Thesis for the Degree of Doctor of Philosophy

Software Defect Prediction Techniques in Automotive Domain: Evaluation, Selection and Adoption

Rakesh Rana

Department of Computer Science and Engineering
University of Gothenburg

UNIVERSITY OF GOTHENBURG

Gothenburg, Sweden 2015

Cover illustration: Catharina Jerkbrant,
IT Faculty, University of Gothenburg

Software Defect Prediction Techniques in Automotive Domain: Evaluation, Selection and Adoption

rakesh.rana@gu.se

ISBN 978-91-982237-1-2

Technical Report no. 116D
Department of Computer Science and Engineering
Division of Software Engineering
University of Gothenburg
SE-412 96 Gothenburg, Sweden
Telephone + 46 (0) 31-772 1000

Printed in Gothenburg, Sweden 2015
Chalmers Reproservice

To my family:

Mom: *Bimla Devi*
Dad: *Jagdish Chand*
Brother: *Rajesh Rana*
& Sister: *Saroj Thakur*

ABSTRACT

Software is becoming an increasingly important part of automotive product development. While software in automotive domain enables important functionality and innovations, it also requires significant effort for its verification & validation to meet the demands of safety, high quality and reliability. To ensure that the safety and quality demands are meet within the available resource and time - requires efficient planning and control of test resources and continuous reliability assessment. By forecasting the expected number of defects and likely defect inflow profile over software life cycle, defect prediction techniques can be used for effective allocation of limited test resources. These techniques can also help with the assessment of maturity of software before release.

This thesis presents research aimed at improving the use of software defect prediction techniques within the automotive domain. Through a series of empirical studies, different software defect prediction techniques are evaluated for their applicability in this context. The focus of the assessment have been on evaluation of these techniques, how to select the appropriate software reliability growth models and the factors that play important role in their adoption in industry.

The results show that - defect prediction techniques (i) can be effectively used to forecast the expected defect inflow profile (shape and the asymptote); (ii) they are also useful for assessment of the maturity of software before release; (iii) executable models can be used for early reliability assessment by combining fault injection with mutation testing approach; and (iv) a number of factors beyond predictive accuracy such as setup, running, and maintenance costs are important for industrial adoption of machine learning based software defect prediction techniques.

The effective use of software defect prediction techniques and doing early reliability assessment on executable models would allow (i) early planning and efficient use of limited test resources; (ii) reduced development time/ market lead time; and (iii) more robust software in automobiles which make them more intelligent, safe and also highly reliable.

ACKNOWLEDGEMENTS

First of all, I would like to thank my supervisor Assoc. Prof. Miroslaw Staron for his invaluable support, guidance, encouragement and feedback which has helped me pursue my PhD study. I am very grateful to Prof. Jörgen Hansson and Martin Nilsson from Volvo Cars who have supported me and this project at all times. Miroslaw, Jörgen and Martin besides your support, without your efforts the project wouldn't even have existed and I am thankful to you all for giving me the opportunity to work on this project.

Next, I would like to thank my co-supervisors Assist. Prof. Christian Berger and Dr. Fredrik Törner for their advice, guidance and support throughout the project. Special thanks to all my colleagues in the software engineering division for creating a friendly and motivating work environment. Also, I would like to thank Prof. Andrei Sabelfeld for his support and guidance throughout the time he has been my examiner.

I am very grateful to all colleagues at Volvo Cars for helping with the research studies. Same goes out to colleagues from other companies involved in the research studies, especially Wilhelm Meding at Ericsson and Christoffer Höglund at SAAB. I appreciate the time and effort you have put on this project and data provided.

My inexpressible appreciation goes to my family and friends that have always supported me. Special thanks to my parents, brother and sister for their unconditional love and support, you guys have always been and will be the main source of motivation, love and support in my personal and professional life. I would also like to thank Ann Veiderpass, Director of Graduate Studies, Prof. Martin Holmen and everyone at Graduate School, Handelshögskolan, Göteborg for making two years of my Master studies a wonderful experience.

The research presented in this thesis has been carried out in a joint research project financed by the Swedish Governmental Agency of Innovation Systems (VINNOVA) and Volvo Car Group. It has been supported under the FFI programme (VISEE, Project No: DIARIENR: 2011-04438).

