Scientific Research Publishing

Journal of Software Engineering and Applications, 2025, 18(8), 303-316
https://www.scirp.org/journal/jsea
ISSN Online: 1945-3124
ISSN Print: 1945-3116

Industrial Software Effort Estimation Development and Correction Effort

Sahar A. Abboud[1], Fatma S. Al-Widyan[2], Shahab A. Nadir[3]

[1]Computer Engineering Department, University of Technology, Baghdad, Iraq
[2]School of Mechanical and Mechatronic Engineering, University of Technology, Sydney, Australia
[3]Robert Bosch GmbH, Stuttgart, Germany
Email: Sahar.A.Abboud@uotechnology.edu.iq, fatma.alwidyan@uts.edu.au, shahab.nadir@de.bosch.com

How to cite this paper: Abboud, S.A., Al-Widyan, F.S. and Nadir, S.A. (2025) Industrial Software Effort Estimation Development and Correction Effort. *Journal of Software Engineering and Applications*, **18**, 303-316.
https://doi.org/10.4236/jsea.2025.188018

Received: July 16, 2025
Accepted: August 15, 2025
Published: August 18, 2025

Abstract

The increase in software size and complexity in automotive and industrial domains not only leads to the improvement of new functionality but also to an increase in 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 on the business. This will strongly influence the product costs for the development of a high-quality product. Automotive products have a long lifecycle compared to the development lifecycle, which explains the development efforts put into such fail-safe products. One of the important boundaries for an industrial project is the budget, where changes to any project parameters can easily lead to negative effects on 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 and the automated calculation of effort estimations for the complete development lifecycle. This study offers a new model to improve the effort estimation from multiple perspectives. The newly developed model was successfully evaluated in the automotive domain. The overall accuracy of the effort estimations was improved by 80%.

Keywords

Software Effort, Software Estimation, Defect Estimation

1. Introduction

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 [1].

Figure 1 and Figure 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 has to decide to accept the change or not, by acceptance, an estimation and schedule of the change is necessary with the state "Decided". After the decision and acceptance of 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 [1].

2. Procedure of the Study

2.1. Automotive Product Engineering

Failsafe products are a major motivation in industrial software teams. Here, we describe the field of automotive product engineering to illustrate the boundary conditions for the development of fail-safe automotive products and describe 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), ASPICE 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 [2]. The extensive use of Electronic Control Units (ECUs) in vehicles today has already led to an average of 20 ECUs in smaller and even more than 70 ECUs in upper-class cars [3]. Developing 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.

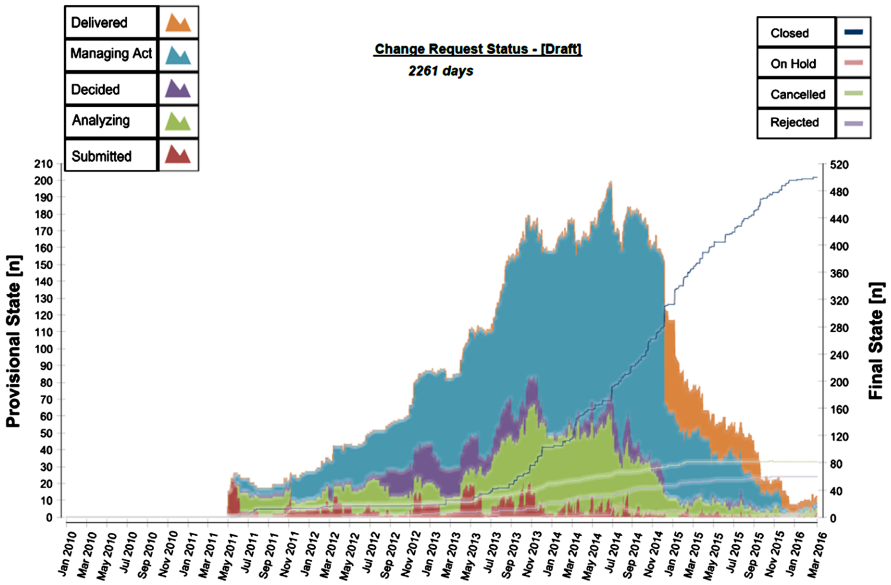

Figure 1. Change request distribution during project lifecycle.

