lead to good product factors, which in turn lead to better error detection in the software.

Figure 5.12 illustrates this part of the model that describes the effect of these parameters on the correction effort.

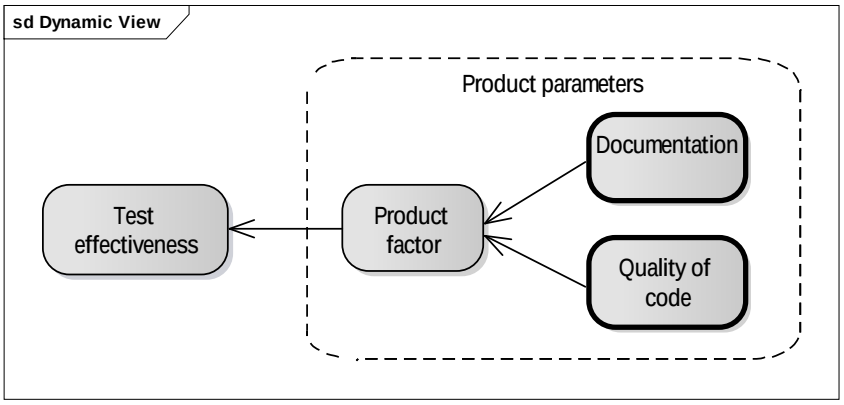


Figure 5.12 Product Parameter in Software Process Model

The risk management and review activities help to secure better product quality. The review activities are included reviewing the requirements, design and code. Higher effort activities lead to a higher product parameter, which in turn leads to a higher rate of detecting software failures and bugs. The experts weighted the minimum effect of these factors in a node, taking into account the phase of the development lifecycle. Three types of reviews were used to assess the quality of the project. The review type depends on the volatility and complexity of the change [100]:

- Inspection: used for high complexity changes.
- Pass around: used for low complexity and high volatility changes.
- Confirmation review: used for low volatility and low complexity changes.

This classification determines the effort needed for review activities. Inspection reviews involve high effort and must be conducted by multiple reviewers. Pass-around reviews also require multiple reviewers, but fewer. Finally, confirmation reviews are carried out by a single reviewer for changes needing less than eight hours to be corrected.

5.1.3.5 Test Activities

Automotive process and work lifecycles devote significant effort to test activities to enhance the quality of the delivered product and satisfy all requirements and requests. Figure 5.14 illustrates the test part of the model. This part of the model describes the effort needed for software correction and the factors that affect this effort even after testing the software. The test phase in the development life cycle detects many failures and defects generated in the earlier phases, as shown in Figure 5.13. It is necessary to estimate these to get to an overall product estimate.

Some defects can be found as a part of the estimated software correction effort. These will be represented in estimated test activity. The test ratio activity represents the engineering effort needed for this phase. The estimated test and test ratio efforts influence the test effectiveness, as shown in Table 5.5. The defect rate after test activities is a measure for test effectiveness for these activities.

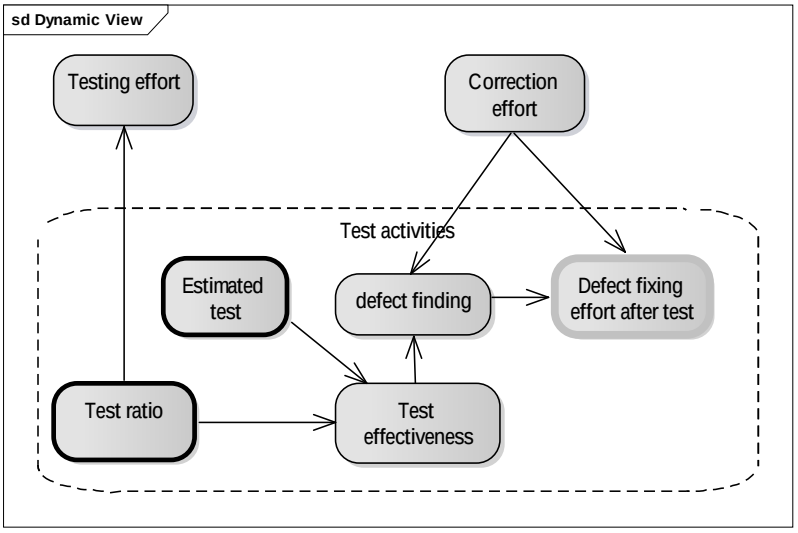


Figure 5.13 Test Activities

Test ratio	Test/ Engineering effort
Low	10%
Medium	21%
High	39%

Table 5.5 Effort Spent on Testing

Table 5.6 illustrates different detection-rates for all test-ratios and estimated test levels. The Detection-rates were analyzed using some external data [101] to indicate whether a test is successful.

Information and expertise was complemented by expert knowledge. The effort needed to correct defects and failures differs depending on the test level. System test effort is represented as an arithmetical node which includes the estimated effort for software correction from change parameters with test activities.

Test effectivity		
Test payoff	Test ratio	Detection rate
Low	Low	3%
Low	Medium	17%
Low	High	28%
Medium	Low	6%
Medium	Medium	36%
Medium	High	55%
High	Low	14%
High	Medium	62%
High	High	84%

Table 5.6 Defect Detection Rates

5.1.4 Model Usage

After being built, the model was validated. Some workarounds were needed to fix parts of the model to evaluate its performance. The model was evaluated to assess the accuracy of its estimates and ensure that it would help Project managers with their decisions.

The model was verified via a simulation of different features in a system. The goal here was to analyze the model's performance in terms of estimating software effort for different changes. This validation considered the complexity and volatility of different features (MCAL, HCC, RTE, APP, and SRV). The dataset used to calibrate and validate the model was generated from an internal dataset.

The model was required to help the Project manager make decisions via accurate estimate. Therefore, the model was tested in this regard.

This was tested using two test scenarios (S1 and S2) with different conditions. Different features were developed with different change requests to reach the specific requirements and quality desired. The important development features were two features – HCC and SRV – as part of the service layer, with 30 iterations simulated to represent the change and development effort. The two scenarios were simulated with different product requirements. To analyze the effect of the deadline pressure in the project, the first test scenario was done with medium deadline pressure whereby the developer could take care of all process activities. The second test scenario was defined after the project deadline had closed, whereby the pressure to complete the activities was high and the developing was done with low process activities and test activities.

5.1.4.1 Results

Table 5.7 shows the validation output for the model test. Developing the features required different effort in the process and engineering models. The MCAL feature needed and estimated 237 hours of engineering effort for the process model and an estimated 231 hours of effort for failure correction. Estimation accuracy was judged based on an internal reference dataset. The process model achieved an accuracy value of 95% for engineering effort and 96% for correction effort. The estimates for other features also showed good results. For the features HCC, RTE, and App, the estimation accuracy was over 80%. The estimates for the

SRV feature were less accurate, at 62% for engineering effort and 25% for correction effort. The SRV feature is supplied with the project and configured using a configuration tool.

	Estimation		Reference		Accuracy	
Feature	Eng.	Corr.	Eng.	Corr.	Eng.	Corr.
MCAL	237	231	250	223	95%	96%
HCC	281	331	292	352	96%	94%
RTE	123	46	142	34	86%	74%
APP	216	23	268	19	80%	82%
SRV	74	8	46	2	62%	25%

Table 5.7 Prediction Accuracy

Figure 5.14 illustrates the results of the validation of the scenarios S1 and S2. The model used several estimates for different change requests. 1216 hours and 618 hours of development effort were estimated for scenarios S1 and S2, respectively.

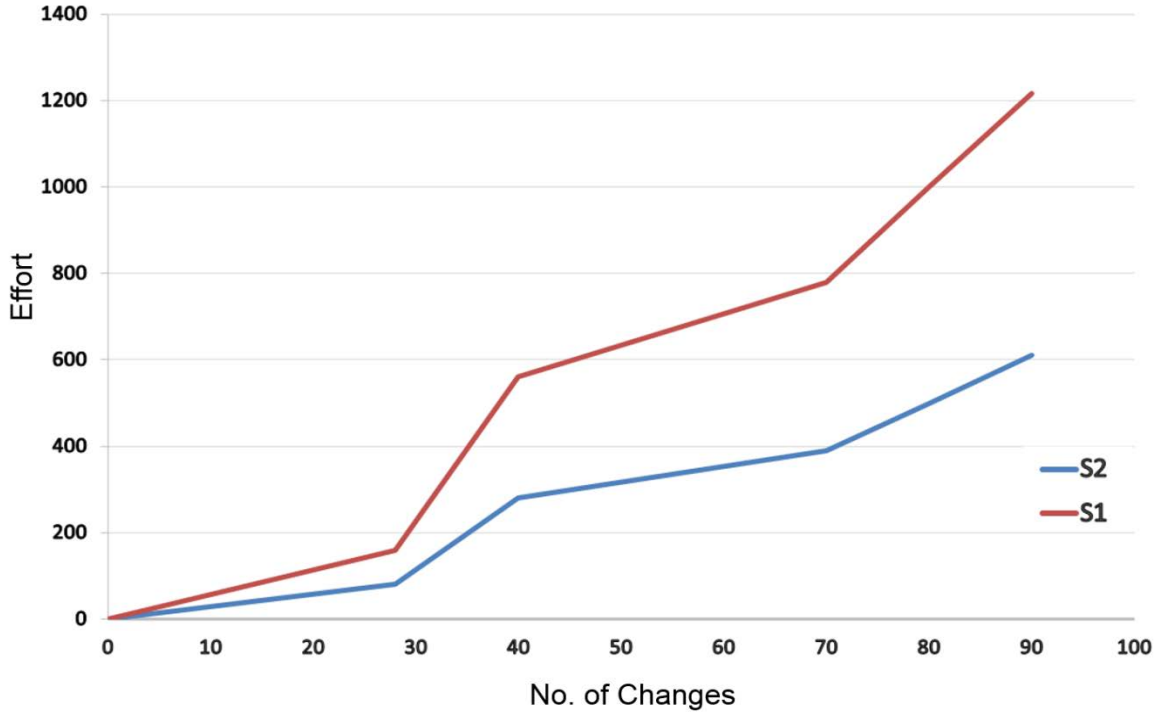


Figure 5.14 Development Effort

Development for S2 needed less effort because of the lack of quality and testing measurement activities, as illustrated in Figure 5.15, although deadline pressure was very high. S1 had lower deadline pressure, therefore all quality and test measurement activities were satisfied. The software developed in S1 was high quality, resulting in minimal effort needed for correction (52 hours). In contrast, the software developed in S2 needed 1123 hours of correction effort.

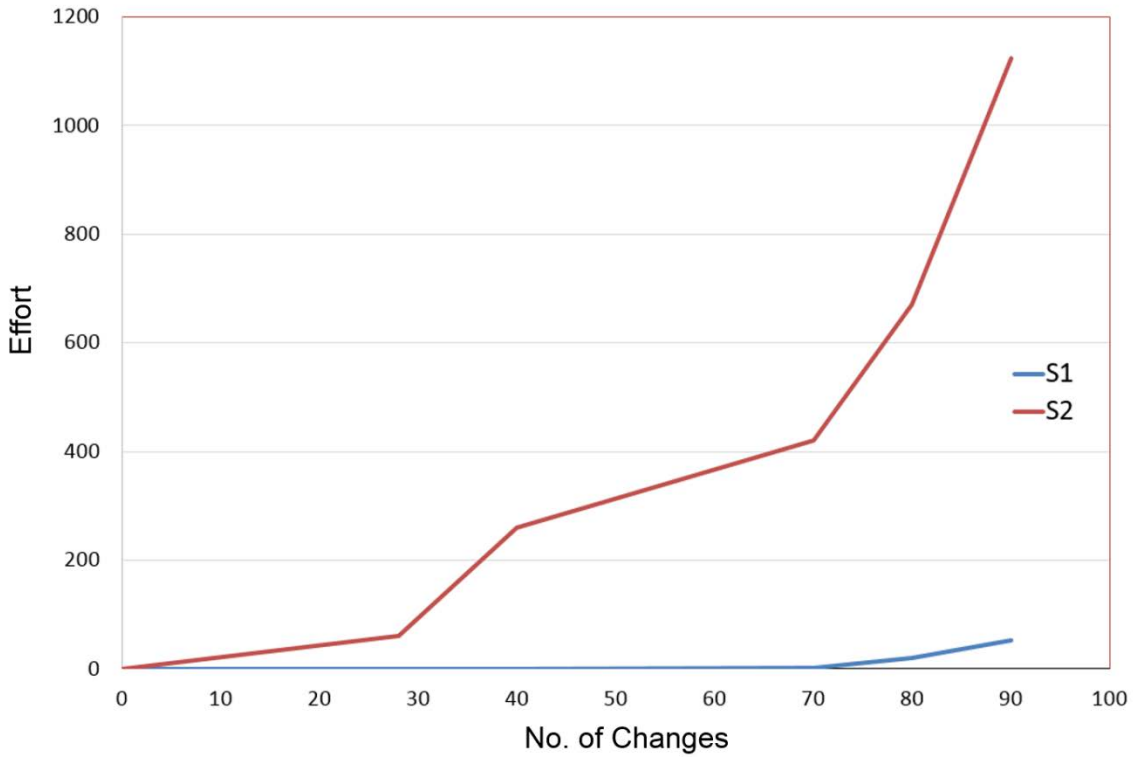


Figure 5.15 Defect Correction Effort

Table 5.8 illustrates the effort needed to get the product to the same quality level. In S1, the model estimated 1216 hours of development effort and 52 hours of correction effort, a total of 1268 hours. S2 needed an estimated 610 hours of development effort and 1123 hours of effort to correct failures and defects, a total of 1733 hours. S2 overran S1 by 447 hours or 25%.

Scenario	Dev. Effort	Corr. Effort	Overall effort
S1	1216	52	1268
S2	610	1123	1733

Table 5.8 Simulation Results

5.2 Defect Cost Flow System

These processes also have to be continuously improved [100]. Quality assurance (QA) is the most important activity in the process, which ensures the quality of the software. Improvement activities of the process lead to identifying the optimal QA effort to detect most software defects and failures. Software product development phases have to ensure the quality in each development phase. Detecting the defects in the same phase where the defects are originated is a goal of each QA. 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.

The initial problem analysis and definition illustrates different impacts of the defect on the effort needed for defect correction depending on the phase where defects are originated and detected. It has been recognized that the correction effort for defects that originated in older development phases than they were detected is higher than the effort needed to correct a defect detected in the same phase of the development lifecycle. Therefore, to reduce the effort of software correction, the different development phases with their defects that originated in the same phase or shifted from older phase were analyzed and assessed to meet all KPIs and

parameters that affected the cost of software over the development phases. The goal of defining these KPIs and parameters is to identify and improve the QA effort to improve the quality of the product with lowest costs.

The DCF helps to have a better estimation with higher accuracy for the effort needed to correct software. The DCF depends on KPIs that define the product, processes and project parameters. DCF helps to analyze the effort spends on software correction compared with the effort spent on software development throughout every phase of development lifecycle, which will help to define the QA needed in each phase.

5.2.1 Problem Definition

Continuous improvement of the development process helps to reduce the effort of the development needed to realize the requirement of the feature. Upon first glance, the QA seems to involve additional effort in addition to the development effort. However, reducing the effort of software correction leads to an increase in the defect rate in the software and it shifts this effort to a later phase of development lifecycle, which means more effort to correct these defects. The problem was having a system able to identify the important optimization of the development process, which led to reducing the effort needed for software correction and thereby the total development effort. The model should be able to identify the distribution effort of QA. Identifying the QA needed for the system leads to reducing the defects in software and minimizing the correction and rework activities needed to fix these defects. Reducing the correction activities means reducing the time and cost needed for the project, as well as reducing the total development effort. Data needed for the model was accumulated and analyzed to define the metrics and their KPIs, as well as the relationships among these KPIs. KPIs and the relationships among them were identified by using a list of questions accumulated and analyzed by using the QSM technique. The most important questions to explain were as follows:

- Development phases quality of development and improving the not necessary rework
- Effort needed for QA-measurements
- Effectivity of QA-measures
- The additional effort needed to correct a failure in later phases

The development effort needed include the effort needed for software correction too. The model deals with the effort needed for software development including the effort of defect correction and the QA needed to ensure the quality of the software and satisfying all of requirements demand.

5.2.2 Model definition

The automotive domain development lifecycle includes several phases and activities as described by using different framework and workflow techniques like waterfall and agile. Software failures and defects are flowed as shown in Figure 5.17. There are five phases of workflow used during the development lifecycle, which were built into the model of software failure effort flow based on the V-Model development process defined in an earlier section: Requirements Engineering (REQ), Design (DES), Sub-Design (SubDES), Implementation (IMP) and Verification & Validation (V&V). The effort needed for failure and defect correction for each phase is determined separately. Each phase of the development phases need to fix the failures generated during the same phase or the earlier phases. Therefore, several quality activities are needed to detect all failures and defects in the software. The ability to detect all failures in the

same phase will need a huge effort, whereby some defects are still hidden and shifted to the next phase of development.

Figure 6.1 illustrates the concept of flowing defects in development lifecycle. There are five development phases built into the model according to the V-Model development process definition in section 2.1.3: Requirements Engineering (REQ), Design (DES), Sub-Design (Sub-DES), Implementation (IMPL) and Validation and Verification (V&V) phases. For every phase, the effort of defect correction is determined separately based on process specific KPIs.

In phase REQ, there is only one possibility to have defective requirements represented by the node “SW Correction effort after QA”. The QA activities, e.g. requirement reviews, help to reduce this effort by detecting the defects and fixed them earlier. The remaining, undetected defects in REQ phase are handed over to the next phase “DES”.

In phase DE, there are two defect possibilities: one is originating in the last REQ phase, e.g. a defect requirement leads to a defect design artifacts. The second is the defect introduced in the DES phase itself represented by the node “SW Correction effort after QA”. REQ defects are adjusted by its phase multiplier. It represents the effort increasing to correct the defects in other phase 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 SubDES there are three different defect types. The first are defects with their origin in phase REQ, e.g. a defective requirement passed through the design process on which an SubDES is based. Second, there are DES defects, e.g. if a defective design is used as basis for a SubDES. Defects with their origin in phase REQ and DES phases are adjusted by their specific phase multipliers. And finally, there are defects originating from a defective SubDES. In this phase typical QA activities are design review reviews.

In phase IMPL there are four different defect types. The first are defects with their origin in phase REQ, e.g. a defective requirement passed through the design process on which an IMPL is based. Second, there are DES defects, e.g. if a defective design is used as basis for an implementation. Third, there are SubDES defects, e.g. if a defective IMPL is used as basis for an implementation. Defects with their origin in phase REQ, DES and SubDES 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 V&V, there are defects from REQ, DES, SubDES, IMPL and V&V itself. This phase represents final integration and test activities for development artifacts of all phases. This is the point where the software product is delivered to the customer. The resulting effort of defect correction after V&V phase represents rework for defects detected by the customer and thereby the additional effort needed to finish the project.