INCLUDED PUBLICATIONS

The main contribution of this thesis is based on ten publications, included in chapters 2 to 7. The chapters have been kept as close as possible to the original publications (listed below). Minor modifications regarding language and layout have been made.

I. R. Rana, M. Staron, J. Hansson and M. Nilsson, "Defect Prediction over Software Life Cycle in Automotive Domain", In the proceedings of 9th International Joint Conference on Software Technologies - ICSOFT-EA, Vienna, Austria, 2014

II. R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, and F. Törner, "Comparing between Maximum Likelihood Estimator and Non-Linear Regression estimation procedures for Software Reliability Growth Modelling", In the proceedings of 23rd International Conference on Software Measurement, IWSM-Mensura, Ankara, Turkey, 2013

III. R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, F. Törner, and N. Mellegård, "Evaluation of standard reliability growth models in the context of automotive software systems", In the proceedings of 14th Product-Focused Software Process Improvement, PROFES, Paphos, Cyprus, 2013

IV. R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, F. Törner, W. Meding, and C. Höglund, "Selecting software reliability growth models and improving their predictive accuracy using historical projects data," Published in Journal of Systems and Software, vol. 98, pp. 59–78, Dec. 2014

V. R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, and W. Meding, "Analyzing Defect Inflow Distribution of Large Software Projects", Submitted to a Journal

-This paper is based (revised and extended) on paper "Analysing Defect Inflow Distribution of Automotive Software Projects", Published in the proceedings of 10th International Conference on Predictive Models in Software Engineering, PROMISE, Turin, Italy, 2014

VI. M. Staron, R. Rana, W. Meding, and M. Nilsson, "Consequences of Mispredictions of Software Reliability: A Model and its Industrial Evaluation", In the proceedings of 24nd International Conference

on Software Measurement, IWSM-Mensura, Rotterdam, The Netherlands, 2014

VII. R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, and F. Törner, "Early Verification and Validation According to ISO 26262 by Combining Fault Injection and Mutation Testing," Published in Software Technologies, Springer, 2014, pp. 164–179.

VIII. R. Rana, M. Staron, J. Hansson, M. Nilsson, and F. Törner, "Predicting Pre-Release Defects and Monitoring Quality in Large Software Development: A Case Study from the Automotive Domain", Submitted to a Journal

IX. R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, and W. Meding, "A framework for adoption of machine learning in industry for software defect prediction", In the proceedings of 9th International Joint Conference on Software Technologies, ICSOFT-EA, Vienna, Austria, 2014

X. R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, and W. Meding, "The adoption of machine learning techniques for software defect prediction: An initial industrial validation", In the proceedings of 11th Joint Conference On Knowledge-Based Software Engineering, JCKBSE, Volgograd, Russia, 2014

ADDITIONAL PUBLICATIONS

The following papers and technical report are not included in the thesis:

I. R. Rana, “Defect Prediction & Prevention in Automotive Software Development”, Ph.D. Licentiate Thesis (Technical Report No 108L), Chalmers/ University of Gothenburg, Sweden, 2013

II. R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, and F. Törner, “Evaluating long-term predictive power of standard reliability growth models on automotive systems”, In the proceedings of 24rd IEEE International Symposium on Software Reliability Engineering (ISSRE), Pasadena, USA, 2013

III. R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, and F. Törner, “Increasing Efficiency of ISO 26262 Verification and Validation by Combining Fault Injection and Mutation Testing with Model Based Development”, In the proceedings of 8th International Joint Conference on Software Technologies, ICSOFT-EA, Reykjavík, Iceland, 2013

IV. R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson and F. Törner, “Improving Dependability of Embedded Software Systems using Fault Bypass Modeling”, In the proceedings of Software-Based Methods for Robust Embedded Systems (SOBRES) Workshop at Informatik, Germany, 2013

V. R. Rana, M. Staron, C. Berger, J. Hansson and M. Nilsson, “Analysing Defect Inflow Distribution of Automotive Software Projects”, In the proceedings of 10th International Conference on Predictive Models in Software Engineering, PROMISE, Turin, Italy, 2014

VI. M. Holmén, E. Nivorozhkin, and R. Rana, “Do anti-takeover devices affect the takeover likelihood or the takeover premium?”, Published in The European Journal of Finance, vol. 20, no. 4, pp. 319–340, Jul. 2012

PERSONAL CONTRIBUTION

For all publications above, the first author is the main contributor. In all publications appended in this thesis, I was the main contributor with regard to inception, planning and execution of the research, and writing. The same applies for the additional publications in which I am listed as first author.