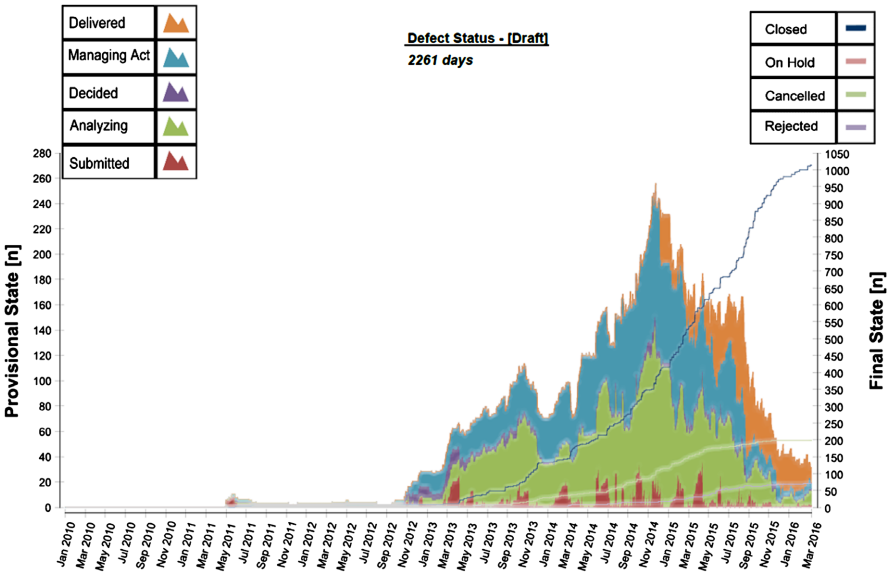

Figure 2. Change request distribution.

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 [4]. 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 3 illustrates the most important phases of development process in an industrial software project,

including the activities needed to manage and control the project [5].

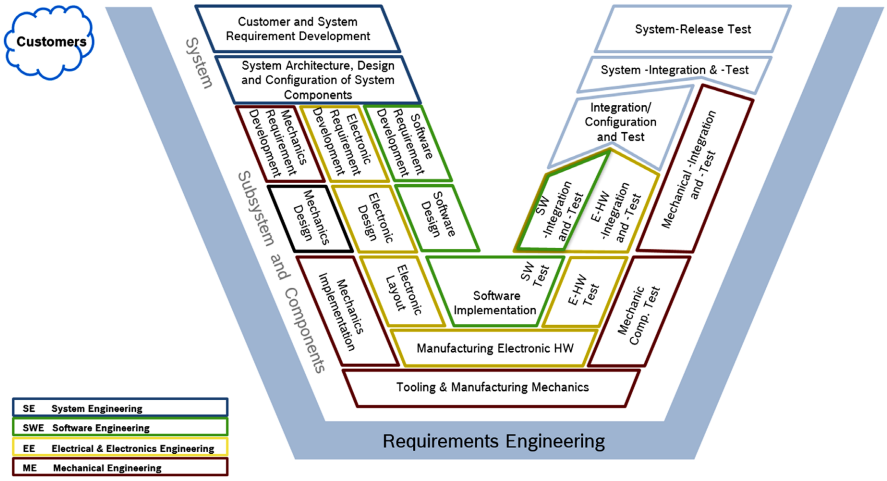

Figure 3. Software development process [1].

2.2. Crosscutting Concerns Matrix Representation

The crosscutting can be represented in matrices, using the concept of a Design Structure Matrix (DSM) [6]. 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 [7].