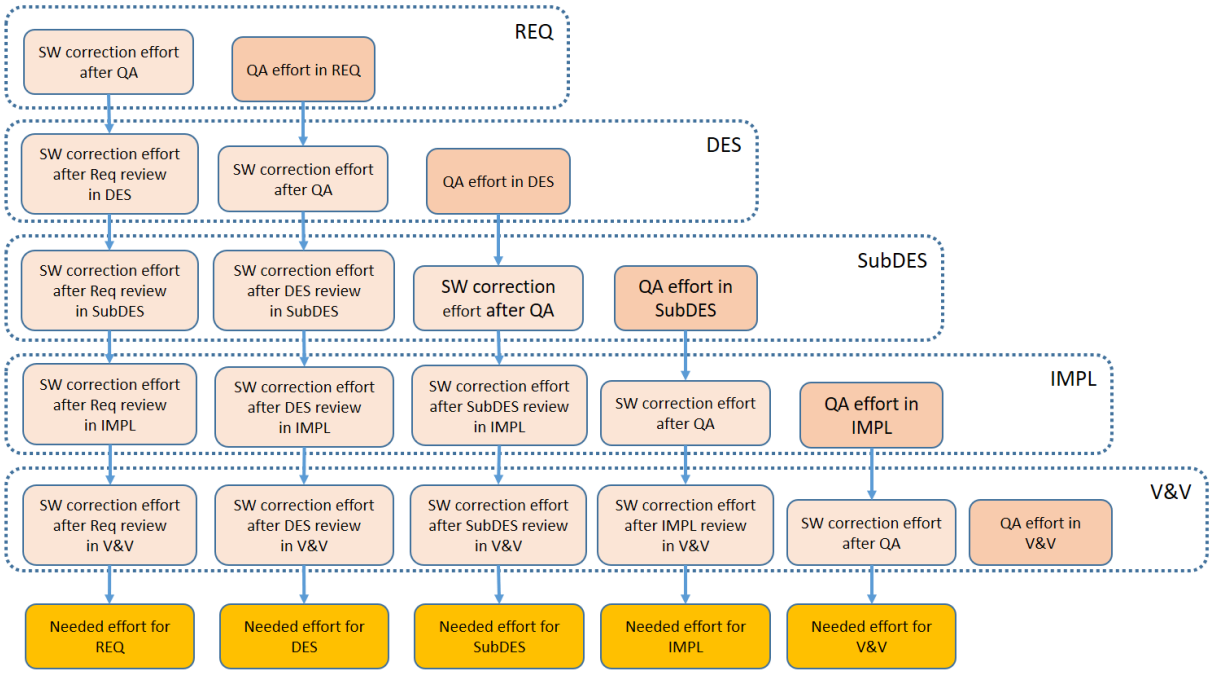


Figure 5.16 Development Phases

Figure 5.17 represents the effort needed to correct these failures and defects in software. The effort needed for correction is presented in the positive axis, whereas the negative axis illustrates the residual effort after the quality assurance measures that are shifted to the next phase.

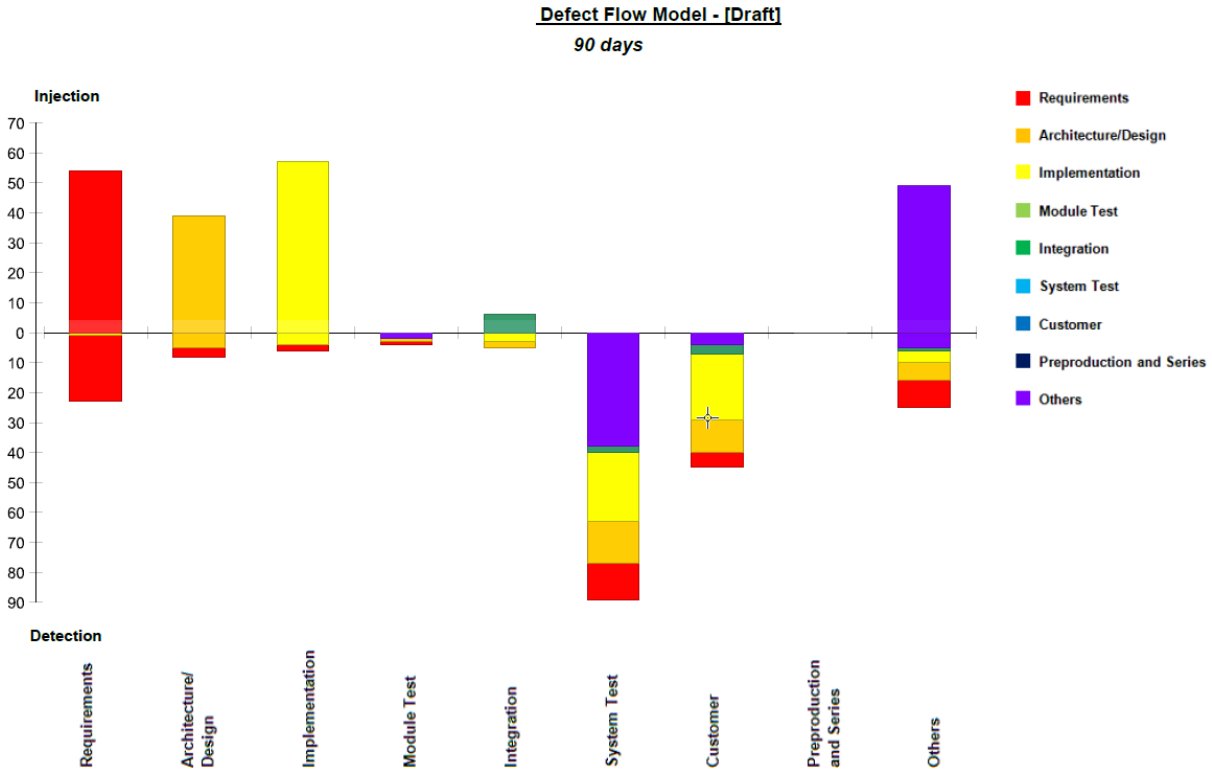


Figure 5.17 Distribution of Defect Originating and Detected Phases

Figure 5.18 shows 680 hours of potential correction effort still in the software. Through the review activities and other quality assurance activities, some failures could be fixed, whereby

with a high reduction rate of effort needed to correct the defects of 80%, the effort residing could be reduced to 544 hours. The total effort for REQ phase will be 136 hours. Some failures could not be detected, whereby the effort needed to correct this nun-detectable will be shifted to the next phase in the lifecycle, i.e. the DES phase. The shifted defect from REq to the DES phase will need more effort to be fixed, which means increasing the effort by a factor. The phase multiplier of the DES phase for the failures shifted from REQ to DES were calculated by calibrating the model and using experts' knowledge management to a value of 4; therefore, the effort needed to correct the shifted failure of 136 hours will raised to 455 hours. In the DES phase, some of the defects originated in the last phase or the REQ phase are fixed and corrected, whereby 80% of correction effort is needed and a correction effort of 413 hours still remains in product for this phase and be shifted to the next phase of development lifecycle or the SubDES phase, where the modules are built for the system. The activities generated in the REQ phase and shifted to DES and then to the SubDES phase will be multiplied by 5 as a calibrated factor for this phase. In the SubDES phase, 60% of these shifted efforts could be fixed, while the rest of the effort is multiplied by the calibrated factor and shifted to the IMPL phase. Some of these defects could be fixed, whereas the others will be shifted to the V&V phase. The correction effort will be multiplied by the calibrated factor again before they are shifted to TST phase. Any other defects found on the customer side will mean a significant effort to be fixed.

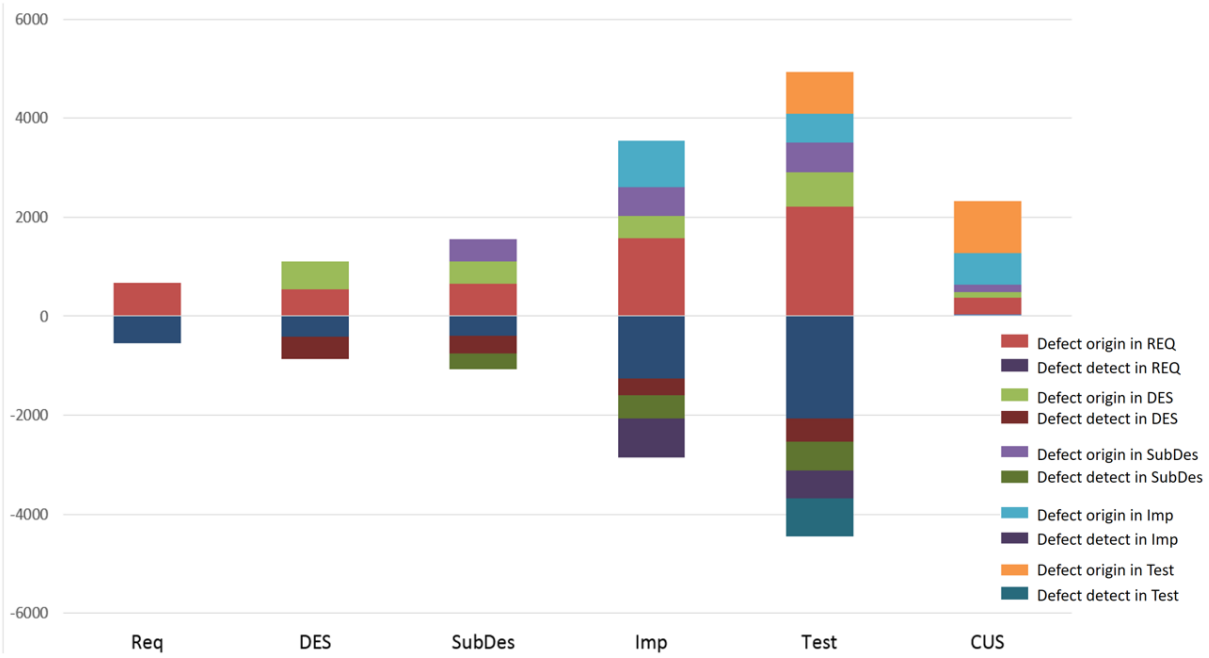


Figure 5.18 Software Correction Effort Distribution

5.2.3 KPIs identification

Different phases of the lifecycle need differing amounts of effort to be spent on their quality assurance measures, whereby each shift of effort from one phase to other later phases means increasing the total development effort. An ideal project estimation need to include the effort of QA and all failures and defects correction into the total development effort. Identifying the ideal amount of effort estimation including all needed activity efforts needs a system with a total understanding of the software and the traceability among their components, whereby such a system could help the project managers to make decisions. Different KPIs are needed to build an estimation system to deal with different possibilities of changes. The definition of the measurement categories was used to identify the PKIs needed for the system. The measurement category will be presented by a set of specific metrics. Obtaining these measurement

categories needs some clarification. Accordingly, a set of questions need to be answered to understand the problem statement and help to derive these measurement categories.

To identify the KPIs and define a characterization for the model, the answers to some questions are needed. Table 5.9 shows some question sets that enable us to identify all major components of the model.

Question	Metrics
Effort to correct the defect	M1: Defect effort-Factor for feature's error-proneness. M2: Potential effort of correction needed to correct the failures and defects.
effort of Quality assurance (QA)	M1: QA effort needed. M2: Effort needed for QA measures. M3: decrease the effort needed to correct problem based on the sufficiency of QA effort. M4: Remaining effort of correction. The Difference between effort needed to correct the defect before and after QA.
Effort raising because of defects shifting over phases	M1: development domain specific factor representing Additional rework if defects flow over development phases.
the overall effort	M1: Effort needed for artifact development in REQ, DES, SubDES, IMP and V&V excluding QA and correction effort. M2: total of development, QA, problem fixing

Table 5.9 QGM of Quality Assurance

5.2.4 Model Creation

A model offers help for project managers with their decision, although it requires including a lot of data and information about the project. This data has to be available in the model to ensure that it could help during different phases of the development lifecycle. Some necessary information will be identified by a set of data accumulated from the old projects by using the change request system, while some others will be identified by the expert knowledge and external data. This information and the answers to the aforementioned questions will help to build a model representing the KPIs of the system. This information will be accumulated and put into the matrix networks of the defect cost flow model. Depending on the problem definition, we will describe the KPIs and put these as nodes into a model including 75 nodes to represent the five phases of the development lifecycle: REQ, DES, SubDES, IMPL, and V&V.

Figure 5.19 illustrate the model structure analyzed and built, whereby some nodes were used as calibration nodes to set up the model. The model describes the different development phases of the software, with every phase including different nodes and their relations. The model describes the shifted defects from earlier phases to later phases in addition to the defects belonging to the same phase.

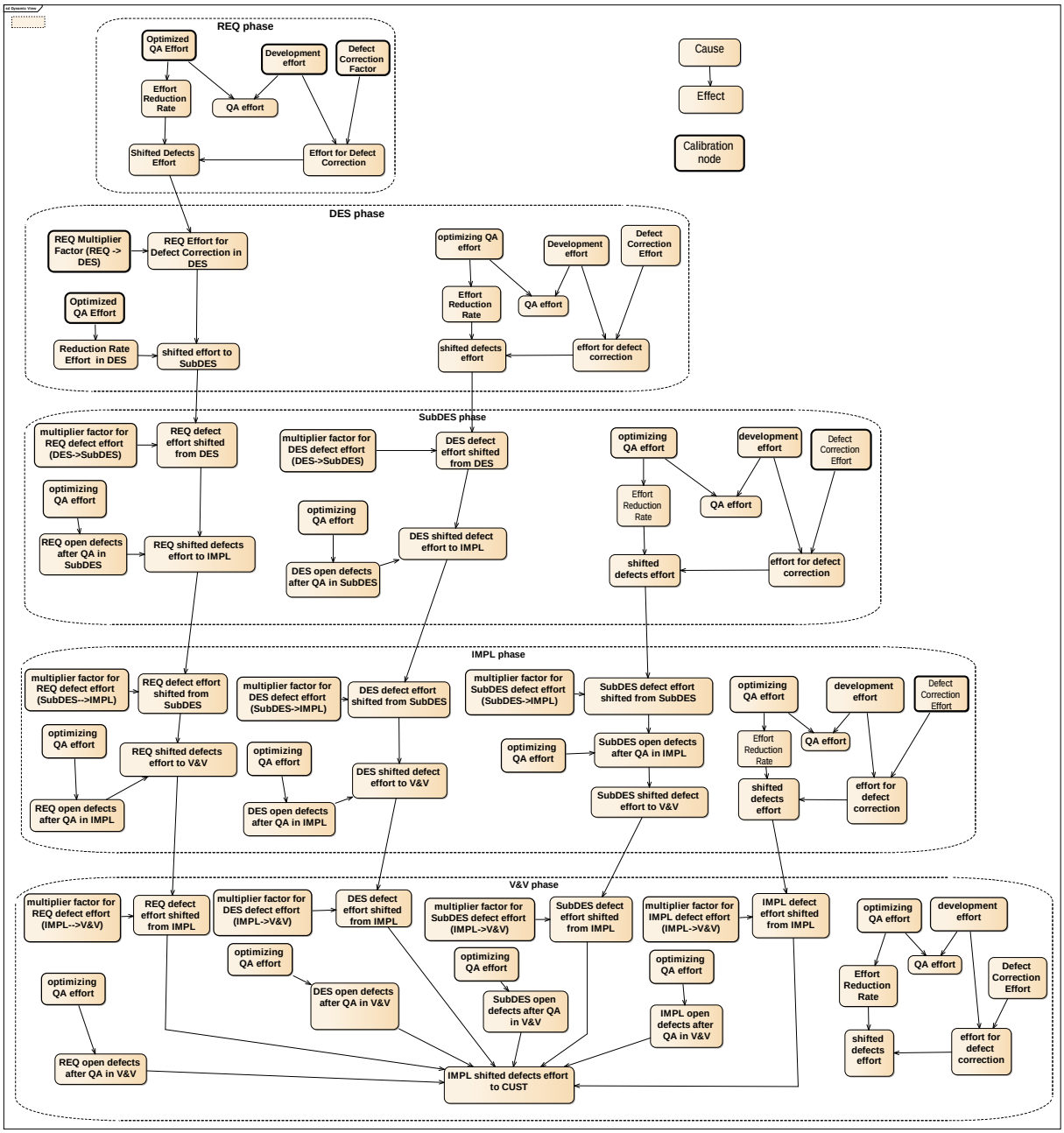


Figure 5.19 Model Represents the Different Phases of Software Development

In phase REQ shown in Figure 5.20: The development and QA activities have the effective effort in the phase. The QA goal reduce the defect possibility in the phase by the defect reduction rate. Other activities are needed to fix the defect detected by the QA activities which is an additional effort for the total development effort. Defects that not detected in this phase will shifted to the next phase (DES), the node “Shifted Defects Effort” represent the shifted effort from REQ phase to DES phase.

In phase DES shown in Figure 5.21: the shifted effort from last phase (REQ) have to multiplied by the phase multiplier to estimate the needed correction effort “REQ Effort for Defect Correction”. Additional effort are needed here to deduce the defect rate of the defects shifted from REQ phase. In this phase a defect rate will produced and need effort to fix them as a part of the development and QA effort in this phase.

In phases SubDES and IMPL: Additional effort needed because of shifted defects effort from REQ and DES into SubDES phase and shifted defects effort from REQ, DES and SubDES into IMPL phase.

The last phase V&V as shown in Figure 5.22: All shifted defects effort needed to accumulate in this phase. Any shifting of effort means release a defective software to customer which means a lot of effort and cost have to be lost. The shifted effort from REQ, DES, SubDES and IMPL phases multiplied by the specific multiplier factor have to be presented here. The calibration nodes in Figure 5.19 are used as input factors to improve the estimation accuracy.