For included publication number VI ("Consequences of Mispredictions of Software Reliability: A Model and its Industrial Evaluation"), the contribution of first author (M. Staron) and second author (R. Rana) were at equal level.

TABLE OF CONTENTS

ABBREVIATIONS

SDP	Software Defect Prediction
SRGMs	Software Reliability Growth Models
VCG	Volvo Car Group
ML	Machine Learning

Chapter 1:
Introduction

1 INTRODUCTION

Finding and fixing defects is overall the most expensive activity in embedded software development [1]. Given the size, complexity, time and cost pressures - tracking and predicting quality is a major challenge in automotive software development projects. To meet the demands of high quality and reliability - significant effort is devoted on software V&V (Verification & Validation). Testing the software is an important part of software V&V used for ensuring correct functionality and reliability of software systems; but at the same time software testing is also a resource intensive activity accounting for up to 50% of total software development costs [2] and even more for safety critical software systems. Thus having a good testing strategy is critical for any industry with high software development costs.

Within about 30 years - amount of software in cars went from about zero to more than 10 million lines of code [3]. Premium segment cars today carry about 30-70 ECUs [4], [5] realizing about 2000 individual functions communicating over five different system buses. High use of software also have associated cost implications, today about 50-70% of development costs [6] of software/hardware systems are software costs and about 40% of vehicle development and production costs [3] of modern cars are attributed to electronics and software.

Software defect prediction techniques offer one way of increasing the efficiency and effectiveness of software testing. Predicting expected defect inflow and/or defect prone files/modules allow effective management of limited testing resources. Primarily software defect and reliability measures are used for [6] [7]:

- Software process improvement,
- Planning and controlling testing resources during software development, and
- Evaluating the maturity or release readiness of software before the release date.

In terms of size and complexity, the automotive domain is similar to other embedded software domains - the amount and complexity of software has been growing exponentially. Also with high proportions of development and production costs incurred on software coupled with market competition largely determining the prices, the need for efficient software development and testing process is apparent. Defect prediction techniques can contribute toward the goal of making software testing more effective and efficient.

The purpose of this thesis is to evaluate techniques for software defect prediction in the automotive domain. Further contributions are also made towards selecting the right technique over the life cycle of software development, selection of appropriate reliability models and identification of factors that are important for companies to adopt machine learning based software defect prediction techniques.

Robert Graddy of Hewlett-Packard stated *"software defect data is [the] most important available management information source for software process improvement decisions"* and that "*ignoring defect data can lead to serious consequences for an organization's business*" [6]. The overall goal of this work is to use the defect data to provide insights to software engineers, quality and project managers and assist them in taking decisions on test resource allocations and assessment of maturity of software system under development.

1.1 Frame of Reference

The research presented in this thesis is focused on the evaluation of applicability of software defect prediction techniques in the automotive software domain. As different software defect prediction techniques are based on different basic assumptions, they require specific inputs and can be applied at/or perform best for certain granularity levels. In the frame of reference we describe different techniques commonly used for defect prediction during software development and maintenance and also provide the context of software development specific to automotive domain.