Different phases of development of life cycle could be represented 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 sorted such that the sources are represented by the row of the matrix, and the targets 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 1** illustrates the dependency matrix of the system, as shown in **Figure 4**.

Table 1. Example dependency matrix and crosscutting matrix.

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	

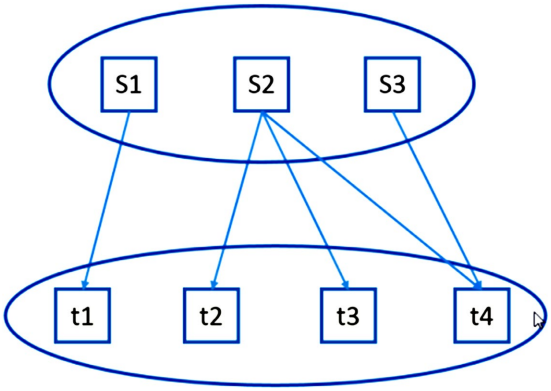

Figure 4. Mapping between elements at two levels of abstraction.

To represent the development phases, a cascading pattern system will be used. For a system shown in **Figure 5**, there are two patterns: between domains A and B, and between Domains B and C.

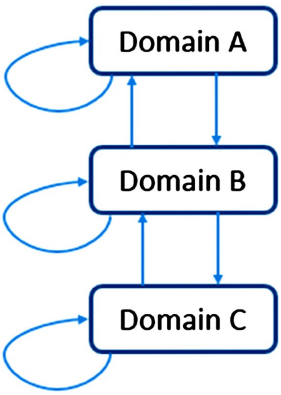

Figure 5. Cascading of traceability pattern.

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 [8]:

$$\exists k \in (1,\cdots,\#\mathrm{B}):(\mathrm{A}[i]\ \mathrm{rel}\ \mathrm{B}[k])\wedge(\mathrm{B}[k]\ \mathrm{rel}\ \mathrm{C}[m])\Rightarrow(\mathrm{A}[i]\ \mathrm{rel}\ \mathrm{C}[m])$$

2.3. DSM Illustration of Software Development Phases

Figure 6 represents a part of a DSM for different levels of development of life cycle. A defect in element M2 in the mapping (F2, M2) could be represented as:

impact (F2, M2) = change ((F2, M2), (M2, C2), (M2, C3)) + 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 impactful 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)).

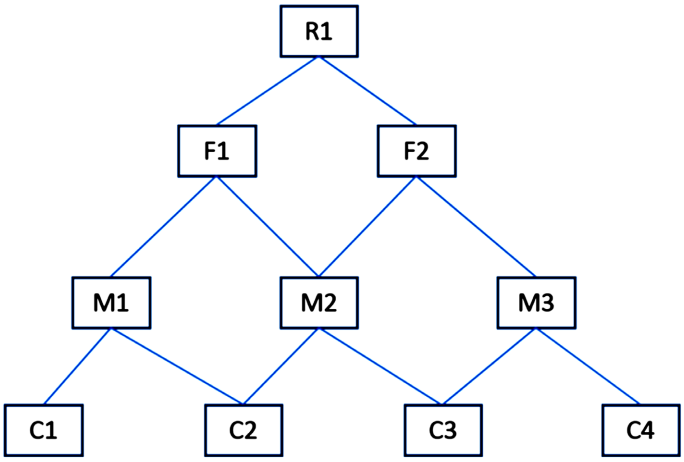

Figure 6. Software system components.

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 illustrates the dependency matrix for the example of Figure 6. There are 4 levels: R, F, M, and C, the cells' value are shaded depending on the relations among the system components.

Table 2. Matrix representation of software system.

	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

With experts' knowledge and the effect of KPIs discussed in this study, a dependency matrix from Table 2 was developed and presented in Table 3. 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.* r1f2 represents the dependency between requirement (R1) and feature (F1) and c3c4 represents the dependency between the artefact (c3) and artefact (c4) in the system shown in Figure 6.

