Industrial software developments effort estimation model

Shahab Nadir
Technical university of Ilmenau
Ilmenau, Germany
sh.nadir@gmx.de

Prof. Detlef Streitferdt
Technical university of Ilmenau
Ilmenau, Germany
detlef.streitferdt@tu-ilmenau.de

Christina Burggraf
Media university of Stuttgart
Faculty of Communication and Information
Stuttgart- Germany
cb137@hdm-stuttgart.de

Abstract

Over the past decade, software has spread to most areas of our lives. This has led to increased demands on product quality and complexity. Industrial software which belong to a safety-critical area where high quality products are essential. Many processes and standards must be completed and met within stipulated deadlines. The complexity of the software and the boundary conditions of developing it dictate the product defect ratio. Requirements will be added and adjusted during the development lifecycle which will leads to changes in the software. All these changes need to be refined and tested to ensure product quality. Predicting software defects and changes is a major part of software engineering.

The software defect count is used to assess quality and estimate the effort needed for software fixes; however, it is rarely accurate. Industrial development involves many standards and norms. This research models correction efforts from multiple perspectives. The effort needed for software fixes needs to be combined with overall product effort to ensure software quality. The most common approach is to update the software. Several software update will be applied with each release. Each defect has specific characteristics. These depend on the process and project parameters, with the parameters determining whether a defect exists and the probability of generating a new one. The software process model contained in this thesis will describe these parameters and illustrate their effect on the software defect rate. The software is developed in different phases during the development lifecycle. The effort needed to fix defects depends on the phases where they originated and were detected. This is modeled in this thesis as a defect cost flow. Describing the distribution of defects during these phases helps control quality assurance and limit the effort needed to increase product quality.

Keywords—Automotive software, Software cost estimation, software traceability, industrial software development.

I. INTRODUCTION (*HEADING 1*)

Safety, information security, and development costs are very important metrics for industrial software. When developing a system to improve these metrics for industry, Sneed's "devil's square" should be taken in account [1]. The devil's square models the relationship between scope, quality, time, and cost and is a useful tool for planning and controlling during the software development cycle and for marketing it once the cycle is complete as described in Figure 1.1.

In order to develop a system which improve the metrics for automotive industry, the square's devil above should be taken in account. These software metrics could be divided depends on their focus in following areas [2]:

- Analysis
- Changeability
- Stability
- Testability

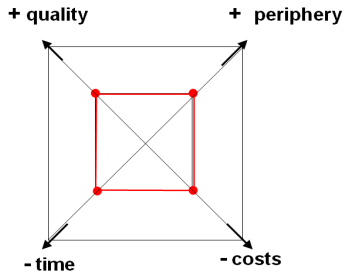

Figure 1.1 Devil's square by Sneed

Satisfying these properties in software project need to have some different processes to be used in automotive industry.

978-1-5090-5510-4/16 $31.00 © 2016 IEEE
DOI 10.1109/CSCI.2016.234

Automotive software is safety-critical so product quality is high priority [3]. The automotive software development cycle is long, with many iterations to ensure the necessary features are present and of sufficient quality. The iterations during the development cycle will involve many change requests: some to add or edit requirements and features, and others to correct defects which originated in a different phase of the cycle. These requests related to changes in project requirements. These change requests and various other factors produced different software defects. These defects had to be analyzed and fixed during the different development phases and before release. Figure 1.2 illustrates the distribution of software defects during the same development lifecycle [4].

The defects count in the software is used to assess the quality and estimate the effort needed for software correction. The defects count is not an accurate indicator for the effort needed to fix the software and adapt the changes. Automotive domain has a lot of standards and norms involve the development process. Therefore, this research modeled the effort of correction from all possible perspectives. The software defects effort need to be inserted in the product effort to ensure the quality of the software. The most used technic to insert these effects within the software product is to add a change for the software. There are a several applied number of changes in the software with each release [5]. Each of these defects has a specific characteristics depending on the process and project parameters, these parameters set the change as a defect and described the probability of generating a software defect. Describing these parameters and their effect on the software defect rate would be illustrated in the model of the software process in this research. The software is developed in different phases during the development life cycle. The effort needed for correction of the defects depends on the phase where they originated and were detected. The description of the defects distribution during these phases helps to control the quality assurance activities and the effort needed to raise the quality level of the product. The defect distribution over the different development phases and the effect of the originated and detection phases on the effort needed for the correction of the software is realized in this research and described as a model of defect cost flow.