1.1.1 Software Defect Prediction

Software defect, commonly also referred to as bug can be defined as an issue or deficiency in the software product which causes it to perform unexpectedly [8]. IEEE standard 1044, Classification for Software Anomalies provides common vocabulary for terms useful in this context, according to the standard [9]:

- ***Defect:*** *An imperfection or deficiency in a work product where that work product does not meet its requirements or specifications and needs to be either repaired or replaced.*
- ***Error:*** *A human action that produces an incorrect result.*
- ***Failure:*** *(A) Termination of the ability of a product to perform a required function or its inability to perform within previously specified limits Or (B) An event in which a system or system component does not perform a required function within specified limits.*
- ***Fault:*** *A manifestation of an error in software.*

- ***Problem:*** *(A) Difficulty or uncertainty experienced by one or more persons, resulting from an unsatisfactory encounter with a system in use or (B) a negative situation to overcome.*

Since defects in software can lead to malfunctioning of the entire embedded software system, which could in some cases also pose serious risk to health/life (in case of safety critical systems), most organizations developing software aim to release software with no known defects. All defects discovered during V&V activities are reported (documented) in the defect database.

Most organizations maintain defect databases, which can be local to a team, project/product or specific section of an organization. All defects found during verification and validation activities are reports in these databases in a pre-defined format - often with the sole purpose of facilitating their resolution. The database usually provides the platform where different stakeholders within and outside of an organization can:

- Access the information about defect(s) of their interest,
- Add, edit, or update the information related to a given defect,
- Comment, provide expertise or guidance to help resolve the defect, and
- Track the progress of reported defect(s) and monitor defect statistics.

To facilitate the documentation and exchange of information, various attributes are recorded for each reported defect. Some of these attributes are mandatory aimed at providing the basic information pertaining to given defect, while others are optional that provide additional details. The overall goal is to provide information from actor (usually tester) who discovered the defect to actor(s) who will resolve or help resolve it (usually developers). Table 1 provides an example of basic attributes that are usually documented when reporting defects in such a database. Most defect databases contain more attributes and information than listed in Table 1. Other attributes are generally customized for given industrial domain, software development process, and needs of testers and developers for effective exchange of information.

Table 1: Defect attributes available for analysis (usually from bug/defect databases)

Information type	**Attribute (example)**	**Example defect**
Basic information	Unique ID	SWI-1234
	Date & time stamp	12-Jul-201X, 16:33:04
Problem Status	Open/ Resolved/ Closed/ Cancelled	Open
Severity	Major/ Minor/ Insignificant	Minor

Problem Type	Requirement defect/ Design defect/ Code defect	Code defect
Title and Description	Title	Misplaced white pixels on the home screen
	Description	Precondition System is up and running. Action Enter home screen in default mode (all tiles in standard size). Note! This error is only applicable first time the user enters the home screen. Once tiles in the home screen have been expanded, the misplaced pixels disappear. Observation In the middle of the home screen there are some white and grey pixels in line with the top of the phone tile (between the phone tile and the media tile). Expected There should not be any colour deviation of "stains" on the background screen. Probability 100%
Additional Information	Attachments	Attached Log file
	Comments	Update 201X-08-26: When entering and exiting the Settings list, the misplaced pixels reappear, even if tiles in the home screen have previously been expanded (and shrunk back to default tile size again).

There also exist several defect classification schemes that can be used to develop templates for defect reporting that share a well-defined structure. Such pre-defined and shared structure facilitates quantitative analysis of defect reports that can provide useful insights to characterize the development process and also assist in identifying improvement opportunities [10]. Examples of defect classification schemes include orthogonal defect classification [11] developed at IBM, schemes based on IEEE standard classification for software anomalies (IEEE Std. 1044) and a light-weight defect classification scheme [10].

1.1.2 Software Defect Prediction Techniques

Software Defect Prediction (SDP) techniques are used either to classify which modules are defect-prone or to predict the number of defects expected to be found in a software module/project. A number of different techniques have been used for the

purpose of classification[1]/predicting defects; they can be broadly grouped into techniques used for predicting expected number of defects to be found in a given software artefact (Prediction) and techniques that are used to predict if or not a given software artefact is likely to contain a defect (Classification). Figure 1 illustrates commonly used software defect prediction techniques grouped according to the purpose – defect count prediction or defect prone classification.