Table 3. Traceability matrix with change effect.

	R	F1	F2	M1	M2	M3	C1	C2	C3	C4
R	r1r1	r1f1	r1f2	r1m1	r1m2	r1m3	r1c1	r1c2	r1c3	r1c4
F1	f1r1	f1f1	f1f2	f1m1	f1m2	f1m3	f1c1	f1c2	f1c3	f1c4
F2	f2r1	f2f1	f2f2	f2m1	f2m2	f2m3	f2c1	f2c2	f2c3	f2c4
M1	m1r1	m1f1	m1f2	m1m1	m1m2	m1m3	m1c1	m1c2	m1c3	m1c4
M2	m2r1	m2f1	m2f2	m2m1	m2m2	m2m3	m2c1	m2c2	m2c3	m2c4
M3	m3r1	m3f1	m3f2	m3m1	m3m2	m3m3	m3c1	m3c2	m3c3	m3c4
C1	c1r1	c1f1	c1f2	c1m1	c1m2	c1m3	c1c1	c1c2	c1c3	c1c4
C2	c2r1	c2f1	c2f2	c2m1	c2m2	c2m3	c2c1	c2c2	c2c3	c2c4
C3	c3r1	c3f1	c3f2	c3m1	c3m2	c3m3	c3c1	c3c2	c3c3	c3c4
C4	c4r1	c4f1	c4f2	c4m1	c4m2	c4m3	c4c1	c4c2	c4c3	c4c4

3. New Estimation Approach

This section presents an approach for project managers and the process owner to support their decisions on 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 is validated and tested. The developed approach, as shown in Figure 7, offers a huge knowledge about the project structure and software process

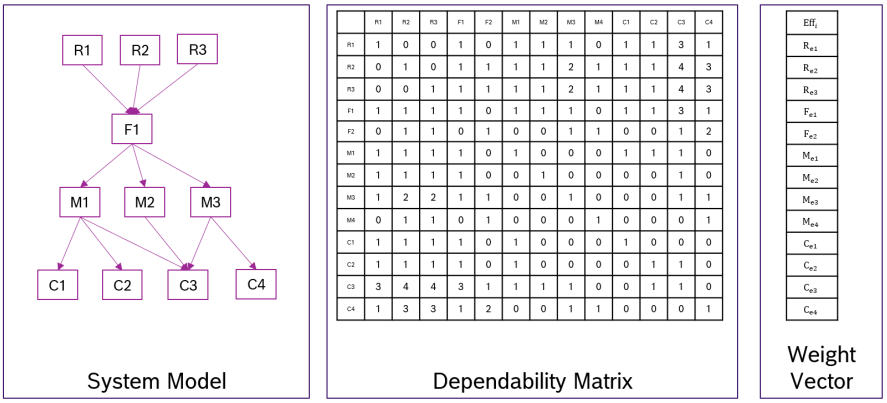

Figure 7. Own approach.

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 another's 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.

Approach Description

This approach has to support project managers with their decisions from multiple domains; therefore, a specification of multiple domain-related specifications is 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:

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 on 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 represented as nodes. The relationships among the nodes are presented as edges connecting these nodes to each other.

The traceability and crosscutting concerns were defined in previous section. A DSM has to represent the system elements from requirements down to implementation activities, as shown in **Figure 6**. 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 [9]. This overview presents a total view of the system. This view helps the decision makers to have better understanding of 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 that are directly related, and other components that are not directly linked to the changed component may also be affected.

2) Dependability matrix

The crosscutting approach is used to describe the system model as a matrix. Each node has a value that describes the relationship between this node and other nodes in the system. This matrix describes 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 depends strongly on the component presented in the column of the matrix, where low value describes 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 effectiveness of a change effort in all affected elements, e.g. the node C2R has a value of four, which presents

a high node value and means that C2 is strongly dependent on R, any change in R could lead to a change in C2. On the other side, the value in node C2M3 is zero that means that C2 is not dependent on F3 change in F3 zero effort for C2.