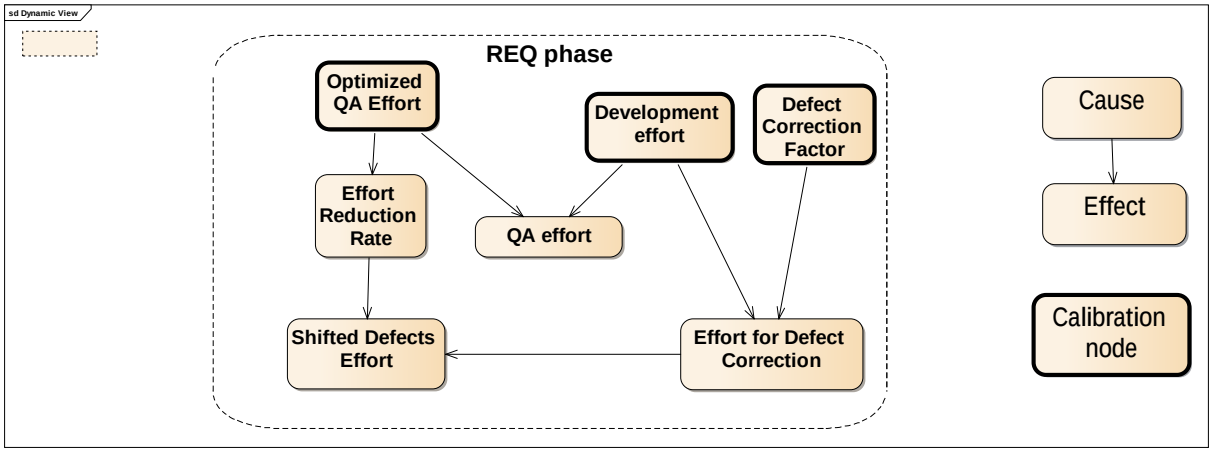


Figure 5.20 Model Represents the REQ Phase of Software Development

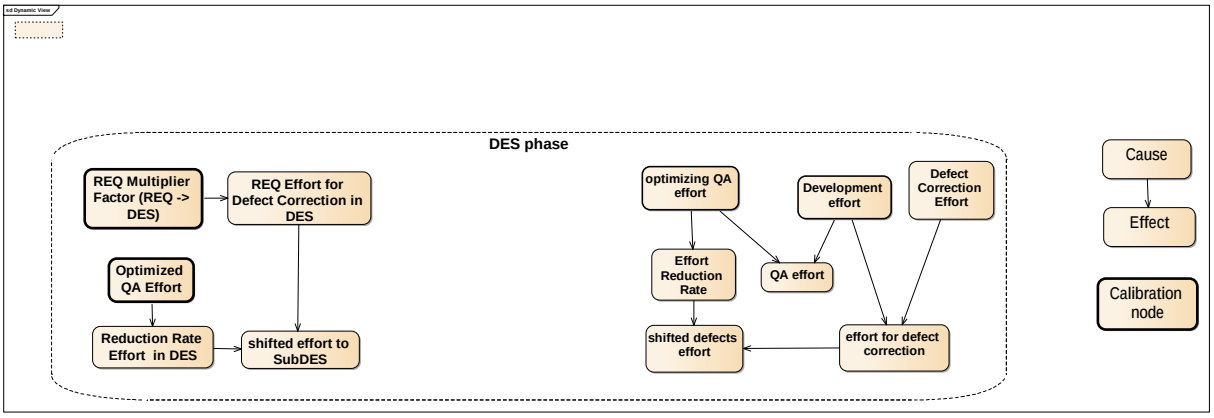


Figure 5.21 Model Represents the DES Phase of Software Development

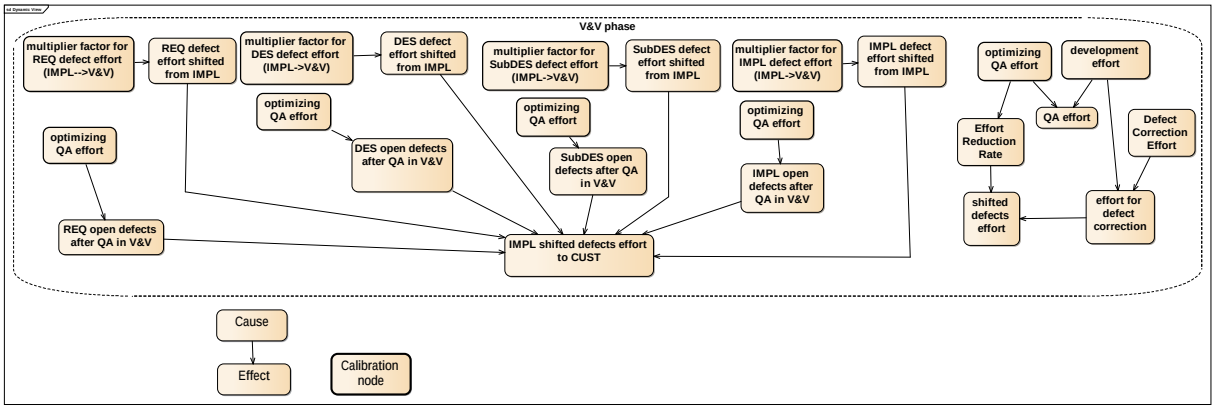


Figure 5.22 Model Represents the DES Phase of Software Development

5.2.5 Model Calibration

This section describes the setting up of calibration nodes and the relation among the nodes of the model.

5.2.5.1 Cost Factors of Defection

There are different phases in the development lifecycle. The correction effort needed for each phase is affected by a specific development effort and defect cost factor for the phase. The defect effort for the phase describes the possibility of the phase to insert a defect into the system, i.e. the defect cost factor for the V&V phase is very low compared with the REQ phase. A maximum sufficiency of quality assurance effort could help to get a phase with very low defect cost factor, which affects the other project parameters. Defect cost factor reduction leads to increasing the effort needed for QA activities. An initial defect rate in the model does not need a high QA effort. This factor is needed to calibrate the model depending on its phase.

Analyzing the internal data set shows the average values of these factors over the development phases. The defect cost factors for different features with different complexities are analyzed, showing that by increasing the complexity of the feature this will increase the defect cost factors. Table 5.10 illustrates the calculated values of the defect cost factor from different internal data sets.

REQ	DES	SubDES	IMPL	V&V
1	0.8	0.7	0.6	0.01

Table 5.10 Prediction Accuracy

5.2.5.2 Quality Assurance Activities

Every phase has own defect cost factor and defect correction effort, whereby the correction factor of the phase is presented in a node including the development effort node and defect cost factor node. The effectivity of the QA activities in the different phases of the development lifecycle could be measured by the nodes optimizing QA effort and the shifted effort from each phase to another. Defining the correction effort is enabled by the shifted effort value from one phase to the next, while the quality assurance activities lead to reduction rates resulting between 0% and 100%. In order to represent the cost factor of a defect, different internal data sets of projects and project managers' perspective were analyzed. The result of this analysis shows four levels ranking the quality effort needed for software correction and the total development effort depending on the all boundary conditions of the project. These four ranks are defined as follows:

- Low: worst-case scenario.
- Medium: average scenario.
- High: ideal conditions scenario.
- Very high: an optimal case scenario.

The review effort E_r for the different sufficiency of QA effort from low, medium, high to very high is shown in Table 5.11 as an average of the development effort E_d needed, e.g. an $E_r(E_d)$ of 10% of additional review effort needed for a planned development effort of 100 hours for a feature, represented by 10 hours.

Rank	$E_r(E_d)$
Low	5%
Medium	10%
High	20%
Very High	40%

Table 5.11 Sufficiency of Quality Assurance Effort

5.2.5.3 Effort Reduction Rates

The QA activities in different phases of development lifecycle lead to detecting most failures produced in the same phase and preventing shifting these defect efforts to next phases. The undetected failure will be shifted to next phases. Shifting a defect from an earlier phase to later phases causes higher effort to be corrected. Each phase of the development phases has different quality assurance activities to fix their failures and the failures shifted from earlier phases. Depending on the project deadline and some other project parameters, a rank of QA is set. There are different correction efforts in each phase of the software, whereby some could be fixed in the phase while some are shifted to other phases. The distribution of correction rate of the software was analyzed for different projects, with the percentage of this analysis illustrated in Figure 5.25. The DES row illustrates the effort reduction of defects shifted from the REQ phase and generated in the DES phase by using different QA activities rank. The SubDES row illustrates the effort reduction for the input nodes for this phase from the REQ and DES phases, as well as the effort needed for defects generated in SubDES phase. Similar to this row, other rows describe the reduction rate of effort needed for software correction by using different quality assurance activities.

Rank of QA effort	DES	SubDES		IMPL				V&V				
	RE	RE	DES	RE	DES	SubD	IMPL	RE	DES	SubD	IMPL	V&V
Low	10%	10%	5%	8%	5%	10%	10%	10%	5%	8%	8%	8%
Medium	55%	50%	25%	35%	35%	38%	55%	40%	38%	40%	50%	50%
High	70%	65%	35%	40%	40%	45%	70%	60%	50%	50%	70%	70%
Very High	90%	85%	50%	60%	50%	60%	80%	70%	65%	70%	80%	80%

Table 5.12 Effort Reduction Distribution Based on QA Effort Sufficiency

5.2.5.4 Development Phase Multipliers

The uncorrected defects in a phase are shifted to the next phases, as illustrated in Figure 5.19. The shifted effort from earlier phases to other phases need to be adjusted by the phase multiplier. Fixing such defects need an additional rework effort over two phases, e.g. a shifted defect from the REQ phase to DES phase needs to be corrected in REQ in addition to the effort needed for software correction in the DES phase. Therefore, a multiplier factor is needed to define this additional effort in the model, whereby these phase multipliers are calculated depending on the analyzed data set and experts' knowledge. Table 5.13 illustrates the phase multiplier for different projects analyzed in the automotive domain. The effect of the tools used, the process of development and other different boundary conditions for the project are taken into account by calculating these factors.

A phase multiplier of 4 is described to represent the effect of shifting effort needed to detect and correct a software problem from REQ to the DES phase, e.g. residing effort of 64 hours in the REQ phase and shifting it to the DES phase means rising effort of factor 4 to 256 hours will be needed to detect and correct these defects. Similar multiplier factors for the different phases are analyzed and could be used from the table.

RE → DES	DES → SubDES	SubDES →IMPL	IMPL → V&V
4	4	3	5

Table 5.13 Development Phase-Multipliers

A phase-multiplier of 4 is taken to represent the effect of shifting effort of detect correction from REQ to DES phase, e.g. residing effort of 64 hours in REQ phase and shift it to DES phase means rising effort of factor 4 to 256 hours will be needed to detect and correct this defects. Similar multiplier factors for the different phases are analyzed and could be used from the table.

5.2.6 Quality assurance effort

The quality assurance activities in each phase of the development lifecycle lead to detecting different defects and failures in the software that were generated in the same phase and prevent shifting defects to other next phases. In order to identify the distribution of these activities through the development lifecycle, a different project was analyzed with their different defects in the system that need 1,000 hours of effort to fix the failures, and the same defects were estimated as simulation for the model with different aspects of QA activities. The quality assurance activities include the activities in the REQ, DES, SubDES, IMP and V&V phases. The test cases analyzed had different complexity levels and different failure origin phases. Due to the time pressure of the project, some activities are done with low effort, which influences the quality of this phase. The cases analyzed have different efforts of these activities to present this effort and their effect on the total project effort. Table 5.14 illustrate the cases analyzed.

QA in Phase	Case1	Case2	Case3
REQ	Low	High	Very High
DES	Low	High	Very High
SubDES	Low	High	Very High
IMP	Low	High	Very High
V&V	Low	High	Very High

Table 5.14 Cases Overview

5.2.6.1 Simulation results

A high-quality software means that all required specifications have to be met with no defects or bugs included. To obtain this quality level, the software has to be developed with a multi-level process to manage all activities needed during the development lifecycle. Different tools are needed to track the development lifecycle. Different norms and standards should be satisfied during developing a software in this domain, as explained in chapter 2. These huge activities and the tools needed to deal with these activities lead to increasing the cost of software development.

One important point in the effort estimation is the cost flow of defects and the process to control this effort during the development period. The model was tested to describe this aspect, whereby some requirements of real projects were analyzed to classify the effort during different

phases of the development lifecycle, while some other requirements were set as test requirements and estimated by different software developers depending on the main phases of the development lifecycle. A lot of data set were analyzed, whereas some data did not include sufficient information needed for this verification. The data sets were filtered and only the change request requirements with sufficient information were analyzed, while some data set lacked some minor information to be completed and thus these were completed with experts' knowledge and be used for these tests. Three scenarios were used to study the effect of quality assurance activities on defect effort and the total development effort. The boundary condition of the software was set with different values to have a total overview over this point. The three test cases were set as explained in Table 5.14. The test cases deal with different requirements in terms of the effort needed in the project, whereby due to different project deadlines the developer analyzed the requirements in different efforts. This deadline pressure leads to including defects in the requirement, which is shifted to other phases that have the same deadline pressure, leading to shifting the defect to the next phases. Due to shifting the defect from one to another phase, some iterations were needed to fix these defects.

5.2.6.2 Test cases results

The not exact estimation of the effort leads to wrong plan of the project. A project deadline pressure will be very high because of this wrong estimation of effort. The effort needed for the development and the effort estimated for the software changes during development lifecycle have to be estimated with the influence of the QA activities that needed to reach the quality level required in the project. Some quality assurance activities couldn't be implemented and validated with satisficed effort that lead to shift some effort from phase to other during the software development lifecycle. The effort shifted from earlier phase to the next phase will be multiplied by a phase multiplier factor. The effort shifted will have arising effort in the new phase and leads to arise the total development cost.

- **Test case 1**

Case 1 described a worst case of development quality with high deadline pressure, whereby most quality assurance activities are undertaken in rash way and a lot of effort is shifted from one phase to another.

Figure 5.23 illustrates the result of analyzing case 1. The effort spent for the requirement of 30 hours rather than the estimated effort of 50 hours did not detect all defects. The remaining necessary effort was shifted to the design phase. Shifted defects from the REQ to DES phase lead to new defects in the DES phase. The effort spent in the DES phase was insufficient to fix all defects in the DES phase and was unable to correct the shifted defects from the earlier REQ phase. The total effort needed in the DES phase would be 80 hours for the shifted effort from REQ phase, and 35 hours for the defects generated in DES phase itself. The effort spent in the DES phase could not correct all defects and some of the effort needed to be shifted to the next SubDES phase.

SubDES phase had two shifted kinds of effort, which were multiplied with phase multiplier factor: 80 hours for the effort shifted from REQ phase and 32 hours as effort shifted from DES phase. The SubDES phase generated by itself some additional effort for defects correction. Some of the effort would be shifted to next implementation phase, whereby these shifted efforts represent the effort shifted from the REQ, DES and SubDES phases. The IMPL phase devoted more effort to the effort shifted from earlier phases and for its own effort. Some effort is be worked around in the phase, while some is shifted to the next test phase. The V&V phase has different input effort from earlier phases and some effort is needed for itself. Huge effort would be needed in this phase due to shifted effort multiplied by the phase factors. A lot of effort is

spent in this phase, whereby to correct defects shifted a lot of effort is needed for several iterations. Some efforts would be shifted again with a release of software to be detected by the customer phase, with these efforts reflecting a huge effort and being dependent on the release type.

The small effort spent in the earlier phases means shifting effort to the next phases, which leads to having a huge effort through every shift from an earlier phase to other later phases.

Figure 5.23 Effort Distribution for Case 1

- **Test case 2**

Case 2 described a widely used case of development and change request effort with medium deadline pressure, whereby most of quality assurance activities be undertaken with high aspect, although some of this effort is shifted from one phase to another.

Figure 5.24 illustrates the result of analyzing case 2. The effort spent for the requirement of 37 hours rather than the estimated effort of 50 hours could not detect all defects. The rest of the remaining effort was shifted to the design phase. Shifted effort from the REQ to DES phase leads to new defects in the DES phase. Effort spent in the DES phase was insufficient to fix all defects in the DES phase and was unable to correct the shifted defects from the earlier REQ phase. The total effort needed in the DES phase would be 48 hours for the shifted effort from the REQ phase, and 35 hours for the defects generated in the DES phase itself. The effort spent in DES phase could not correct all defects and some of the effort needed would be shifted to the next SubDES phase.

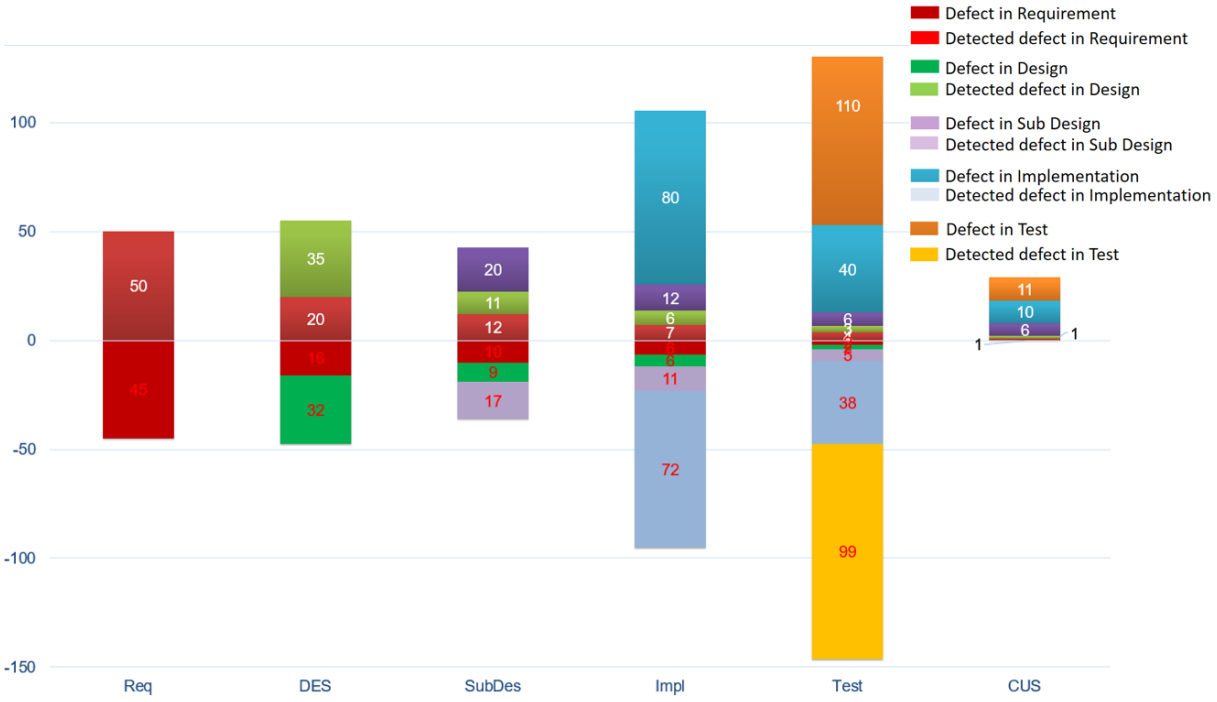


Figure 5.24 Effort Distribution for Case 2

- **Test case 3**

In this case, the optimum effort needed for quality assurance activities and other needed efforts were satisfied. Most defects were detected locally in each phase of development. Although a major effort was required in each phase, there is some effort shifted from one phase to others. Some points need a lot of clarification to be fully understandable, where such small defects could be shifted from one to other phase as a hidden effort. In the new phase and owing to the sufficient time to satisfy most quality assurance, most effort is fixed in the second phase and was not shifted to several phases that the effort increased in each new phase because of the phase multiplier factors.