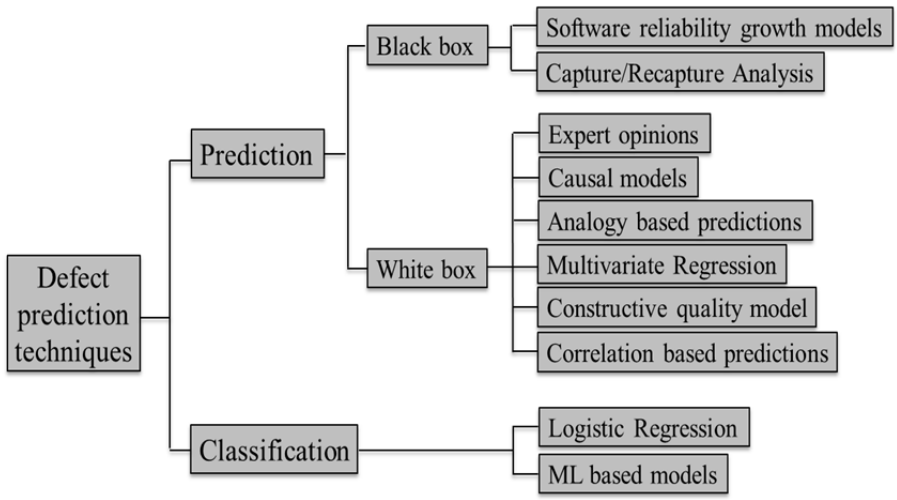

Figure 1: Overview of different software defect prediction techniques

1.1.2.1 Techniques for predicting number of defects

The prediction models may only use number of defects discovered during development and testing without considering other attributes related to the internal structure/design/implementation of the project/product – these are grouped as black box defect prediction techniques. On the other hand defect prediction techniques that use attributes related to process and product e.g. size, complexity, changes are classified under white box techniques.

Software Reliability Growth Models (SRGMs)

SRGMs are mathematical equations used to model the growth of software/system reliability using defect inflow data from the development/testing phase. Appropriate model is selected based on software development/testing process or using empirical

[1] Defect classification can be used to refer either to categorize a defect into classes [11] as in case of defect classification schemes defined in section 1.1.1 or to refer to approach of software defect prediction that involves categorizing modules into fault-prone and non-fault-prone [12]. In this thesis beyond section 1.1.2, unless specified we use the later meaning of defect classification.

evaluations of performance of a sub-set of models on the testing data, which is then used to select appropriate models and make defect forecasts. Applying SRGMs to an on-going project involves fitting mathematical growth models to the observed partial defect inflow data from testing. The fitted model is then used to make final defect count predictions or predicting possible latent defects. SRGMs can be used to model reliability growth over testing period or over the software lifecycle using models such as Rayleigh model. Wood [13] applied eight SRGMs on industrial defect inflow data and found significant correlation between pre-release and post-release defect count. A comparison of SRGMs and their use in practice within consumer electronics embedded software is also presented in study by Almering et al. [14]. Staron and Meding [15] studied defect data from the telecom domain and concluded that models based on moving average provided good predictability for weekly defect predictions; the model was also found to be better than the predictions made using expert opinions [16].

Capture Recapture analysis

This technique of defect prediction is based on analysis of patterns of defects discovered in a given software artefact by independent defect detection activities [17]. Latent defects count (number of defects remaining in a system) is estimated using the overlap among defects identified by independent activities or group of testers. The capture/recapture techniques is also referred to as defect pooling [18]. Briand et al. [19] provide a comprehensive review of capture recapture techniques for software defect count prediction.

Expert opinions

If experts are available, the fastest and easiest method of defect prediction is using them for predictions based on their experience. The drawback of this methodology is its subjective nature and inability to scale down properly at lower levels of granularity. This method can be useful in cases where defect prediction is to be done at project level or large components level and where experts can draw on their experience to make forecasts, but when defect predictions are to be made at lower granularity levels (sub-systems, functions, files etc.), this method does not scale down. Predictions using expert opinions is compared to performance of software reliability growth model in work by Vincent et al. [14].

Causal models

Causal models attempt to establish causal relationships between software process and product attributes with number of defects expected to be found or number of latent defects in the system. Fenton and Neil [20] critique the use of statistical based software defect prediction models for their lack of causal link modelling and proposes use of Bayesian Belief Networks (BBNs). Bayesian Nets have been used to show their applicability for defect forecasting at very early stages of software projects [21].