The DSM will be constructed in an "across rows" fashion [10]. 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 4**.

Table 4. Traceability matrix for system described in **Figure 6**.

	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

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 depends 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 directly affect the effort of change. This effort depends on the process and quality assurance used throughout the development lifecycle.

4) Complexity factor of change

Effort of changes is 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 has to be estimated. Halstead model [11] [12] is used in this study to estimate the complexity. Halstead model has been in use for years to estimate the complexity of software code [13] [14]. Halstead's metrics are 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 present in Equation (1):

$$E = V * D \tag{1}$$

Equation (1) could be used to estimate the effort for development of systems containing several models. The effort of change in a module is estimated as in Equation (2):

$$\text{Effort} = \text{Complexity} \times \text{Weight [hour]} \tag{2}$$

In order to use Halstead model in estimation of software effort in a project, the metrics could be obtained as Equation (3):

$$\text{Complexity} = \left(N_1 + N_2\right) \cdot \log_2\left(N_3 + N_4\right) \tag{3}$$

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 presents the complexity and the connectivity among the different modules in system. The complexity of each module has to be estimated with Halstead model as in Equation (3).

The effort of change in a module is estimated as:

$$\text{Effort} = \sum \text{Complexity} \times \text{Weight}\left[\text{hour}\right] \tag{4}$$

5) Weighing of change

Industrial software development has to ensure the quality and more different metrics in software before releasing the software and the product. Several processes have to be used to cover all the important properties of the product. The development could be done in different phases by different teams. Adapting new change request in a software system could have huge effort more than the effort needed for code implementation. Estimating the effort needed for adapting a change or fixing a defect in software system has to focus on all activities described in the process. This section 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 Equation (4).

This study used some standard methods, e.g. crosscutting concern and Halstead, to build the estimation system, they were adapted to this use. The new KPIs needed for the system were analyzed and described as a new technique in the study. **Figure 5** describes the steps to estimate the effort of changes in software project, the own technique is bordered with a red square.

4. Practical Study

Automotive software development needs several processes and is done based on different standards and norms to satisfy their quality level. Development of lifecycle takes its time, several versions of system are released and tested during the total lifecycle till the start of production. The development phases are planned with their quality assurance needs and the specific milestones. The defect effort will be included in the development effort during the lifecycle. A lot of change requests are developed and defects are fixed through these phases. These requests should be estimated by the developer before the development of the request started. A lot of these requests were estimated with the help of the models built in this study and a traceability matrix for the system was established.

A huge number of Change Requests (CRQs) will be developed throughout this analysis. The analysis deals with a project developed for a real Electric Control Unit (ECU), the data needed for this analysis was accumulated from the change request system. Some of change request data doesn't include all information needed for the analysis. The result of this analysis will be represented in this section, which deals with 50 change request requirements throughout the total development phases. The requirement analysis, design and code implementation will be done by a 13 developers team, the integration and the system test will be done by a different team of 5 developers.

5. Practical Results Analysis

A safety-critical software development is done 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. 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 [8]. 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 the last section. Effort of 2652 hours was needed to complete these CRQs with their specific quality activities. This study offers a model to support the PrjM in their decision by estimating different change requests during the development lifecycle. **Figure 8** illustrates the effort distribution over these CRQs as estimated with the support of the model, and by developer estimation. The last effort value represents the real effort needed to complete each CRQ.