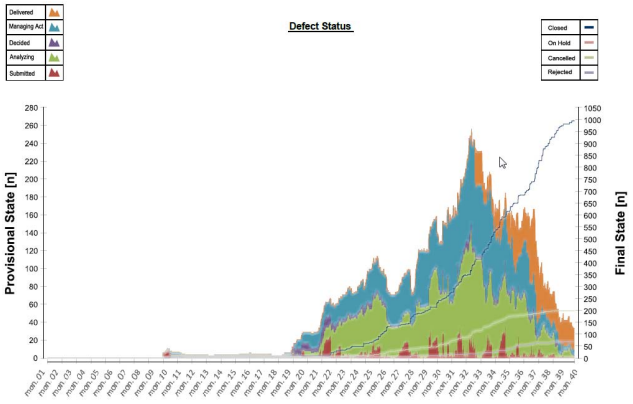

Figure 1.2 Defect distribution during development

The research used the Crosscutting concern technique to illustrate the interaction among the system components [6]. A lot of data of completed projects were analyzed in combining with the experts' knowledge of different develop teams and several research in this direction was analyzed in this research to describe all Key Performance Indicators (KPI) [7] of the system which affect the defect rate and their needed correction effort. Controlling these huge amount of information to be used by the project managers with their decisions is a big challenge. A Design Structure Matrix (DSM) [8] is constructed to include this information. The DSM describes the traceability among the different component in the software and offer a help for the project managers with their decisions. Figure 1.2 illustrates the distribution of software defects during the same development life cycle.

II. Software Estimation Techniques

The Complex software controlling is a very important aspect of investment of the software. The project planning and control the risk analyses, and effort estimation of software effect on the cost of production and the quality demand.

A lot of research in the direction of software cost modeling had be done, since 1965 there are a lot of studies and research of the attributes of software projects be doing. This led to the development of different models start with the late 1960's and early 1970's [9].

Different estimation models with different techniques for effort estimation have been developed and applied in last decades. The widely use of Software leads to increase the complexity and arise the risk of the production. Ensure the required metrics of the software required a better estimation technique to help the project managers with their decision. The estimation models help to offer a help for the mangers to have an estimation for a different range of software development and maintenance activities, needed in different phases of development life cycle, starting with the requirement analysis, architecture, implementation, testing, and maintenance of software. Some mostly used techniques of estimations models [10]:

- Model based Techniques
- Expertise Based Techniques
- Learning Oriented Techniques
- Dynamic Based Techniques
- Regression Based Techniques

Composite Techniques template is used to format your paper and style the text. All margins, column widths, line spaces, and text fonts are prescribed; please do not alter them. You may note peculiarities. For example, the head margin in this template measures proportionately more than is customary. This measurement and others are deliberate, using specifications that anticipate your paper as one part of the entire proceedings, and not as an independent document. Please do not revise any of the current designations.

III. Software Effort Estimation Model

In this research we developed an expert system which helps in effort estimation of industrial software change. The

guidelines to design expert systems [11] is followed in this research as a development procedure. The goal of the thesis is to design a system with main application to support the project managers (PrjM) and process owners with decisions related to software change estimation. Figure 4.1 illustrates the procedure used to build the models. The problem definition and the key performance indicators (KPI) related to the problem are essential for every expert system. The problem definition of the models described in this research are derived from the research goal defined in the research. The Design Structure Matrix (DSM) approach [12] which is represented by the traceability matrix has been used to identify relevant factors of effect (KPIs) of the model and their relations. The problem definition includes its purpose, object of interest, issue and user's point of view are included in DSM approach.

To get the KPIs and the relation among the KPIs a huge data knowledge is needed, the data used to obtain the model are mostly obtaining from:

- Internal sources: The data management system including the change and defect management system in an automotive industry are used where all change specific information is stored. Three project change systems were analyzed with different real automotive life to realize a specific part of a feature.
- Expert knowledge: the project managers and developers are the best experts in the project. The knowledge of the experts was accumulated and listed by several discussions and discussion matrix filled by the experts in order to get a dataset for the expert's knowledge and can be used in the built model.
- External sources: several studies and publications help to get a theoretical knowledge about the system.