Analogy based predictions

Analogy based estimation techniques rely upon collection and comparison of variety of metrics between past and current project to identify the most analogous project(s) [22]. For software defect predictions typically size, type of application, complexity of functionality and other parameters are used to identify similar projects to make the estimations. The analysis can be done at project, sub-system or component level.

Multivariate Regression

Regression based models use statistical regression for making defect predictions using a set of software metrics or code change attributes as predictor variables. Multiple linear regression can be used to estimate the number of expected defects in a given software project or modules (sub-systems/functions etc.). A range of software process and product metrics has been used as the independent variables in the regression based models; most common among them are the code complexity metrics and source code evolution (change) metrics. Multiple linear regression is used to model software changes in work by Khoshgoftaar et al. [23] - where a set of software complexity metrics are used as independent variables. Khoshgoftaar et al. [24] used linear regression for predicting program faults, their model also relied on set of code complexity metrics and number of changes to a given module to predict the dependent variable (program faults).

Constructive quality model (COQUALMO)

The constructive quality model [25] is based on the software defect introduction and removal model proposed by Barry Boehm [26] which in turn is analogous to Capers Jones [27] tank and pipe model. The model use expert-determined defect introduction and removal sub-models to construct a quality model refereed as COQUALMO. Under this model, firstly number of non-trivial requirements, design and coding defects introduced are estimated using Defect Introduction (DI) sub-model. The DI sub-model uses software size estimate and other attributes related to project and process (platform, personal etc.). The output of DI sub-model is used as input to the Defect Removal (DR) sub-model together with inputs from defect removal profile levels and software size estimates. The output of DR sub-model is an estimation of number of residual defects per unit size [25].

Correlation based models

Correlation based models also use defect data found during the software development and testing process. Number of defects found at a given phase or iteration during the development process is used to predict number of defects expected to be found in later phases/iterations. Yu, Shen, and Dunsmore [28] evaluated the relationship between defects in earlier and later phases using linear regression model. While regression and correlation based models both use linear

regression (univariate or multivariate linear regression) for defect count prediction, we distinguish between the two as follows:

a. When at least one of the predictor (independent) variable used in the regression model is a direct measure of defect count in an earlier phase or iteration – the model is categorised under correlation based models,
b. While if the prediction model does not use defect count measure of earlier phase/iteration as a predictor variables, the model is classified under regression based models.

1.1.2.2 Techniques for defect classification

The other main approach to defect prediction is software defect classification. These models strive to identify fault-prone software modules using variety of software project and product attributes. Defect classification models are usually applied at lower granularity levels such as file and class level. Software artefacts thus identified as defect prone can be prioritized for more intensive verification and validation activities.

Logistic regression

Logistic regression can be used to classify software modules as defect-prone or not. Similar to multivariate regression, range of process and product metrics are used as predictor variables for the classification of software modules. Logistic regression has been used by Khoshgoftaar and Allen [29] for classifying modules as fault-prone or not. Zimmermann, et al. [30] also used logistic regression to classify file/packages in Eclipse project as defect prone.

Machine learning models

Machine learning based models use algorithms based on statistical methods and data mining techniques that can be used for defect classification/predictions. These methods are similar to regression based methods and use similar input data (independent variables). The key difference being that machine learning based methods are dynamic learning algorithms that tend to improve their performance, as more data is made available. Using code metrics data of projects from NASA IV&V facility Metrics Data Program (MDP), Menzies et al. [31] model based on naïve Bayes predicted with accuracy of 71% (pd, probability of detection) and probability of false alarm (pf) of 25%. Gondra [32] also using NASA project data set (JM1) obtained correct classifications of 72.6% with ANNs and 87.4% with SVMs. Using data from 31 projects from industry and using BNNs Fenton et al. [21] obtained an R^2 of 0.93 between predicted and actual number of defects.

1.1.3 Software Development in Automotive Domain

Automotive software is a form of embedded software, which is defined as the software that resides permanently within a device (hence embedded) and contributes to the device control and functionality. Automotive software is diverse and complex, the major reasons for complexity can be attributed to factors such as [1]:

- Interaction between software and hardware with number of sensors and actuators,
- Expected real time behaviour based on states and events,
- Systems with long life time where embedded software is expected to continue working often without updates, and
- Demands for high reliability and dependability especially for applications that are safety critical.