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 describes 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 is calculated as:

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

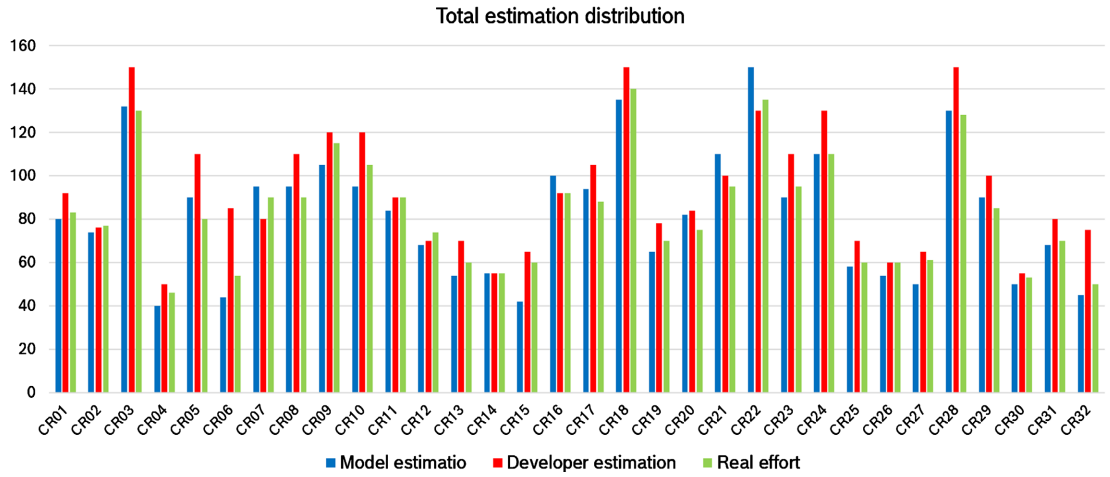

Figure 8. Total project effort distribution (unit: hours).

The effort ratio helps to compare the estimated effort by the model against the real effort needed. **Figure 9** illustrates 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.

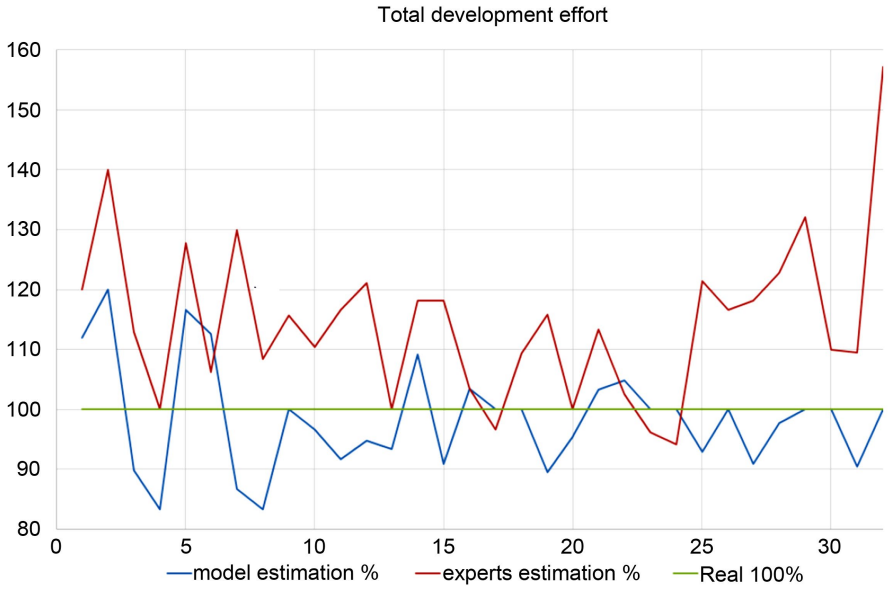

Figure 9. Effort ratio distribution over the CRQs.