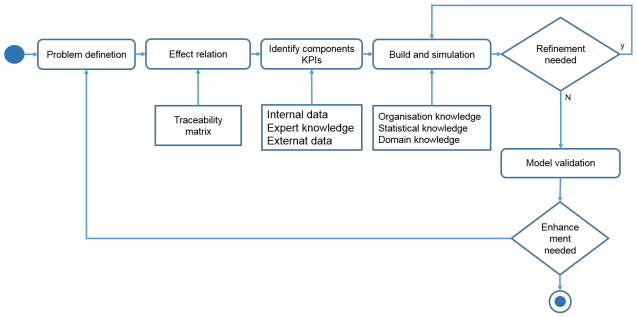

Figure 3.1 model development procedure

The data used for model built and analysis has been made anonymous due to their confidentiality. There is a need for huge for expertise in multiple fields to create and simulate a model, which is a big challenge. A model creation need a deep understanding of the different domains of the model which to be established. And a good knowledge in software engineering of automotive domain is required to present the models in this research.

To have an expert model, a fully understood for the problem is needed including its Key Performance Indicators (KPIs) and how they are related to each other. Here a huge knowledge in the specific automotive work environment is needed to be able to map the problem definition from theory to practice.

A. *Traceability matrix*

DSM is used in this research to represent the developed system elements from the requirement down to the implementation activities. The DSM shows the relation among these elements.

DSM matrix will be constructed in forward process flow depending the across rows way [12]. DSM matrix diagonal elements will describe the effort needed to change the element as a standalone component. The effort of the related elements have a relation with the element in a row, the relation will set as 1 when the element have a relation and 0 when a relation missing. Table 4.1 illustrates the matrix for the system shown in Figure 3.2.

The model is built in a hierarchical design, changing in a component in this system will affect other component which have direct relation with this component, and some other components which haven't direct relations with the changed component could be effected by this change. The traceability matrix will be more complex than in Table 3.1.

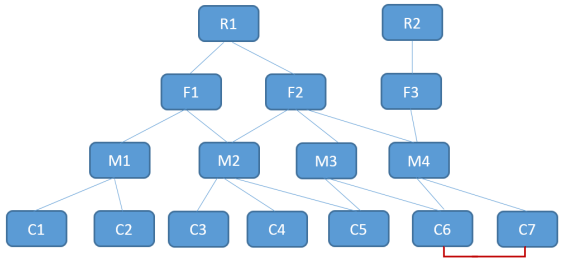

Figure 3.2 software system components

Analyzing the effect relation among the different components in the system is done by the use of crosscutting principles. The traceability matrix for the system shown in Figure 4.2 which include all effective relations in direct and no direct relations among the different components of the system as shown in Table 3.1.

	R1	R2	F1	F2	F3	M1	M2	M3	M4	C1	C2	C3	C4	C5	C6	C7	*	Effort factors
R1	1	0	1	1	0	1	2	1	1	1	1	2	2	2	2	1		KPI_{R1}
R2	0	1	0	0	1	0	0	0	1	0	0	0	0	0	1	1		KPI_{R2}
F1	1	0	1	1	0	1	2	0	0	1	1	1	1	1	0	0		KPI_{F1}
F2	1	0	1	1	1	0	2	1	2	0	0	1	1	2	1	1		KPI_{F2}
F3	0	1	0	1	1	0	0	0	1	0	0	0	0	0	1	1		KPI_{F3}
M1	0	0	1	0	0	1	0	0	0	1	1	0	0	0	0	0		KPI_{M1}
M2	0	0	1	1	0	0	1	1	0	0	0	1	1	2	0	0		KPI_{M2}
M3	0	0	0	2	1	0	1	1	1	0	0	0	0	2	2	0		KPI_{M3}
M4	0	0	0	2	1	0	1	1	1	0	0	0	0	0	2	0		KPI_{M4}
C1	0	0	0	0	0	1	0	0	0	1	1	0	0	0	0	0		KPI_{C1}
C2	0	0	0	0	0	1	0	0	0	1	1	0	0	0	0	0		KPI_{C2}
C3	0	0	0	0	0	0	1	0	0	0	0	1	1	1	0	0		KPI_{C3}
C4	0	0	0	0	0	0	1	0	0	0	0	1	1	1	0	0		KPI_{C4}
C5	0	0	0	0	0	0	1	1	0	0	0	1	1	1	1	1		KPI_{C5}
C6	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	2		KPI_{C6}
C7	0	0	0	0	0	0	0	0	1	0	0	0	0	0	2	1		KPI_{C7}

Table 3.1 Traceability matrix for system

B. *KPI definition*

Analyzing different project lead to define the KPIs of software elements which affect the effort of the development

and maintenance of the software. The research presents two models to define and calibrate these KPIs needed for software development in industrial domain. Figure 4.3 presents a process model of development life cycle. There are different metrics affect the effort of development and need to be deal with them to get a better estimation. The process model classify the development process into four classes

- Process parameters
- Product parameters
- Change parameters
- Test parameters

Analyzing calibrate these classes help to define some of KPIs affect the effort needed for software development and maintenance.