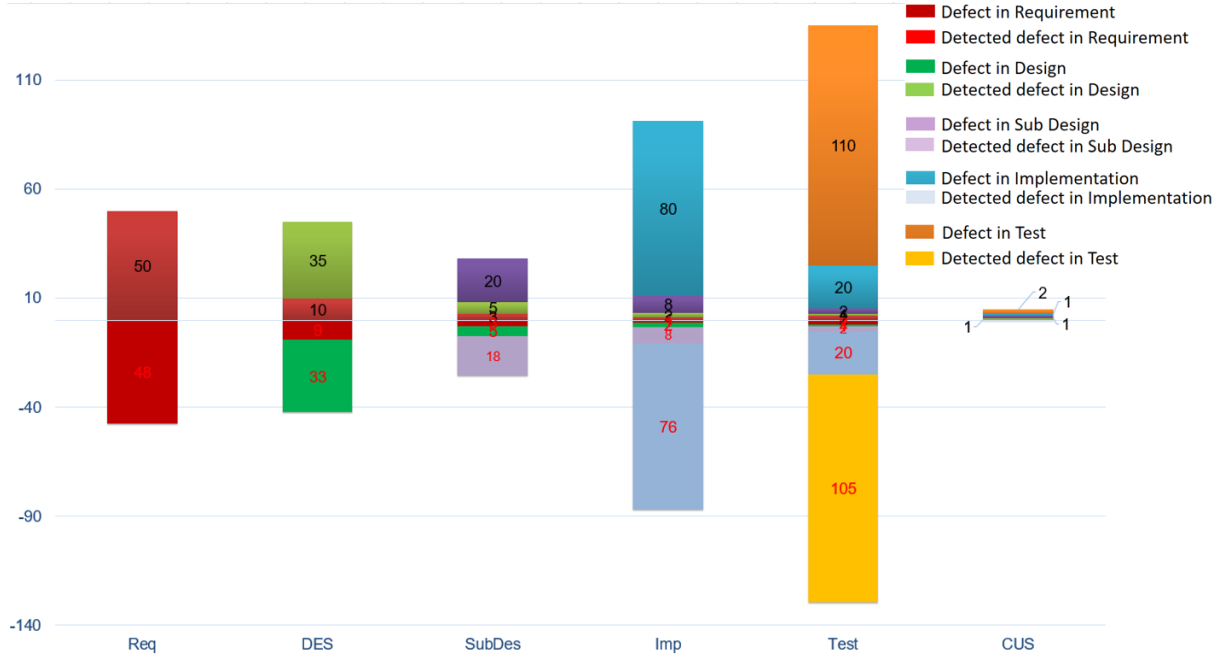


Figure 5.25 Effort Distribution for Case 3

Figure 5.25 illustrates the effort spent in the REQ phase of 45 hours that helps to detect most defect possibilities. Only 5 hours were shifted to the DES phase. In the DES phase, there was sufficient time taken for the DES effort needed and to complete the missing effort in the REQ phase. The shifted effort to the SubDES phase was very small compared with the other two test cases. The same situation could be recognized in the other phases, and it is important to recognize that the effort shifted to the customer phase was very small and could be fixed in the next releases.

5.2.6.3 Conclusion

The test cases were undertaken using the development process clarified earlier and some experts' knowledge available in the project. The cases show the effect of the quality assurance on the development effort. It illustrates that planning the optimum effort for QA activities in each phase leads to reducing the total effort of the development.

Case 1 scenario was selected to describe the worst-case scenario in effort distribution overall regarding the different phases of development of a specific feature. Case 1 describes the consequences on saving QA activity efforts in each phase and focuses only on feature development. The QA activities were reduced to the minimum, which led reducing the defect correction rate and shifting the effort needed to detect the defects and failure in software to the next phase. This shifting of effort from the earlier phase of the development leads to increasing the overall development effort and causing a shift in a lot of effort to the customer phase, which means a huge effort in the next releases.

Case 2 indicates the case scenario where more effort is required during each development phase. These efforts lead to having a stable development and preventing a huge defects rate in the phase. The effort spent in this case was sufficient to detect most defect possibilities on the product, although many defect possibilities were still included in the software. This needed effort was shifted from earlier phases to other phases. Shifting effort from an earlier phase to another one leads to increasing the overall effort depending on the phase multiplier factor of the phase. Case 2 illustrates that some of the effort is shifted to the customer phase, which means a huge effort in the next release of the product.

Case 3 indicates the case with the best scenario. A lot of effort was spent in this case. This effort leads to increasing the defect correction rate and reducing the effort shifting rate from earlier phases to other phases. There were some hidden efforts that could not be satisfied in the local phase and were shifted to other one. Developing software with a zero-defect rate during the development phases needs a huge effort and despite having this effort there is a possibility to get some hidden defect that were detected in the next phases by having sufficient effort to ensure all phases during the development lifecycle. Case 3 illustrates that a high effort of QA activities leads to reduce shifting effort to the customer phase. In this test, it was a point that needed some clarification with the customer; therefore, this effort was shifted to the next release of the software.

- **REQ phase**

Figure 5.26 represents the effort spent in the REQ phase during the development lifecycle. The three requirements deal with different features, each of which needed similar effort compared with the other two features. These efforts were set by the experts and the project manager's estimation that they were developed in similar old projects. The project parameters were changed and the deadline pressure and information flow and clarifications were undertaken with different effort. This project pressures lead to shifting some effort from the REQ phase to the DES phase. The phase multiplayer factor was used to define the new effort needed to

close the effort shifted from the REQ phase. The spent requirements efforts for the three cases were 30, 38 and 45 hours for case 1, 2 and 3, respectively. The shifted requirement efforts were 20, 12, 5 hours for cases 1, 2, and 3, respectively. In the DES phase, these efforts were represented after being multiplied them by the DES multiplier factor as 80, 48, and 20 hours. Due to the same project conditions, the effort spent in DES was unable to fix all shifted effort from the REQ phase. The effort spent in the DES phase to complete the effort shifted from REQ (80, 48, and 20 hours) was 53, 40, 18 hours in case 1, 2 and 3, respectively. The QA activities in DES such as reviews and DRBFM were undertaken in different levels, whereby these QA activities were not insufficient to fix all missing efforts and some of the already-shifted effort from REQ phase to DES phase was shifted again to the next SubDES phase. The shifted effort from the REQ phase to DES phase led to generating some defects in the DES phase, and due to weak QA in this phase these defect efforts were also shifted to the next phases. This shift of effort from the earlier to next phases also led to shifting a lot of effort to the CUS phase. Table 5.15 represents the REQ effort shifted from REQ to other phases.

Figure 5.26 illustrates that shifting effort from an earlier phase due to deadline pressure leads to a huge effort in the next effort. Shifting effort causes iteration of the development phases and increases the defects rate, which means more effort to complete the product. Case 1 needs to spend 236 hours in the V&V phase to detect all defects shifted from earlier phases, while case 3 needs only 2 hours of effort to detect the shifted effort from REQ phase.

The quality assurance effort depends on the complexity of the software features and the interfaces among them. These complexities are an important factor for the cost of development and correction of the software product. The QA activities needed for different features depend on these complexities. Thus, the sufficiency of QA effort for each development phase during the development lifecycle depends on the process activities used.

Figure 5.27 describes the total effort needed to complete the REQ phase. The effort shifted from the REQ to DES, SubDES, IMPL and V&V phases due to different engineering artifacts in the project lead to an increasing effort required in each phase. The overall effort of the development needed would increase in value depending on the QA activities in each phase. The shifted effort of REQ leads to generating more defects in the different phases owing to the requirements misunderstanding that includes defects in the design and implementation of the software. A huge effort in the V&V phase was needed to correct all defects generated due to the requirement shifting effort. Detected defects in the last phase of development need to have a new iteration of most development phases to fix all hidden defects and ensure the quality of the software. The effort devoted in the REQ phase leads to reducing the defect rate of the software and the overall effort of the development. Case 1 devoted little effort to the requirement phase due to the project deadline pressure, whereby some QA activities were not accrued or planned with very low quality. This leads to including different defects in the product and shifting these defects from one phase to another. The overall requirement development effort is very high for case 1 compared with case 3. Case 3 devoted some more effort in the REQ phase and did not shift a lot of effort to the next phases.

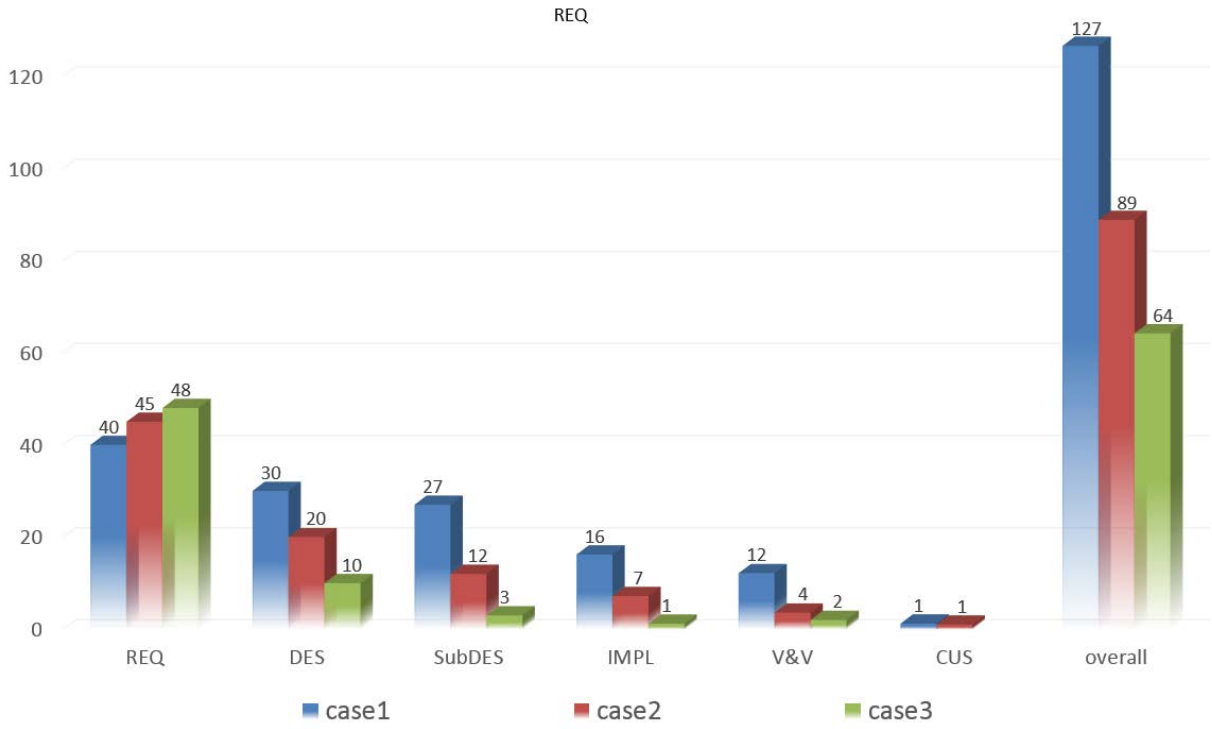


Figure 5.26 Effort Distribution for Different Scenarios of REQ Phase

	REQ	DES	Sub-DES	IMPL	V&V	Cus	Total effort
Case1	40	30	27	16	12	1	127
Case2	45	20	12	7	4	1	89
Case3	48	10	3	1	2	0	64

Table 5.15 REQ Effort Distribution into Development Phases

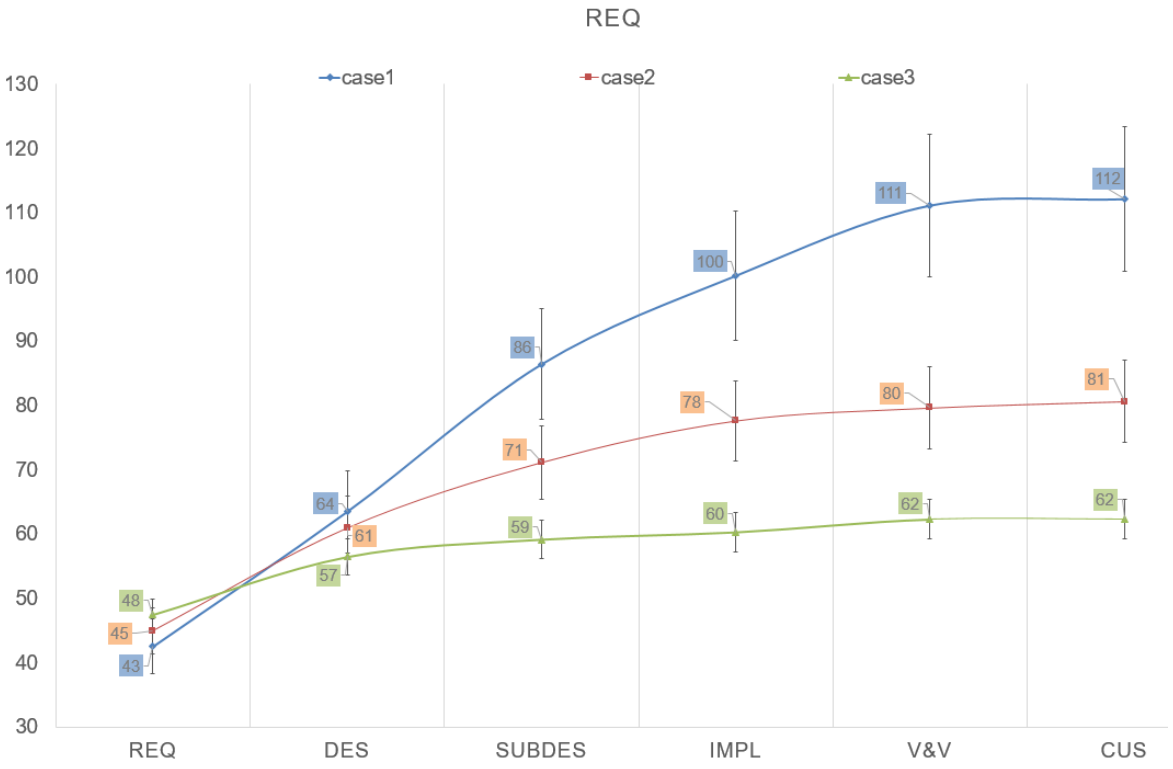


Figure 5.27 Overall Development Requirement Effort

- **DES phase**

The different QA in the REQ phase led to shifting effort from the REQ to DES phase of 20, 12, 5 hours for case 1, 2 and 3, respectively, whereby these efforts needed additional effort in the DES phase to be completed. The different project pressure parameters also caused having a weak QA in DES phase for case 1, a high QA for case 2 and a very high QA for case 3. Most shifted effort could not be completed for case 1 and thus much of this effort was shifted to the next SubDES phase. Case 2 could fix more shifted effort and case 3 could complete most of this shifted effort.

In the DES phase, there were some new defects and efforts that could not be completed in this phase itself. Case 1 spent only 24 hours and shifted some effort to the SubDES phase. Case 2 could spend 28 hours with better quality assurance, and case 3 spent 32 hours to complete the phase with the best quality and prevent shifting huge effort to the SubDES phase.

Figure 5.28 illustrates the distribution of the effort spent in the DES phase and the shifted effort of generated defects in the DES phase, as well as the effort needed to detect and correct these defects in the next phases of the development lifecycle. The effort needed to fix the shifted effort from the DES phase increased through the phases due to the phase multiplier factors; for instance, in the V&V phase, the effort needed to fix the DES shifted effort 52, 21, and 2 hours for case 1, 2 and 3, respectively. Shifting small effort from earlier effort through saving some effort there leads to spending huge effort in later phases, which means increasing the overall development effort.

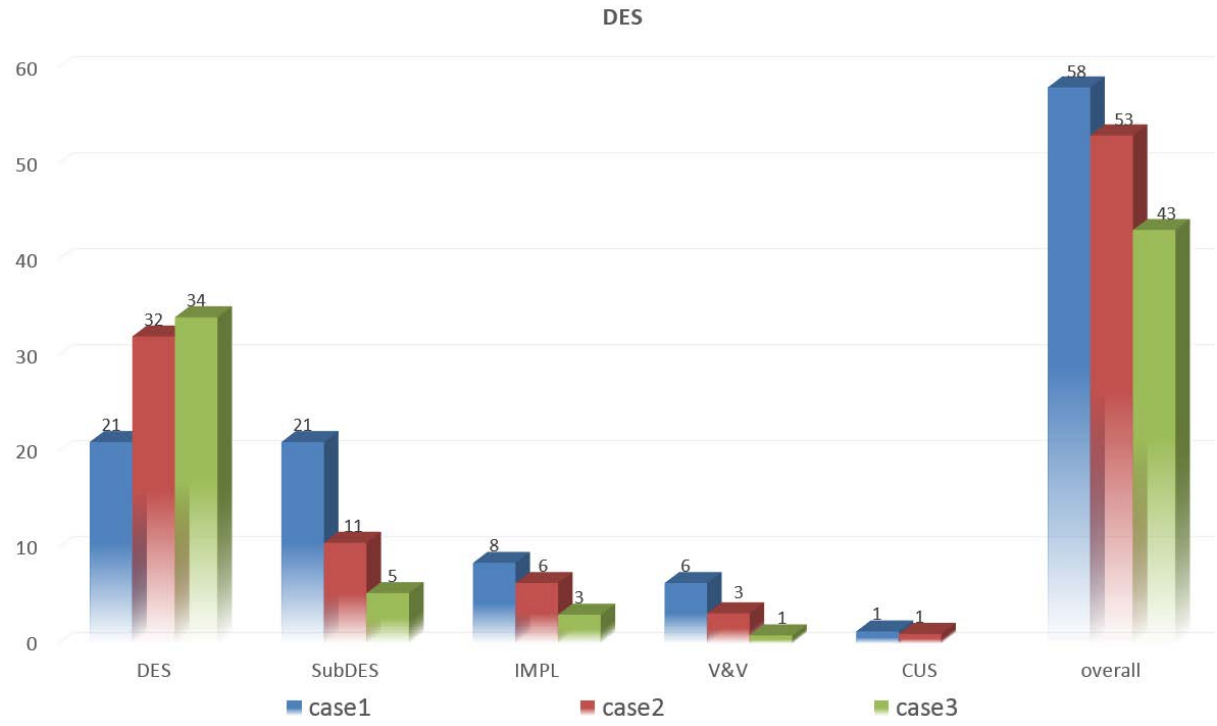


Figure 5.28 DES Effort Distribution through Development Phases

Figure 5.24

- **IMPL phase**

Quality assurance of the implementation phase of development is an important activity. The minor effort spent in QA activities in the IMPL phase leads to shifting some effort to the next V&V phase, which means increasing the effort by a phase multiplier factor. This weak QA level leads to increasing the defect rate in the software and the overall cost of the development and change request of the software. Figure 5.29 illustrates the effect of shifting effort from the IMPL phase to the V&V phase and the CUS phase.

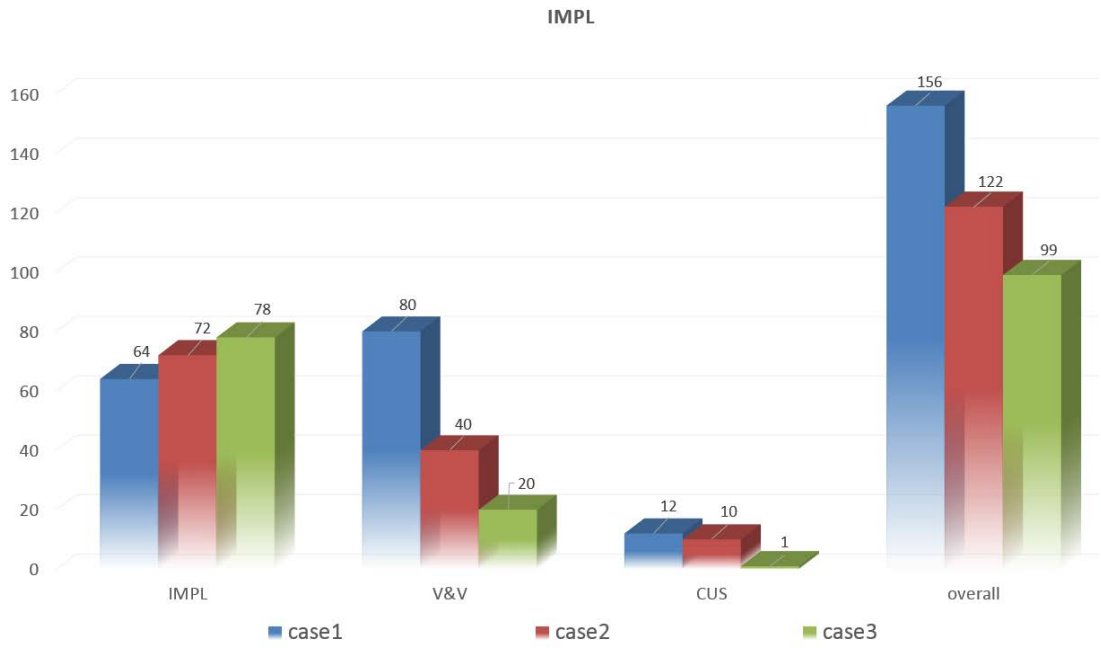


Figure 5.29 Implementation Effort Distribution over the Development Phases

- **V&V phase**

The last phase of the lifecycle before releasing the software is to validate and verify the product. Most shifted effects from earlier phases should be detected in this phase. This phase takes place in software projects with a high priority to ensure the quality of the product. The effort needed for this phase depends on the project engineering condition. The quality level of the product leads to planning a specific effort for this phase.