Figure 9 shows the advantage of the support model for the PrjM, most of the 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, and supports the PrjM with all KPIs that affected the effort for each CRQ [15]. This system information with all interfaces supports the managers with all needed information to have a good estimation. **Figure 9** shows that the accuracy of the CRQ was between 83% - 120 % based on the real effort needed, which means the model supports helping PrjM in effort estimation with 83% accuracy.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Nadir, S., Streitferdt, D. and Burggraf, C. (2016) Industrial Software Developments Effort Estimation Model. 2016 *International Conference on Computational Science and Computational Intelligence* (*CSCI*), Las Vegas, 15-17 December 2016, 1248-1252. https://doi.org/10.1109/csci.2016.0235

[2] Bajusova, D., Silhavy, P. and Silhavy, R. (2024) Enhancing Software Effort Estimation with Self-Organizing Migration Algorithm: A Comparative Analysis of COCOMO Models. *IEEE Access*, **12**, 67170-67188. https://doi.org/10.1109/access.2024.3399060

[3] Jones, C. (2005) Software Cost Estimating Methods for Large Projects. *The Journal of Defense Software*, 8-12.

[4] Automotive SIG (n.d.) Automotive SPICE Process Reference Model. https://vda-qmc.de/wp-content/uploads/2023/12/Automotive-SPICE-PAM-v40.pdf,%202023

[5] Post, A., Hoenicke, J. and Podelski, A. (2011) Vacuous Real-Time Requirements. 2011 *IEEE* 19*th International Requirements Engineering Conference*, Trento, 29 August-2 September 2011, 153-162. https://doi.org/10.1109/re.2011.6051657

[6] Turner, P. (2021) Econometrics in Practice. Mercury Learning and Information.

[7] Nguyen, V., Steece, B. and Boehm, B.W. (2008) A Constrained Regression Technique for Cocomo Calibration. *Proceedings of the* 2*nd ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, Kaiserslautern, 9-10 October 2008, 213-222. https://doi.org/10.1145/1414004.1414040

[8] Knowledge Plan (2005) Software Productivity Research LLC. Knowledge Plan User's Guide, Version 4.1. SPR.

[9] Engel, A. and Browning, T.R. (2008) Using the Design Structure Matric (DSM) and Architecture Options to Optimize System Adaptability. *Proceedings of the* 10*th International DSM Conference*, Stockholm, 11-12 November 2008, 389-401. https://www.designsociety.org/publication/27454/using_the_design_structure_matrix_dsm_and_architecture_options_to_optimize_system_adaptability

[10] Hossain, S. (2010) Efficiently Computing with Design Structure Matrices. *Proceedings of the* 12*th International DSM Conference*, Cambridge, 22-23 July 2010, 345-358. https://www.designsociety.org/publication/30378/efficiently_computing_with_design_structure_matrices

[11] Trendowicz, A., Heidrich, J., Münch, J., Ishigai, Y., Yokoyama, K. and Kikuchi, N. (2006) Development of a Hybrid Cost Estimation Model in an Iterative Manner. *Proceedings of the* 28*th International Conference on Software Engineering*, Shanghai, 20-28 May 2006, 331-340. https://doi.org/10.1145/1134285.1134332

[12] Abrahamsson, P., Moser, R., Pedrycz, W., Sillitti, A. and Succi, G. (2007) Effort Prediction in Iterative Software Development Processes—Incremental versus Global Prediction Models. *First International Symposium on Empirical Software Engineering and Measurement* (*ESEM* 2007), Madrid, 20-21 September 2007, 344-353. https://doi.org/10.1109/esem.2007.16

[13] Cheng, J. (2001) Virtual Vehicle Sensors Based on Neural Networks Trained Using Data Generated by Simulation Models. Ford Global Technologies, Inc.

[14] Hariprasad, T., Vidhyagaran, G., Seenu, K. and Thirumalai, C. (2017) Software Complexity Analysis Using Halstead Metrics. 2017 *International Conference on Trends in*

Electronics and Informatics (*ICEI*), Tirunelveli, 11-12 May 2017, 1109-1113. https://doi.org/10.1109/icoei.2017.8300883

[15] Singh, C., Sharma, N. and Kumar, N. (2019) Analysis of Software Maintenance Cost Affecting Factors and Estimation Models. *International Journal of Scientific & Technology Research*, **8**, 276-291.