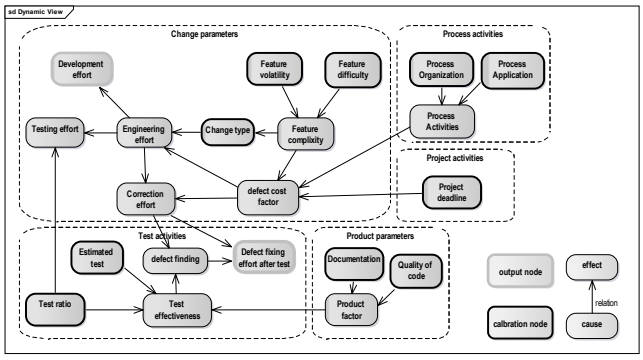

Figure 3.3 Software process model

Figure 3.3 presents a process model to illustrate the defect cost flow of the software development and maintenance phases. The process build depending on using V-Model as development life cycle [13]. The development activities which affect the effort of the development be analyzed and calculated.

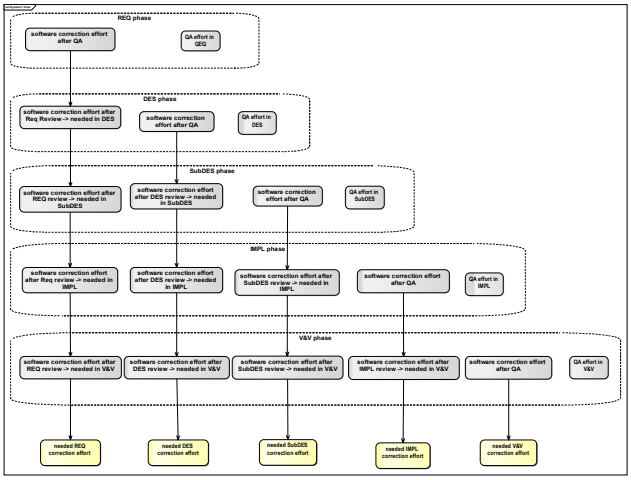

Figure 3.4 development phases

IV. PRACTICAL RESULTS ANALYSIS

The safety critical software developed based on different processes and standards & norms to satisfy their quality level. The quality assurance activities are important attributes effected in the quality of the software [14]. The effort needed for this quality depending on the quality levels needed in the product. The CRQs analyzed in this chapter deals with ASIL C level of software according to ISO-26262 [15]. The quality assurance planed with high effort to ensure the level of the software quality. The effort of different development phases was described in last section. Effort of 2652 hours was needed to complete these CRQs with their specific quality activities. This work offers a model to support the PrjM in their decision by estimation different change requests during the development life cycle. Figure 8.10 illustrates the effort distribution over these CRQs as estimation with the support of the model, and by developer estimation. The last effort values represents the real effort needed to complete each CRQ.

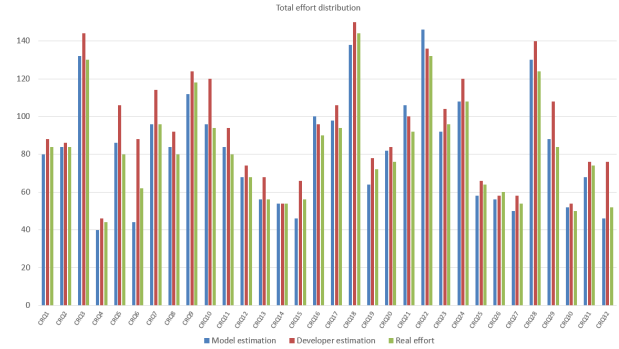

Figure 4.1 total project effort distribution

The range of needed effort for each CRQ varied from 44hours to 144 hours. The complexity of the change has an effect of these 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 won't be discussed here. The ratio of each change be calculated as

*Effort ratio = (Estimated effort) / (Real effort) * 100%* (4.1)

The effort ratio helps to compare the estimated effort by the model against the real effort needed. Figure 4.1 illustrated the effort ratio of the CRQs for the estimated effort by the model and by the developer team including the estimation of the test team.