Figure 5.30 illustrates the effort planned for case 1, 2 and 3 as 330, 208 and 105 hours, respectively. Due to the different deadline pressure of the project, different effort is devoted during the last development phases. Case 1 had weak QA activities, which led to shifting a lot of effort from earlier phases, leading to some defects in the next phases of the development lifecycle. The V&V effort needed for case 1 was 482 hours, 46% more effort than planned. Case 2 needed 228 hours to complete the V&V phase, which represents ~10% more effort than planned. Case 3 shows a best estimation effort whereby the phase needed 4% effort less than planned.

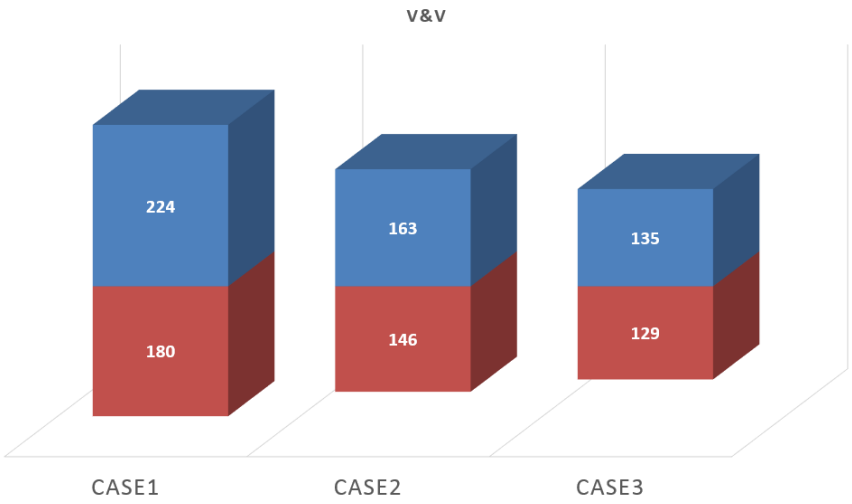


Figure 5.30 V&V Effort Distribution

The earlier QA effort saves the effort in the last phase of development. A defect correction for late detection leads to increasing the effort to a high level.

The software product under development phases will have complex KPIs with different relationships and many QA activities, which increase the difficulty of describing the effect of all of these parameters in the product effort. This increases the challenge for the project managers to take their decision. The effort flow of the defects model helps the project managers to have a sufficient overview and monitor the status of a project to estimate the project state under the different situations. Moreover, the model could support a deep knowledge for the system and the process. This knowledge and overview of the whole system help to improve the process maturity level. The model offers good help for Project manager to plan a better software project in terms of time, quality and cost.

6 Practical result

This chapter describes the use of the suggested models in this thesis in a real-life environment. A Prototype for the models were developed and tested. The evaluation criteria are based on existing studies [91] [92] and the available experience in the company with resource estimation methods as described in 3.4. Three categories of evaluation criteria were used [70]:

- Model and Estimate Criteria.
- Estimation Method Criteria.
- Application Criteria.

6.1 Prototype system

To evaluate and validate the models and based on the domain specification a first prototype model has been created. Several changes in software have been estimated based on these models. The prototype model be tested and iteratively improved as in Figures 6.1 and 6.2.

	Assumptions on which the expert estimation is based:				
	Cost drivers:	Acronym:	Rating:	Cost factor:	Rating description (updates automatically):
1	Required reliability	RELY	Normal		easily recoverable losses
2	Database size	DATA	Normal		
3	Product complexity	CPLX	Normal		detailed selection of complexity
	Control Operations	CPLX_1	Normal		ple nesting. Some intermodule control. Decision tab
	Computational Operations	CPLX_2	Normal		dard math and statistical routines. Basic matrix/vect
	Device-dependent Operations	CPLX_3	Normal		ing includes device selection, status checking and e
	Data Management Operations	CPLX_4	Normal		put and single file output. Simple structural change
	User Interface Management Operations	CPLX_5	Normal		of widget set.
4	Development for reuse	RUSE	Normal	1,00	across project
5	Documentation match to life-cycle needs	DOCU	Normal	1,00	Right-sized to life-cycle needs
6	Execution time constraint	TIME	Normal	1,00	Use of 50% of available execution time
7	Main storage constraint	STOR	Normal	1,00	Use of 50% of available storage
8	Platform volatility	PVOL	Normal	1,00	major change: 6 month; minor change 2 weeks
9	Analyst capability	ACAP	Normal	1,00	55th percentile
10	Programmer capability	PCAP	Normal	1,00	55th percentile
11	Personnel continuity	PCON	Normal	1,00	12% per year
12	Application experience	APEX	**High**	**0,88**	3 years
13	Platform experience	PLEX	**High**	**0,91**	3 years
14	Language & tool experience	LTEX	Normal	1,00	1 year
15	Use of software tools	TOOL	Normal	1,00	basic lifecycle tools, moderately integrated
16	Multi-site development	SITE	Normal	1,00	Narrowband email
17	Required development schedule	SCED	Normal	1,00	100% of nominal
	Total cost factor:			**0,80**	**80% of nominal**

Description
This cost driver accounts for the additional effort needed to construct components intended for reuse on the current or future projects. This effort is consumed with creating more generic design of software, etc.

Low
Normal
High
Very High
Extreme High

Figure 6.1 Implemented Tool of the Estimation Model

	Company and project characteristics				
	Scale drivers:	Acronym:	Rating:	Value:	Rating description (updates automatically):
1	Precedentedness	PREC	Low	4,96	largely unprecedented
2	Development flexibility	FLEX	High	2,03	general conformity
3	Architecture/risk resolution	RESL	Normal	4,24	often (60%)
4	Team cohesion	TEAM	Very High	1,10	highly cooperative
5	Process maturity	PMAT	High	3,12	CMMI Level 3 (Defined)
	Sum of scale drivers:			15,45	(SF_SUM)

	Estimated development time		
	Effort formula		Duration = C * Effort_PM^F
	C (Calibration factor)	3,67	Default is 3,67
	Avg. hours / month	152	Default is 152
	Effort [Hours]	23.001	W/o Requirements & Management efforts; Ignoring SCED Cost Driver
	Effort [Person months]	151	Person months (12,6 years)
	F (Math. Power)	0,311	F = 0,28 + 0,2 * 0,01 * SF_SUM
	Nominal development time	17,5	Calculated by normal effort formula
	Avg. Staff	8,6	Persons
	Nominal development time:		17,5 Person months

Figure 6.2 Implemented Tool of the Estimation Model

6.2 Developing Knowledge

Software development needs specific knowledge in different domains. A model or system with the ability to offer help for project manages in software effort prediction is a major challenging and needs a lot of experts' knowledge. The process of development and effort distribution through the multi-level of development needs to be identified and analyzed to identify the important KPIs of software and the development process. These KPIs and their effect of the system and how could they generate a defect are necessary for the model. This information needs to be classified and described using a modeling technique to present an overview of the system and the interfaces among the system components.

The previous chapters described a model for the software process, in which the main KPIs of the software process and their interfaces were identified. A model describing the software cost flow was build, whereby mainly the KPIs and QA activities of the development lifecycle were described. This knowledge and some other experts' knowledge about software development was accumulated and presented as graphic matrix for a development system. Several change requests were set and analyzed in different ways to present the advantage of these models. The knowledge needed to build the model should be sufficient to identify three development areas:

6.2.1 Development area:

Software engineering knowledge is needed to describe the domain of the model in general. A deeply understanding for the system complexity and the defects possibilities and their KPIs to define the software successful or failure. Software correction and defect preventing by using different QA activities knowledge are to be available in order to have such a model.

6.2.2 Process area

Software development need for a process and in automotive domain there are different process need to be understandable in order to develop a safety critical product. A knowledge about several standards and norms should be available.

6.3 Model usage

The models used in different projects to describe the usability and the advantage of their use. Most experiments and tests were conducted in five different projects for hundreds of change request. Two of the projects were already developed, whereby the model was used to document the development result and plan the further change requests. Two other projects were started with the model based on the knowledge of the last projects. One project was conducted from start of the project, which has some similarities with other project, in which the project was developed and managed by different company and different project managers. The knowledge of the model was transferred by some trainings and meetings. The results of these different projects are similar and do not include a major challenging in use. Therefore, there was no need to classify the result into projects teams.

The traceability matrix is the important attribute of the model. A traceability matrix for each project was set. The interface among the system components was described in the specific matrix. The traceability was conducted with the help of UML [102] and enterprise architect [11] as a design tool. Some of the projects developed some Excel macros to present the traceability for their use. The traceability matrix described the most commonly-used KPIs of the components and represented the effect of the interfaces belonging to these KPIs. This matrix was updated in different iterations depending on the change of the KPIs. The traceability among the system components are very important to describe the effective change on each component and accumulate the total effort needed for each change request.

The change requests were classified as change requests and defect requests. Change requests deals with the change needed in the system due to new customer requirements, whereas defects deals with software failures and bugs that need to be fixed in the product. The effort needed for these requests were described in earlier chapters; therefore, this chapter did not classified the request as a customer change or defect. The effort for each change was set and analyzed with no dependency on the reason of the request. A change request system was used to track the request. An estimation for each request was undertaken by the experts on the specific team. Parallel to the request system, an effort estimation for each change request was conducted by the project managers using the model described. Some experts' knowledge were needed although by using the model due to some change complexity. The change request system was set in a way that the developer should estimate each level of development separately, i.e. requirement, design, implementation and test estimations were needed. That leads to have the estimation and the real developing effort of each development level. The experts were able to estimate the requirement and then started to describe the features of the requirement with all necessary QA activities, before starting with the design level. The design should be estimated and designed with all necessary QA activities, before the implementation level is started. Estimating and implementing the code with all QA activities should be closed before starting with the test level. These requests were very important to describe the exact effort needed for each development phase and each quality assurance activity.

6.4 Defect source analysis

The defects overall the development lifecycle for different projects were analyzed. It was able to classify the defects into two defect data sets based on the defect detection phase. The defects generated in the requirement phase could be fixed locally in the phase by using the review activities as a possible QA activity, while some others were shifted to next phases and could be detected there.

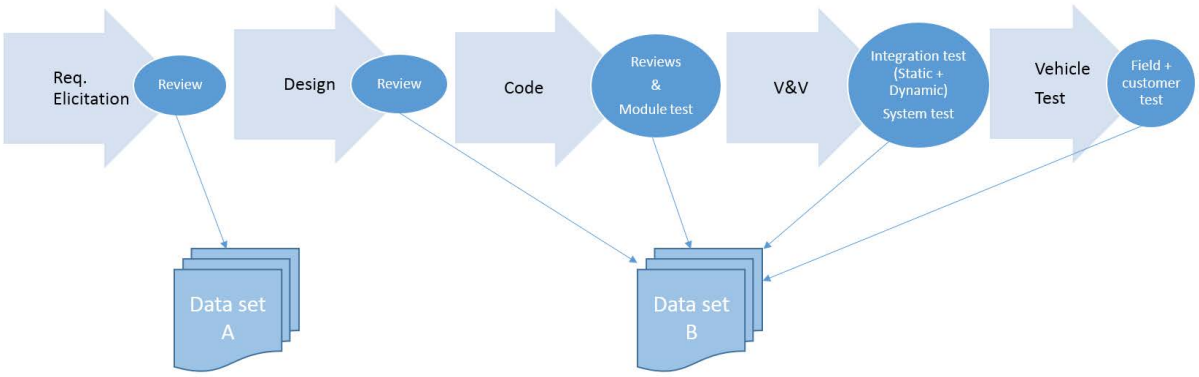


Figure 6.3 Defect Data Sets used for the Analysis of Requirement Defects

Figure 6.3 illustrates the defect data sets for defects generated in the requirement phase. Data set A includes the requirements describing the defects detected in the requirements review phase, while data set B includes the requirements describing the defects that shifted from this phase and could not be detected and detected in later development phases. For a sum of 500 defects occurring during the lifecycle, there were 439 defects in data set A and 61 defects in data set B. The distribution of these defects based on IEEE 830 standard [103] is analyzed and represented in Table 6.1.

The main costly defects in a system were caused due to shifting the effort from an earlier phase to another phase. To have a strategy to detect the failure and prevent its occurrence, the defects be analyzed and classified based on the attributes of requirements in the IEEE 830 standard. The defect is classified into two classes depending on their occurrence ratio and the correction cost.

	incorrect	ambigous	incomplete	inconsistent	not ranked	not verifiable	not modifiable	not traceable	∑
data set A	151	35	116	19	0	25	62	30	439
data set B	23	5	15	7	1	0	2	8	61
∑	174	40	131	26	1	25	64	38	500

Table 6.1 Requirement Defects Classified Based on IEEE 830

Four defects lists were be established that each list included the defects with specific occurs class and classified into:

1. List for most occur defects
2. List for least occur defects
3. List for most costly defects to be corrected
4. List for least costly defects to be corrected.

In order to identify and predict the defect class as above, there is a need to identify the defect sources. An analysis of different data sets and the experts' knowledge was conducted. The most important defect source with the most costly correction effort was classified in the requirement phase. The source of defects generated in the requirement phase could be classified into several possible defect sources.

One of the defect sources is the parameter in name, value and unit. The variant could cause a defect by indicating a wrong specification. The wording is another defect source, whereby the requirement could be written in wrong compliant to formulations, template phrases or the desired precision of requirements agreed upon in the project. The specified timing parameter, state machine which is modeling the system behavior, calculation or comparison, figure or related to a figure, missing or wrong requirement link, or a functionality understanding in the requirement need to be analyzed and documented. The intern data set was analyzed to detect the source of requirement defect. Table 6.2 illustrates the result of the distribution of defect source against the IEEE 830 classification.

	incorrect	ambigous	incomplete	inconsistent	not ranked	not verifiable	not modifiable	not traceable	∑	incorrect	ambigous	incomplete	inconsistent	not ranked	not verifiable	not modifiable	not traceable	∑
Parameter	19	1	12			1	8	20	61	6	2	2				1		11
variant	11	4	24	2			1	4	46								6	6
word	29	11	8	2		13	12		75									0
timing	4		4						8			2	2					4
state machine	13	1	3	2					19	4		5	2					11
calculation	11								11	5								5
figure	6	1	9	2			1	1	20									0
organisation	7	3	9	1			6	5	31		1		1	1				3
functionality	11	3	2	2		2	3		23	2	1	2	1					6
others	40	11	46	8		9	31		145	6	1	4	1			1	2	15
∑	151	35	117	19		25	62	30	439	23	5	15	7	1		2	8	61

Table 6.2 Defect Source Distribution against the IEEE 830 Classification

The defect source analysis leads to identify the defects sources in each phase of development lifecycle and help to prevent such defects. The result of this analysis added some new aspects in the defects prevention. To improve the effect detection, some points could be set in project overview like:

1. Involve the V&V engineers in the project from the beginning.
2. Parameters need to be used Separate from their concrete values.
3. Refinement process along the functional structure during development of requirement specification.
4. Inconsistency detection by using some automated tools. An automated tools with the ability of model checking techniques to detect even elaborate forms of inconsistencies between requirements need to be used [104] [105].
5. A report of defect need the type of the defect.

6. Analyze defects in order to screen the requirements engineering process.

6.5 Development effort analysis

Automotive software development needs several processes and is undertaken based on different standards and norms to satisfy their quality level. Development lifecycle take its time and several versions of system are released and tested during the total lifecycle until the start of production. The development phases include their quality assurance activities needed planned in specific mile stones. The defect effort will be included into the development effort during the lifecycle. Many change requests are developed and defects are fixed through these phases. These requests are estimated by the developer before developing the request starts. Many of these requests were estimated with the help of the models built in this thesis and a traceability matrix for the system was established. This section presents the results of these requests. A huge number of change requests (CRQ) were developed throughout this analysis. The analysis deals with a project developed for a real electric unit control (ECU), whereby the data needed for this analysis was accumulated from the change request system. Some of change request data does not include all information needed for the analysis. The result of this analysis in this section deals with 50 change request requirements throughout the total development phases. The requirement analysis, design and code implementation is undertaken by a thirteen-developer team, whereas the integration and the system test was completed by a different team of five developers.

6.5.1 Requirement phase

The 32 CRQs were estimated by the project manager by the help of the models and the KPIs described in the last chapters. The same CRQs were estimated by the developer in a separate system to have independent estimations. Eighteen CRQs included some defects data whereby the developers' estimation of the effort were conducted in the wrong way and the needed information in the system was not completed in the right way. Figure 6.4 illustrates the estimation effort from both systems and the real effort needed for these requirements. The total effort estimated by Project manager with help of the model was 530 hours, while the development estimation needed to complete these requests was 544 hours. The accuracy of the estimation is close to 97%. It is difficult to set this accuracy rate in general, because the effort needed for some change requests has lower accuracy. The defect resource leads to this lower estimated accuracy. Analyzing of the developer estimation describes a higher deviation of the real needed effort, because the developers give their estimation for the total CRQ including the design and implementation, and not only the requirement phase. Figure 6.5 shows the estimation accuracy as a percent ratio of the real effort needed.

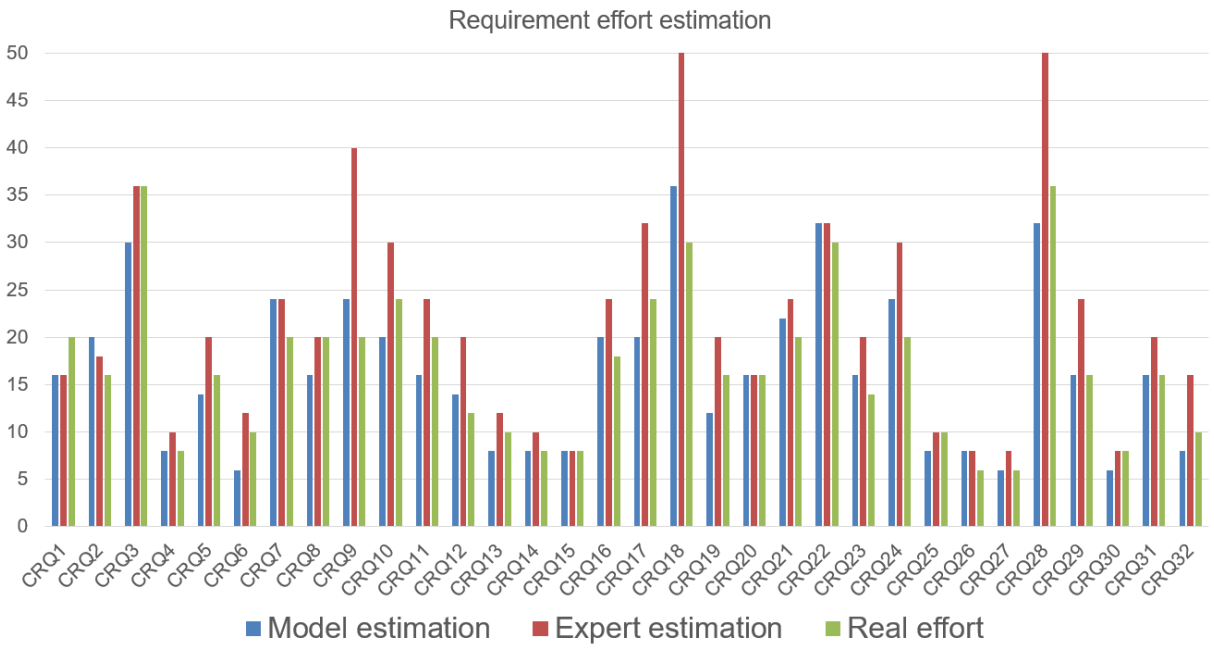


Figure 6.4 Estimated Requirement Effort

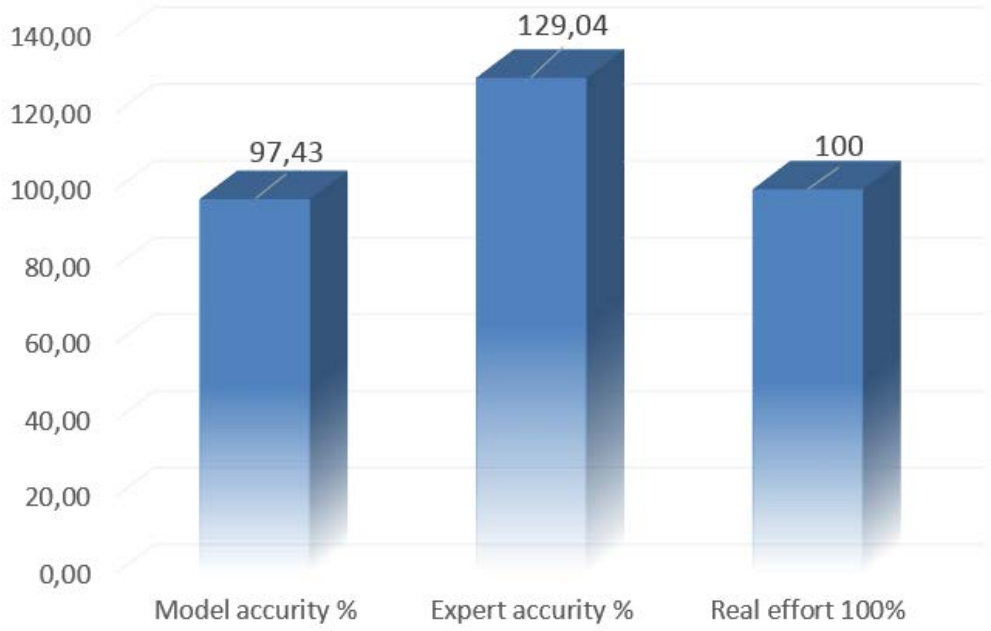


Figure 6.5 Requirement Effort Ratio

6.5.2 Design analysis

Most of these change requests deal with a defect or some changes required by the customer and do not deal with a feature development. Therefore, the design estimation includes the feature and module design phases which were not estimated as two phases of development. There were some changes included the change in features, while others included changes in module but the effort was estimated as a design estimation effort. The total effort needed for design phase was 426 hours as they accumulated from the change system after completing the development of all change requests. The effort needed for completing these changes was

estimated by the developer and the project managers with the help of the model and KPIs described in this thesis. The estimated effort was 396 hours and 440 hours, respectively.

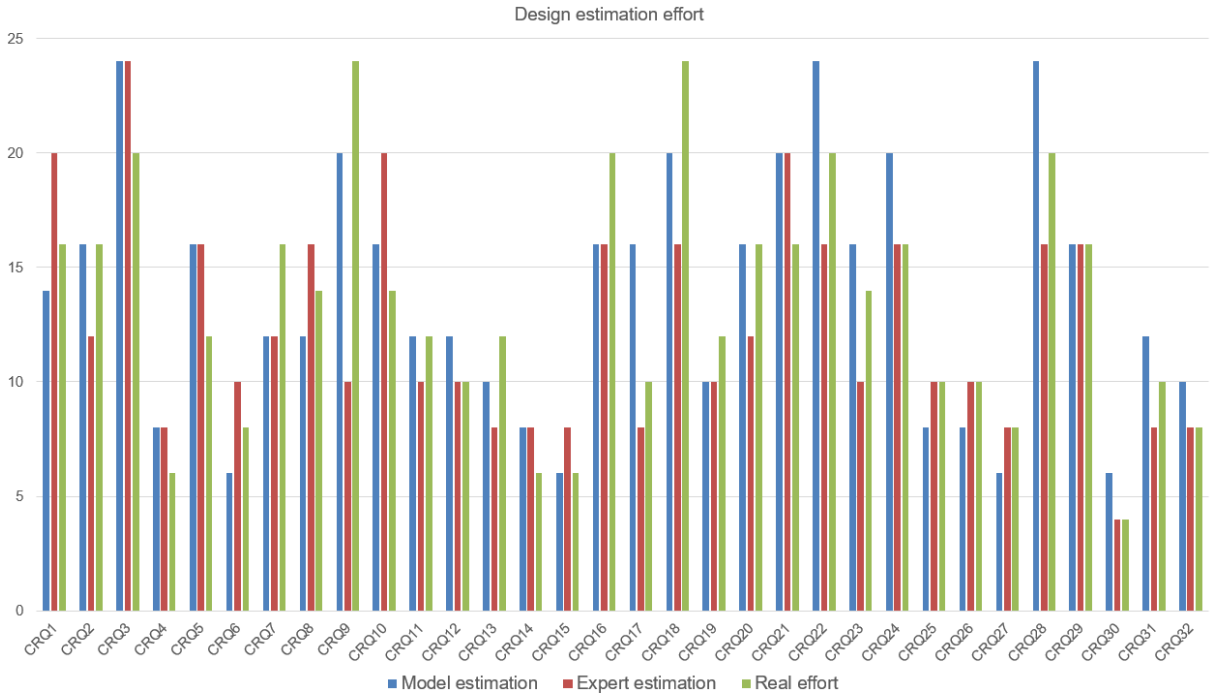


Figure 6.6 Effort Distribution for Design Phase

The design distribution in hours illustrates the necessary estimation effort undertaken by the Project manager with the help of thesis model, as well as the effort estimation by the team where the developer set the necessary effort that they predict for the specific change. The third line illustrates the real effort needed to complete the specific change. The effort values were taken from the change request system. The developer estimated the total changes of the CRQ and some developers estimated the total effort needed for the CRQ and divided the total effort among the different development phases. Figure 6.6 describes the effort needed for each CRQ. To illustrate the deviation of the estimation from the real effort spent in this phase, the estimation accuracy was calculated as a ratio of the effort divided by real effort of the same CRQ.

*Estimation accuracy = estimated effort / real effort *100*

Figure 6.7 illustrates the accuracy ratio of the total effort needed in the design phase.

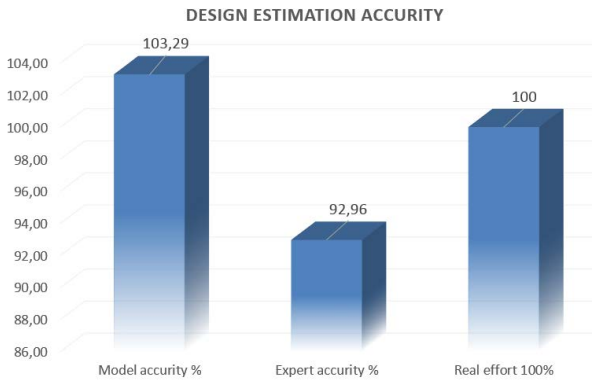


Figure 6.7 Design Accuracy Estimated Effort Ratio

6.5.3 Implementation effort distribution

The implementation phase presents the phase of code implementing in different ways. Some of the code was written manually, some others using a code generator tool and some by configuring the parameters with the help of some lists tools. The quality assurance in this phase includes a module test. The module test was conducted as open box test for the code using the MISRA rules for C coding and similar rules for Java and other programming languages. Some other unit tests using RTRT or similar method will be done to ensure the quality of the implementation phase. Code review is even an activity to be undertaken in this phase.

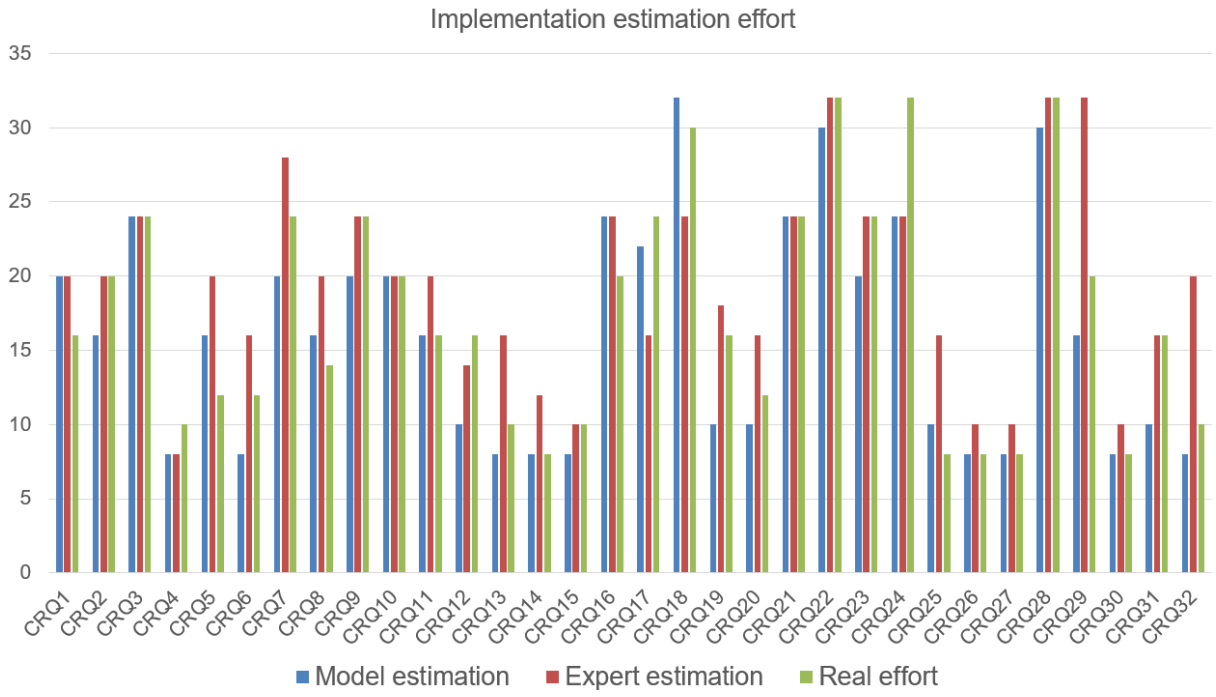


Figure 6.8 Implementation Effort Distribution

The total effort needed for the implementation phase was 560 hours, whereas the estimation effort with help of the supporting model was 512 hours. Accordingly, the phase estimation ratio is ~ 91%.

Figure 6.8 illustrates the effort estimated for each CRQ separately, whereby some of the estimation was done with very high accuracy to the real needed effort. Some of the CRQs had a deviation in effort from the real effort needed. Figure 6.9 illustrates the ratio of deviation from the real effort needed for total CRQs in this phase.

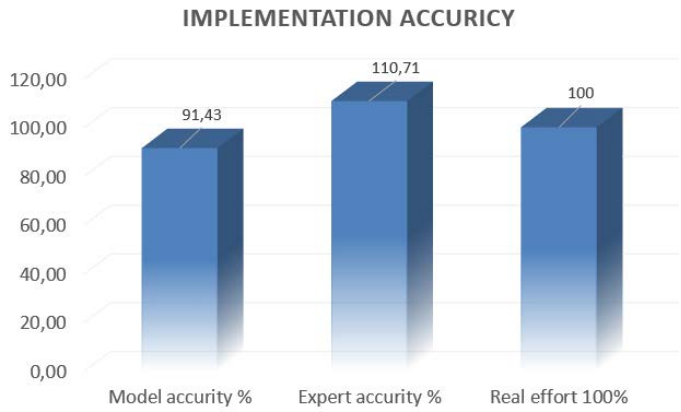


Figure 6.9 Effort Ratio Distribution

6.5.4 V&V phase

The implementation phase included the module test activities. The V&V phase starts with the integration of the software modules with the specific quality assurance and the system test activities. The system test was conducted with a different team to the software developer team. Test case implementation started parallel to the developing phases. The test specification requirement is the input node for the system test phase. The test cases were developed in the same development process used for the software with similar process with respect of the phase of the test. Several hardware and software supported this phase to prove the satisfaction of the system with the requirements. Most test cases were running automatically, while some needed to be run manually depending on the test case. The running effort was a major challenge to get the best estimated effort. Several test cases were developed and executed in Germany and some others were undertaken outside of Germany, which added a new challenge to the estimation due to personal factors. The estimation effort distribution among the CRQs is presented in Figure 6.10.

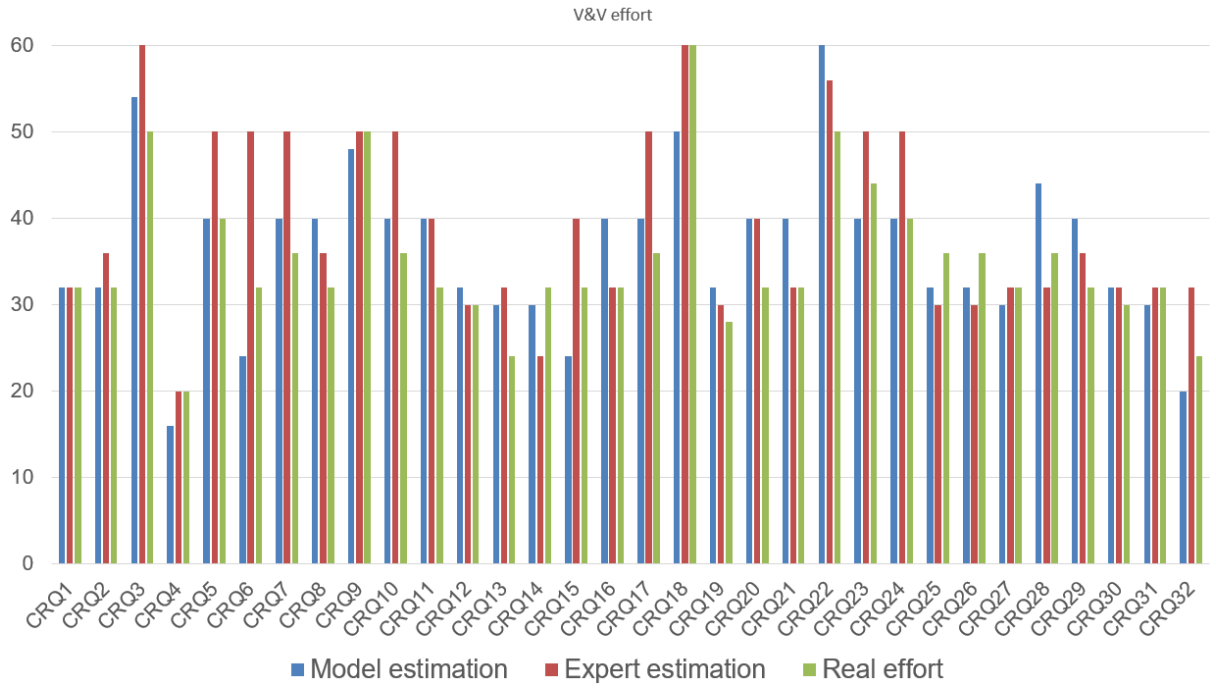


Figure 6.10 V&V Effort Distribution along the CRQs

Figure 6.10 illustrates the effort needed to complete each CRQ, and the estimated effort done by the project managers with the help of the model presented in this thesis, and the estimation done by test developer for the same CRQ. The total test effort needed was 1122 hours, and the estimated effort was 1164 hours.

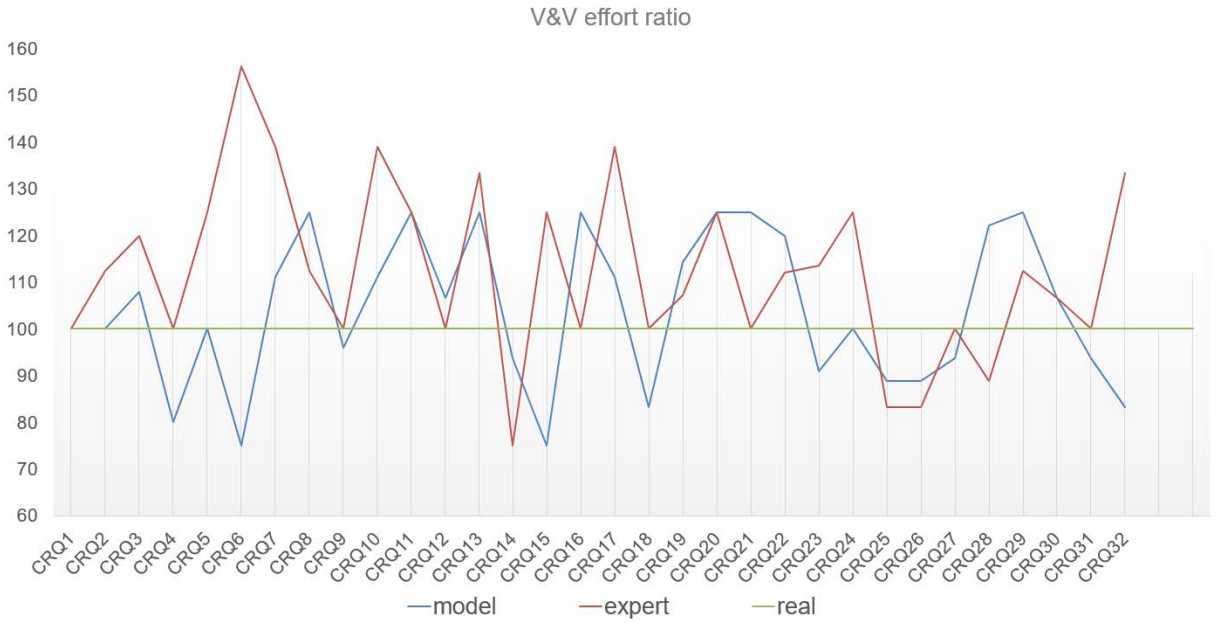


Figure 6.11 V&V Effort Ratio based on Real Effort

Figure 6.11 illustrated the effort ratio for the estimated effort to the real effort. The complexity of some CRQs and the different tools used in this phase led to a deviation in some CRQs. The deviation of the estimated effort from the real effort is close to 80%, with a few exceptions that the effort has a worse ratio, although analyzing these CRQs shows that these high deviations were caused due to the low effort needed for the CRQ; therefore, every small deviation in effort could lead to a high deviation ratio.