Figure 4.2 shows the advantage of the support model for the PrjM, the most estimation values of the CRQs are nearly to the real development effort.

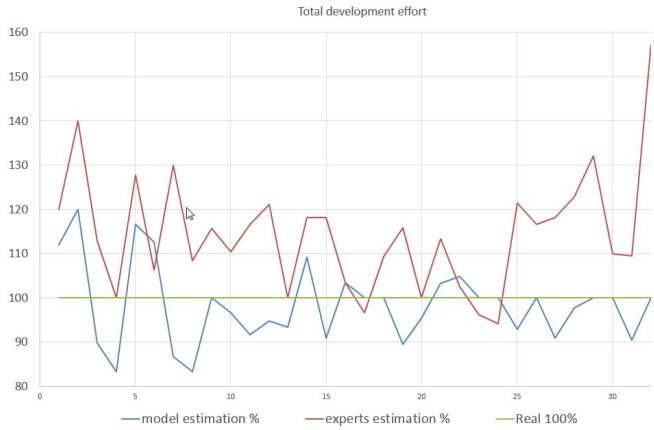

Figure 4.2 effort ratio distribution over the CRQs

The support model describes the system and support the managers with a good overview over the project, and support the PrjM with all KPIs which effected the effort for each CRQ. This system information with all interfaces support the managers with all needed information to have a good estimation. Figure 4.2 shows that the accuracy of the CRQ were between 83 – 120 % based on the real effort needed, that means the model support a help for PrjM in effort estimation with 83% accuracy.

ACKNOWLEDGMENT

Support for this work, by University of Ilmenau, was provided by Computer Science Department/ Software Architectures and Product Lines Group. And the authors would like to thanks in advance for all participated persons as subjects in this work.

REFERENCES

, even if they have been submitted for publication, should be cited as “unpublished” [4]. Papers that have been accepted for publication should be cited as “in press” [5]. Capitalize only

[1] Sneed H.M., Huang S., "Sizing Maintenance Tasks for Web Applications," 11th European Conference on Software Maintenance and Reengineering, CSMR '07, 2007.

[2] JWilliam A. Florac, Software Quality Measurement: A Framework for Counting Problems and Defects, Technical Report, CMU/SEI-92-TR-022, ESC-TR-92-022.

[3] http://www.trinity-cmmi.co.uk/CMMI_Representations.htm, 02-02-2011.

[4] T. Schulz, “A Bayesian Network for Predicting Defect Correction Effort”, 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.

[5] Boehm B.W., Abts C., Chulani S., “Software development cost estimation approaches: A survey”, Annals of Software Engineering 10, pp. 177-205, 2000.

[6] M. Eaddy, T. Zimmermann, K. Sherwood, V. Garg, G. Murphy, N. Nagappan, “Do crosscutting concerns cause defects”, IEEE Transactions on Software Engineering 34, pp 497–515, 2008.

[7] Ernst, M., Hirz, M., and Fabian, J., "The Potential of Key Process/Performance Indicators (KPIs) in Automotive Software Quality Management," SAE Technical Paper 2016-01-0046, 2016, doi:10.4271/2016-01-0046.

[8] M. Revelle, T. Broadbent, and D. Coppit, "Understanding Concerns in Software: Insights Gained from Two Case Studies", workshop. on Program Comprehension, 2005.

[9] Vu Nguyen, Barry Boehm, "A Controlled Experiment in Assessing and Estimating Software Maintenance Tasks", APSEC.2009.49 Conference: 16th Asia Pacific Software Engineering Conference, APSEC 2009.

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

[11] Ngai E.W.T., "Design and development of a fuzzy expert system for hotel selection." Omega (The international journal on Management Science) Vol. 31, pp. 275 – 286, 2003.

[12] TR Browning, "Applying the design structure matrix to system decomposition and integration problems: a review and new directions", Engineering Management, IEEE Transactions on, 2001.

[13] Markus Völter,Thomas Stahl,Jorn Bettin,Arno Haase,Simon Helsen , “Model-Driven Software Development: Technology, Engineering, Management”, ISBN-13:978-0-470-02570-3, 2005.

[14] http://www.cio.bund.de/Web/DE/Architekturen-und-Standards/V-Modell-XT/ vmodell_ xt_node.html.

[15] Road vehicles Functional safety, http://www.iso.org