6.6 Practical results analysis

A safety-critical software development was completed based on different processes, standards and norms to satisfy their quality level. The quality assurance activities are important attributes affected in the quality of the software. The effort needed for this quality depends on the quality levels needed in the product. The CRQs analyzed in this chapter deal with ASIL C level of software according to ISO-26262 [9]. The quality assurance was planned with high effort to ensure the level of the software quality. The effort of different development phases was described in last section. An effort of 2,652 hours was needed to complete these CRQs with their specific quality activities. This thesis offers a model to support the Project manager in their decisions by estimation different change requests during the development lifecycle. Figure 6.12 illustrates the effort distribution over these CRQs as an estimation with the support of the model, as well as a developer estimation. The last effort values represent the real effort needed to complete each CRQ.

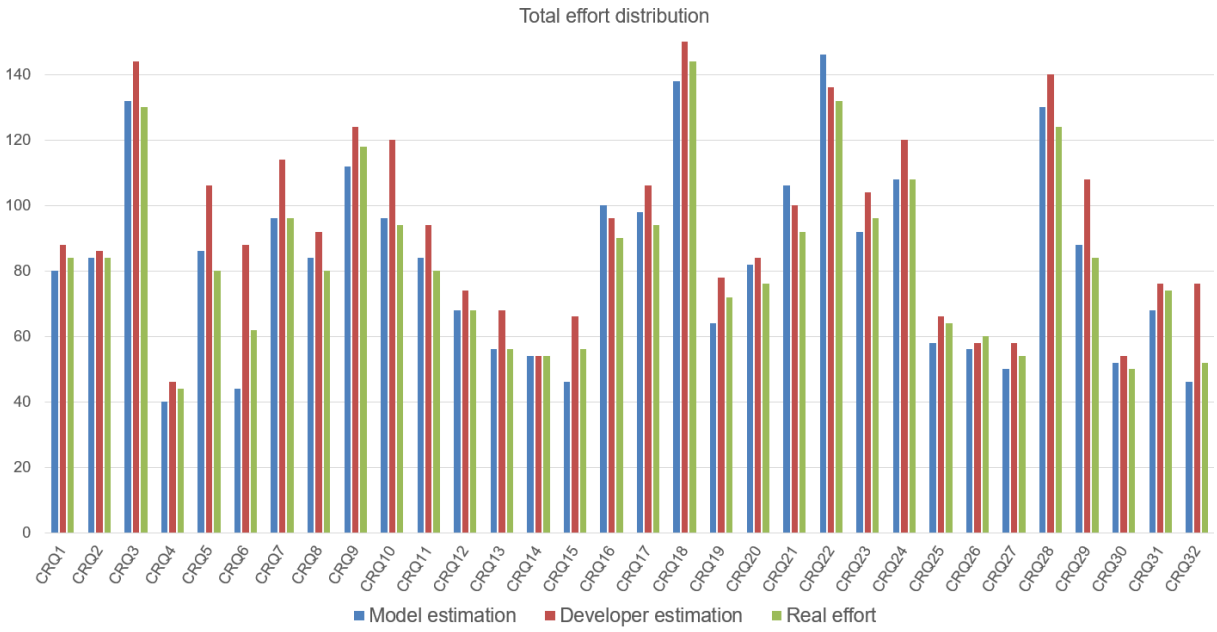


Figure 6.12 Total Project Effort Distribution

The range of needed effort for each CRQ varied from 44 hours to 144 hours. The complexity of the change has an effect on this effort in addition to other project parameters discussed in earlier chapters. This chapter described the effort estimated by the managers and the real effort needed. The change and project parameters were implicitly included in the model and will not be discussed here.

The ratio of each change was calculated as

$$\textit{Effort ratio} = \frac{\text{Estimated effort}}{\text{Real effort}} * 100\%$$

The effort ratio helps to compare the estimated effort by the model against the real effort needed. Figure 6.13 illustrates the effort ratio of the CRQs for the estimated effort by the model and the developer team, including the estimation of the test team.

Figure 6.13 shows the advantage of the support model for the Project manager, whereby most estimation values of the CRQs are close to the real development effort. The support model describes the system and supports the managers with a good overview of the project, as well as supporting the Project manager with all KPIs that affected the effort for each CRQ. This system information with all interfaces supports managers with all needed information to have a good estimation. Figure 6.15 shows that the accuracy of the CRQ was between 83 and 120 % based on the real effort needed, which means that the model provides help for Project manager in effort estimation with 83% accuracy.

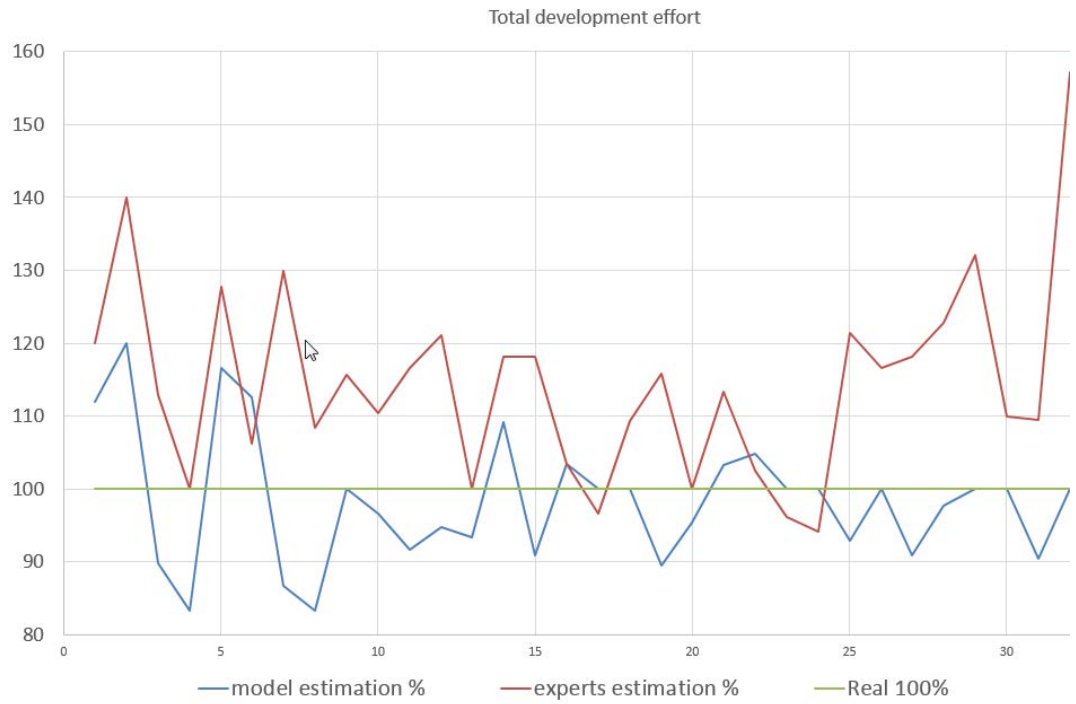


Figure 6.13 Effort Ratio Distribution over CRQs

The Requirements in this chapter were done with real projects. The result and users opinions after developing the pilot project using the method suggested in this thesis shows:

Hypothesis (H), Goals / Requirement (G)	Refinement
G1: The overall development effort shall be decreased by using a new model to identify the relevant quality assurance measures.	Better description for the system helps to plan right quality assurance and lead to effort reduction.
G2: A new method is needed to identify hot spots in a software product (e. g., components or even single functions). These hot spots are most promising for a decrease of the defect rate and/or the development effort.	DSM and crosscutting approach represent a total overview on developed system. That led to detect most of defects in earlier phases of development lifecycle and reduce the defect rate and the development effort.
G3: A development process has to be created to derive the KPIs for the reduction of the development effort as well as the effort of defect correction.	The process developed and used with positive results.
H1: Current estimation models can be improved by the integration and feedback of process data. The estimation accuracy will increase.	Results show an improvement in estimation accuracy compared with the real effort needed.
H2: With a new estimation model (see H1) project managers can identify the hot spots in a products architecture where the most effort reduction is possible.	The second iteration of using the model in the project illustrates a defect rate reduction because of the improvement activities in the first iteration.
H3: A new development process with distributed efforts for quality assurance measures will reduce the effort of failure correction as well as the overall development effort.	The model shows a reduction in quality assurance effort and lead to reduce the total development effort

7 Summary

The essential physical characteristics and the majority of the lifecycle costs of a new vehicle are determined during the initial and conceptual design phases. The features determined in these early development phases are important for the whole vehicle design process. The increase in product complexity, functional requirements, and number of derivatives has brought about a need for new technical solutions and tools to support decision makers in their projects. This thesis presents a model to help project managers and the decision makers make their decisions. A DSM was created to describe the system and all interfaces among system components and provide a good overview of the system. Defining the KPIs which most affect costs, leads to good cost predictions during the development lifecycle.

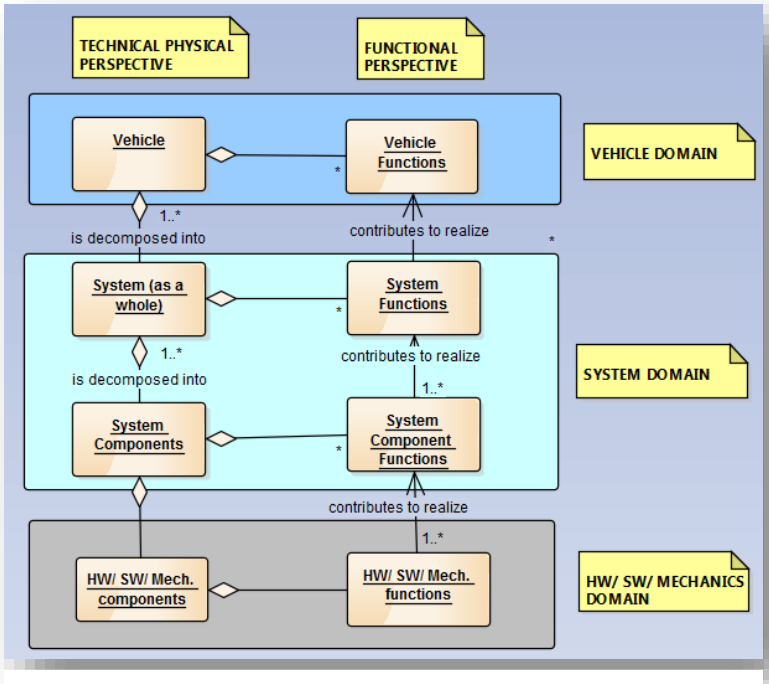
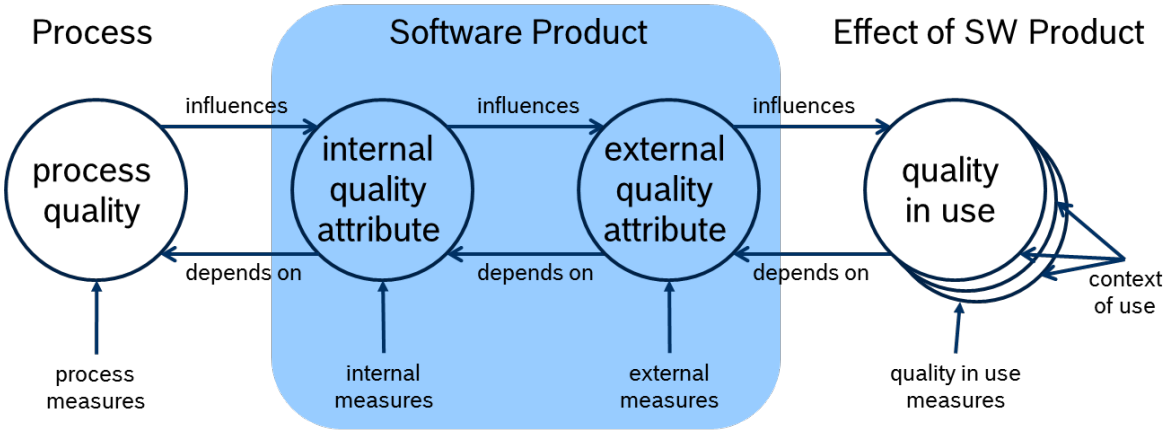


Figure 7.1 Architect and Design Developing

7.1 Software development and automotive domain

Automotive software is a safety-critical product. Product quality is crucial. High quality requirements increase development effort. Figure 7.2 shows a software system with the desired quality attributes.



Reference: ISO/IEC 25010:2011

Figure 7.2 Software Quality Attributes

Developing high quality software requires a qualitative process to ensure product quality. System development starts by analyzing the functionality of the product and walking through all the necessary product details. Figure 7.3 shows a product design walkthrough. The complexity of the software and the huge amount of interfaces between components all but guarantees defects will be generated during the development lifecycle. The cost control of the project needs to control the cost of these defects, too. Therefore, defect effort is included as part of development effort.

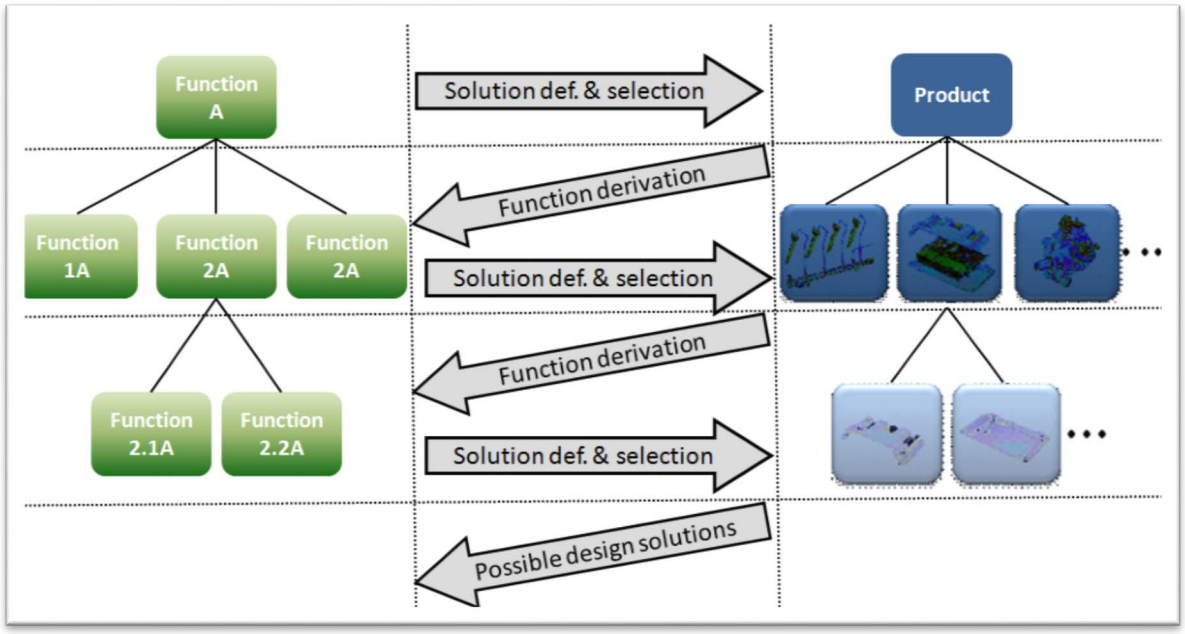


Figure 7.3 Design Procedure

Managing this huge number of parameters with deadline pressure is a major challenge for project managers. A model to support the managers in their decision making will be a major boon in facing this challenge.

7.2 DSM and traceability

Complex system development needs good traceability between the different system components. The last few chapters illustrated traceability from requirements to design to implementation, including test activities in the V&V phase. DSM an provide a clear overview of traceability. The interfaces between the different models of the system have different effects on each other. The interface effects were described and calculated using the crosscutting concern method. These effects require vast knowledge about the system. The most influential KPIs in the system were described and illustrated to help estimate the effect of each system component.

7.3 Correction process model

Software development which takes a long period of time must define of a large number of metrics to manage this development. Development processes are very important to meet functional and nonfunctional requirements at a good quality level. Many parameters affect software cost and quality. The process model describes the KPIs needed for cost illustration and controlling. The model describes the ratio between software defects and their correction effort, allowing the process to be improved while also improving the defect rate during the development lifecycle. The model's status as a graphical representation makes the identification of each problem easier to be estimated even for high complex system, challenges of changes in a system with high complexity will be available and identifying the weaknesses in each component or even by added a new features in complex systems will be possible with lower effort.

7.4 Cost flow model

Dataset from closed and ongoing projects were analyzed in conjunction with different expert knowledge and literature. A model describing the cost flow in a software project was presented. The KPIs which most influence cost flow during the different phases of the lifecycle were derived and included in the model. Including all KPIs which affect development effort helps achieve a better cost estimate for system defects. The cost flow model highlights the areas which generate the most defects. It highlights the need for quality assurance activities in these phases and avoids a major defect rate.

7.5 Perspectives

Estimating the effort required for development and software changes is essential for controlling a project and achieving the desired quality. Better effort estimation leads to better scheduling of the project milestones, higher product quality, and reduced costs. The processes presented in this thesis in depth experience and knowledge of software development in the automotive industry. The continuous improvement of these processes is an important step in improving the estimation performance of future projects. A user-friendly framework that provides this information should be established. A database will be needed to store the decision arguments that lead to improvements in the estimation process. To generalize this framework for use outside of the automotive industry, further processes and KPIs must be collected and analyzed.

8 References

[1] K. Hauser and W. Zimmermann, “The growing importance of embedded software: Managing hybrid hardware-software business,” *Technical report, The Boston Consulting Group, Inc.,* 2004.

[2] Robert Bosch GmbH Central (intern), “Central directive quality (CDQ),” 2010.

[3] P. Marwedel, “Embedded System Design,” Springer, 2nd edition, 2011. doi:10.1007/978-94-007-0257-8, pages 1, 9, 15, and 33.

[4] A. Avizienis, J.-C. Laprie and B. Rand, “Basic concepts and taxonomy of dependable and secure computing,” *IEEE Transactions on Dependable and Secure Computing, 1(1):11–33, 2004. doi:10.1109/TDSC.2004.2,* pp. pages 1, 15, 16, 17, 18, 19, and 32, 2004.

[5] S. Litoriya, N. Sharma and Ratnesh, “Incorporating Data Mining Techniques on Software Cost Estimation: Validation and Improvement,” *International Journal of Emerging Technology and Advanced Engineering, vol.-2,* 2012.

[6] S. McConnell, Software Estimation, Microsoft Press, ISBN: 978-0-7356-0535-0, 2006.

[7] L. Insup Lee, J. Y. Leung and S. H. Son, Handbook of Real-Time and Embedded Systems, Chapman & Hall/CRC, ISBN 1584886781, 2007.

[8] R. J. Madachy, “A Software project dynamics model for process cost, schedule and risk assessment,” http://sunset.usc.edu/TECHRPTS/PhD_Dissertations/files/Madachy_Dissertation.pdf.

[9] ISO Org, “Road vehicles Functional safety,” http://www.iso.org.

[10] IEEE, “Software & Systems Engineering Standards Committee of the IEEE Computer Society”.*IEEE Standard for Software Quality Assurance Processes, https://ieeexplore.ieee.org/stamp/stamp.jsp? tp=&arnumber=6835311.*

[11] B. Boehm, S. Chulani and Horowitz, Software Cost Estimation with Cocomo II, Prentice Hall, 2000.

[12] Software Engineering Institut, “Key Practices of the Capability Maturity Model,” http://www.sei.cmu.edu/reports/.

[13] J. A. Farquhar, S. C. RAND and M. CALIF, “A Preliminary Inquiry Into the Software Estimation Process,” *DEFENSE TECHNICAL INFORMATION CENTER,* 2006.

[14] N. Karunanithi, D. Whitley and Y. K. Malaiya, “Using Neural Networks in Reliability Prediction,” *IEEE Software, vol. 9, no. 4,,* pp. pp. 53-59..

[15] M. Schofield and C. Shepperd, “Estimating Software Project Effort Using Analogies,” *IEEE Transactions on Software Engineering,,* no. Vol. 23, No. 11, 1997.

[16] L. C. Briand,, K. El Emam, D. Surmann and I. Wieczorek, A Comparison of the Jensen and COCOMO Schedule and Cost Estimation Models, Proceedings of the International Society of Parametric, ISBN:1-58113-074-0.

[17] CMMI, “http://www.trinity-cmmi.co.uk/CMMI_Representations.htm,” 2011.

[18] V. Basili, L. Briand and S. Condon, “Understanding and predicting the process of software maintenance releases,” in *Proceedings of International Conference on Software Engineering*, Berlin, Germany, 1996.

[19] CIO Bund, “V-Modell-XT,” http://www.cio.bund.de/Web/DE/Architekturen-und-Standards/V-Modell-XT/ vmodell_xt_node.html, .

[20] G. G. Judge, W. E. Griffiths and C. R. Hill, The Theory and Practice of Econometrics, John Wiley and Sons, 1985.

[21] A. Garcia and C. Lucena, “Taming heterogeneous agent architectures,” *COMMUNICATIONS OF THE ACM,* no. Vol. 51, No. 5, p. pp 75–81, 2008.

[22] D. Sundar, Software engineering, Published by Laxmi Publications Pvt. Ltd., ISBN 13: 9789380386195, 2010.

[23] T. R. Browning, “Applying the design structure matrix to system decomposition and integration problems: a review and new directions,” *Engineering Management, IEEE Transactions,* pp. pp. 292-306, 2001.

[24] Autosar.org, “Autosar.org,” Overview and Revision History, V1.2.0 Release 4.1 Rev 3, [Online]. Available: http://www.autosar.org/.

[25] A. V. Aho, Modeling, Measuring, and Modularizing Crosscutting Concerns, Doctor of Philosophy in Computer Science at Columbia University, 2006.

[26] ISO, “Software Engineering Product quality – Part 1: Quality Model,” *International Organization of Standardization,* pp. ISO/IEC 9126-1, 2001.

[27] R. Filman, T. Elrad and S. Clarke, Aspect-oriented software development, Addison-Wesley Professional, ISBN:0321219767, 2006.

[28] N. Nagappan, T. Ball and A. Zeller, “Mining Metrics to Predict Component Failures,” *ICSE '06 Proceedings of the 28th international conference on Software engineering, China,* pp. pp. 452-461, 2006.

[29] W. E. Wong, S. S. Gokhale and J. R. Horgan, “Quantifying the closeness between program components and features,” *Journal of Systems and Software, 54,* pp. pp 87-98, 2000.

[30] M. Revelle, T. Broadbent and D. Coppit, “Understanding Concerns in Software: Insights Gained from Two Case Studies,” in *Workshop. on Program Comprehension*, 2005.

[31] V. Nguyen and B. Boehm, “A Controlled Experiment in Assessing and Estimating Software Maintenance Tasks,” in *Conference: 16th Asia Pacific Software Engineering Conference*, 2009.

[32] D. Freiman and R. Park, “PRICE Software Model. Version 3. An Overview,” *Proceedings of IEEE-PINY Workshop on Quantitative Software Models for Reliability,* 1979.

[33] P. Suri and P. Ranjan, “Comparative Analysis of Software Effort Estimation Techniques,” *International Journal of Computer Applications,* no. 21, p. pp. 975 – 8887, 2012.

[34] L. H. Putnam and M. Ware , Measures For Excellence: Reliable Software On Time, Within Budget, Yourdon Press Computing Series, ISBN-10: 0135676940, 1992.

[35] B. Boehm, Software Engineering Economics (Prentice-Hall Advances in Computing Science & Technology Series), New Jersey: Prentice-Hall PTR, ISBN 0-13-822122-7, 1981, pp. pp. 4-21.

[36] B. Boehm, B. Clark, E. Horowitz and C. Westland, “Cost Models for Future Software Life Cycle Processes: COCOMO 2.0,” *Annals of Software Engineering Special Volume on Software Process and Product Measurement,* vol. Vol. 1, pp. pp. 45 - 60, 1995.

[37] D. J. Reifer, “Industry software cost, quality and productivity benchmarks,” *DACS Newsletter,* p. software. Tech News 7, 2004.

[38] S. Vicinanza, T. Mukhopadhyay and M. Prietula, “Software Effort Estimation: An exploratory study of expert performance,” *Information Systems 2,* pp. pp. 243-262, 1991.

[39] Chemuturi and Murali, “Delphi Technique for software estimation,” www.chemuturi.com.

[40] B. Boehm, C. Abts and S. Chulani, “Software development cost estimation approaches – A survey,” *Annals of Software Engineering 10,* p. P. 177–205, 2000.

[41] A. Gray and S. MacDonell, “Alternatives to regression models for estimating software projects,” *Information Science Discussion Papers Series,* no. No. 96/17, 1996.

[42] S. Anthony, E. Kazmierczak and L. Sterling, “Software Engineering Methods for Neural Networks,” *IEEE Proceedings of the Tenth Asia Pacific Software Engineering Conference,* pp. pp 468-477, 2003.

[43] R. C. Satyananda and K. Raju, “A Concise Neural Network Model for Estimating Software Effort,” *International Journal of Recent Trends in Engineering,* no. Vol.1, 2009.

[44] J. Cheng, “Virtual vehicle sensors based on neural networks trained using data generated by simulation models,” Ford Global Technologies, Inc. (Dearborn, MI), United States Patent, 2001.

[45] M. Mozolin, J. Thill and L. E. Usery, “Trip distribution forecasting with multilayer perceptron neural networks: A critical evaluation,” *Transportation Research Part B 34,* pp. pp 53-73, 2000.

[46] T. Abdel-Hamid and S. E. Madnick, Software project dynamics : an integrated approach, Prentice Hall, ISBN-13: 978-0138220402 .

[47] V. Nguyen, B. Steece and B. Boehm, “A constrained regression technique for COCOMO calibration,” *Proceedings of the 2nd ACM-IEEE international symposium on Empirical software engineering and measurement (ESEM),* pp. pp. 213-222, 2008.

[48] B. Yang and Y. Clark, “COCOMO II. 2003 Calibration Status,” *CSE Annual Research Review, ,* no. http://sunset.usc.edu/events/2003/March_2003/COCOMO_II_2003_ Recalib_ration.Pdf, 2003.

[49] T. Schulz, Ł. Radliński and T. Gorges, “A Bayesian Network for Predicting Defect Correction Effort,” in *Proceedings of the 6th International Conference on Predictive Models in Software Engineering Article No. 16, ACM*, New York, NY, USA , ISBN: 978-1-4503-0404-7, 2010.

[50] A. Gelman, J. Garlin, H. Stern and D. Rubi, Bayesian Data Analysis, Chapman Hall, ISBN: 9781439898208, 2013.

[51] D. D. Galorath, “Software total ownership costs: development is only job one,” in *Software Tech News 11*, 2008.

[52] J. Ramil, Continual Resource Estimation for Evolving Software,", University of London, Imperial College of Science, 2003.

[53] I. Myrtweit, E. Stensrud and M. Shepperd, “Reliability and Validity in Comparative. Studies of Software Prediction Models,” vol. IEEE Transactions on Software Engineering, no. Vol. 31, No. 5, 2005.

[54] D. Caivano, F. Lanubile and G. Visaggio, “Software renewal process comprehension using dynamic effort estimation,” *Proceedings of International Conference on Software Maintenance, Florence, Italy,,* p. pp. 209–218, 2001.

[55] M. Baldassarre and N. Boffoli, “Improving Dynamic Calibration through Statistical Process Control,” in *IEEE International Conference on Software Maintenance (ICSM’05)*, 2005.

[56] A. Trendowicz, J. Heidrich, J. Münch and Y. Ishigai, “Development of a hybrid cost estimation model in an iterative manner,” *Proceedings of the 28th International Conference on Software Engineering, China,* 2006.

[57] P. Abrahamsson, R. Moser, W. Pedrycz and A. Sillitti, “Effort Prediction in Iterative Software Development Processes Incremental Versus Global Prediction Models,” in *Proceed-ings of the First International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2007.

[58] H. Sneed and S. Huang, “Sizing Maintenance Tasks for Web Applications,” in *11th European Conference on Software Maintenance and Reengineering, CSMR '07*, 2007.

[59] H. M. Sneed, “Estimating the Costs of a Reengineering Project,” in *Proceedings of the 12th Working Conference on Reverse Engineering (WCRE’05)*, 2005.

[60] F. Niessink and H. Van Vliet, “Two case study in measuring maintenance effort,” *Proceedings of International Conference on Software Maintenance, Bethesda, MD, USA,* p. pp. 76–85, 1998.

[61] A. Abran, I. Silva and L. Primera, “Field studies using functional size measurement in building estimation models for software maintenance,” *Journal of Software Maintenance and Evolution,* pp. pp. 31-64, 2002.

[62] F. Fioravanti and P. Nesi, “Estimation and prediction metrics for adaptative maintenance effort of object-oriented systems,” *IEEE Transactions on Software Engineering 27,* p. pp. 1062–1084, 2001.

[63] V. Basili, S. Condon, K. Emam and R. Hendrick, “Characterizing and Modeling the Cost of Rework in a Library of Reusable Software Components,” *Proceedings of the 19th International Conference on Software Engineering,* pp. pp.282-291, 1997.

[64] A. De Lucia, E. Pompella and S. Stefanucci, “Assessing effort estimation models for corrective maintenance through empirical studies,” *Information and Software Technology 47,* p. pp. 3–15, 2005.

[65] M. K. Østvold and J. Magne, “A review of software surveys on software effort estimation,” *Empirical Software Engineering, ISESE 2003. IEEE,* p. pp. 223–230, 2003.

[66] D. Yang, W. Qing Wang, L. Mingshu Li, Y. Ye and K. Ye, “A survey on software cost estimation in the chinese software industry,” in *Proceedings of the Second ACM IEEE international symposium on Empirical software engineering and measurement*.

[67] A. Trendowicz, J. Münch and R. Jeffery, “State of the practice in software effort estimation: a survey and literature review,” *Software Engineering Techniques ,* p. pp. 232–245, 2011.

[68] T. Chow and C. Dac-Buu, “A survey study of critical success factors in agile software projects,” *Journal of Systems and Software,* p. pp. 961–971, 2008.

[69] K. M. Furulund, “Empirical research on software effort estimation accuracy,” PhD thesis, Department of Informatics, University of Oslo, 2007.

[70] L. C. Briand and I. Wieczorek, “Resource Estimation in Software Engineering,” International Software Engineering Research Network, Technical Report (ISERN 00-05), 2005.

[71] S. MacDonell and M. Shepperd, “Combining techniques to optimize effort predictions in software project management,” *Journal of Systems and Software,* no. 66, p. pp. 91–98, 2003.

[72] X. Huang, D. Ho, J. Ren and L. F. Capretz, “Improving the COCOMO model using a neuro-fuzzy approach,” *Applied Soft Computing,* no. 7, p. pp. 29–40, 2007.

[73] E. Stensrud and I. Myrtveit, “Human performance estimating with analogy and regression models: an empirical validation,” in *The 5th International Software Metrics Symposium*, 1998.

[74] M. Jørgensen, “A review of studies on expert estimation of software development effort,” *Journal of Systems and Software,* no. 70, p. pp. 37–60, 2004.

[75] E. S. Jun and J. K. Lee,, “Quasi-optimal case-selective neural network model for software effort estimation,” *Expert Systems with Applications,* no. 21, pp. pp. 1-14, 2001.

[76] S. Bibi and I. Stamelos, “Selecting the appropriate machine learning techniques for the prediction of software development costs,” *Artificial Intelligence Applications and Innovations, vol. 204, Springer, Boston,* p. pp. 533–540, 2006.

[77] I. Tronto, J. Silva and N. S. Anna, “An investigation of artificial neural networks based prediction systems in software project management,” *Journal of Systems and Software,* no. 81, p. pp. 356–367, 2008.

[78] A. Idri, A. Zahi, E. Mendes and A. Zakrani, “Software cost estimation models using radial basis function neural networks,” *International Workshop on Software Measurement,* p. pp.21–3, 2007.

[79] S. Berlin, T. Raz, C. Glezer and M. Zviran, “Comparison of estimation methods of cost and duration in IT projects,” *Information and Software Technology,* no. 51, p. pp. 738–748, 2009.

[80] L. C. Briand, T. Langley and I. Wieczorek, “A replicated assessment and comparison of common software cost modeling techniques,” in *Proceedings of the 22nd International Conference on Software Engineering*, Limerick, Ireland, 2000.

[81] M. O. Elish, “Improved estimation of software project effort using multiple additive regression trees,” *Expert Systems with Applications,* no. 36, 2009.

[82] A. Heiat, “Comparison of artificial neural network and regression models for estimating software development effort,” *Information and Software Technology,* no. 44, p. pp. 911–922, 2002.

[83] Y. Li, M. Xie and T. Goh, “A study of mutual information based feature selection for case based reasoning in software cost estimation, Expert Systems with,” no. 31, p. pp. 5921–5931, 2009.

[84] E. Mendes and N. Mosley, “Bayesian network models for web effort prediction: a comparative study,” *IEEE Transactions on Software Engineering,* no. 34, 2008.

[85] S. Bibi, I. Stamelos and L. Angelis, “Combining probabilistic models forexplanatory productivity estimation,” *Inf. Softw. Technol. , http://dx.doi.org/10.1016/j.infsof.2007.06.004.,* no. 50, p. pp. 656–669, 2008.

[86] Y. Fu, M. Li and F. Chen, “Impact propagation and risk assessment of requirement changes for software development projects based on design structure matrix,” *International Journal of Project Management,* pp. pp 363–373, 2012, 2012.

[87] B. W. Boehm, “Software Engineering Economics,” *IEEE transactions on software engineering, VOL. SE-10, NO. 1,* pp. pp. 4-21, 1984.

[88] J. E. Gaffney, “A comparison of a complexity-Based and Halstead program size estimates,” in *presented at the 1979 ACM Comput. Sci. Conf., Dayton, OH*, 1979.

[89] S. M. Weiss, A Practical Guide to Designing Expert Systems, Rowman & Littlefield Publishers, Inc., ISBN 9780865981089., 1984.

[90] V. Basili, G. Caldiera and D. Rombach, The Goal Question Metric Approach, Encylopedia of Software Engineering-2, ISBN #1-54004-8, 1994.

[91] B. Kitchenham, “Software Development Cost Models,” in *Software Reliability Handbook*, Elsevier Applied Science, NY, 1990, pp. pp. 487-517.

[92] M. Andolfi, M. Costamanga, E. Paschetta and G. Roseng, “A multicriteria-based methodology for site prioritisation in sediment management,” Environment international, 2009.

[93] R. G. Sargent, “Verification and validation of simulation models,” *WSC 05: Proceedings of the 37th conference on Winter simulation ,* p. pp 130–143, 2005.

[94] Autosar, “Release 4.1 Overview and Revision History, V1.2.0 Release 4.1 Rev 3,” http://www.autosar.org/.

[95] S. Nadir, D. Streitferdt and C. Burggraf, “Software code generator in Automotive field,” in *International Conference on Computational Science and Computational Intelligence*, 2015.

[96] NCHRP, “Incorporating ITS Into the Transportation Planning Process: An Integrated Planning Framework (ITS, M&O, Infrastructure),” Practitioner's Guidebook, https://www.nap.edu/read/22017, 2002.

[97] D. Ernst and F. Houdek, “GQM Method Application,” http://www-ivs.cs.uni-magdeburg.de/ sw-eng /us/java/GQM/link3.shtml, 2013.

[98] R. Pulfer, “Comparision of models of complex system,” in *United States Patent Application Publication, Pub No. US 2010.0036874A1*, 2010.

[99] SIG, Automotive, “Automotive SPICE Process Reference Model,” http://www.automotive, 2012.

[100] D. A. Wheeler , B. Brykczynki and R. N. Messon, Software Inspection: An Industry Best Practice for Defect Detection and Removal, CA, USA: IEEE Computer Society Press Los Alamitos, ISBN:0818673400 , 1996.

[101] C. Jones, Software Quality: Analysis and Guidelines for Success, International Thomson Computer Press, ISBN 9781850328674, 2000.

[102] J. Rumbaugh, I. Jacobson and G. Booch, Unified Modeling Language Reference Manual, Pearson Higher Education, ISBN:0321245628, 2004.

[103] IEEE Standards Association, “IEEE Standards Association,” https://standards.ieee.org.

[104] A. Post and J. Hoenicke, “A New Property for Real Time Requirements,” *International Conference, European Conferences on Theory and Practice of Software,* p. pp 34–49, 2011.

[105] A. Post, J. Hoenicke and A. Podelski, “Vacuous real-time requirements,” *IEEE International Requirements Engineering Conferencen 19th, Trento, Italy,* p. pp 153–162, 2011.

[106] S. Højsgaard and S. Lauritzen, “Graphical Gaussian models with edge and vertex symmetries,” *Journal of the Royal Statistical,* pp. pp. 1005-1027, 2008.

Selbstständigkeitserklärung

Ich versichere, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.

Insbesondere habe ich hierfür nicht die entgeltliche Hilfe von Vermittlungs- bzw. Beratungsdiensten (Promotionsberater oder anderer Personen) in Anspruch genommen. Niemand hat von mir unmittelbar oder mittelbar geldwerte Leistungen für Arbeiten erhalten, die im Zusammenhang mit dem Inhalte der vorgelegten Dissertation stehen.

Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder ähnlicher Form einer Prüfungsbehörde vorgelegt. Ich bin darauf hingewiesen worden, dass die Unrichtigkeit der vorstehenden Erklärung als Täuschungsversuch angesehen wird und gemäß § 7 Abs. 10 der Promotionsordnung den Abbruch des Promotionsverfahrens zu Folge hat.