At the same time the diversity of automotive software range from entertainment related software to safety-critical real time control software [33]. Based on the application area and non-functional requirements these areas can be grouped into five clusters as defined in [33]:

1. Multimedia, telematics, and MMI software: typically soft real-time software which also has to interface with off-board IT, dominated by discrete event/data processing.
2. Body/comfort software: typically soft real-time, discrete-event processing dominates over control programs.
3. Software for safety electronics: hard real-time, discrete event-based, strict safety requirements.
4. Power train and chassis control software: hard real-time, control algorithms dominate over discrete-event processing, strict availability requirements.
5. Infrastructure software: soft and hard real-time, event-based software for management of the IT systems in the vehicle, like software for diagnosis and software updates.

The software development and testing process used is usually influenced by the application area, for example multimedia software may be developed in an agile way with very short iteration time, while infrastructure software developed using suppliers would generally follow “V” model of software development with longer time-span. The focus of this thesis is at the level of full EE (Electronics & Electrical system) development, which constitutes the complete development of software and electronic hardware (Electronic Control Units) in the automotive domain. At full EE level, the projects are referred to as platform projects (within VCG, Volvo Car Group). These projects typically consist of following steps:

- Requirements are set at vehicle level,
- The system is discretised into several functions and logical components,
- Functions and logical components are mapped to individual ECUs,
- Software is implemented and unit tested at ECU level in-house or at supplier, and
- The system is integrated followed by integration, function and acceptance testing.

The following section describes the development process in details.

Most automotive Original Equipment Manufacturers (OEMs) follow Model Driven Development (MDD) and since car platform projects are often large and spread over several months, they are executed in number of iterations. In literature and development standards, software development life cycle in automotive domain has been illustrated as approaches based on V-model [34], [35].

The process followed at each iteration within the production software development phase can be described using a V-model, essentially for each iteration - first the requirements are set or reviewed followed by System Design (functional design and system architecture). Following the system design ECU specifications is done which can also be referred as software design since software is usually designed for specific ECUs and they are generally co-developed, optimized for particular functionality.

Next comes the implementation where designed software is implemented (as code either manually written in object oriented language or auto-generated from a functional model build using some domain specific language (DSL) such as Matlab/Simulink). The implemented code usually undergoes rigorous testing under simulated environment to ensure correct working of intended functionality and fulfilment of desired quality requirements. The testing of software in simulated environment is termed Model-In-Loop testing where different functional models/code is also integrated and tested.

The software code is then integrated within the hardware/ECU, followed by testing in Hardware-In-Loop configuration (for all iterations) and testing within complete vehicle prototypes (for certain iterations). Major types of testing carried out to verify and validate the functionality include unit testing, sub-system integration and testing, system integration and testing, functional and acceptance testing.

Software development in automotive domain mainly follows V-model where left branch (early phase) is dominated by software design and implementation, while

verification and validation is prominent on the right branch. Figure 2 shows the mapping of different stages/phases in automotive software and electronic hardware (ECU) development at the industrial partner (VCG). Requirements at the vehicle level are grouped based on features (or functions), each function has an assigned owner responsible for overlooking the design-to-acceptance of feature in the final product. System designers design the system based on all the functions that are carried over and to be introduced (new). The system is designed such that each ECU is assigned number of logical components that implements the required functionality. Thus there is one to many relationship between function and logical components for example to provide an Anti-lock feature/function, central electronic module (ECU) may have a logical component named Anti-lock control component, while ECU controlling the wheel braking may have another logical component that implements the braking action under anti-lock conditions, which together fulfil the full functionality of Anti-lock braking feature.

It is common in automotive domain that OEMs such as VCG take responsibility of design and acceptance testing of software and hardware at vehicle level, while electronic hardware (ECUs) and base software for the ECUs are developed by their suppliers. While OEMs do implement some of the application level software in-house (generally functions/features that are new and innovative which provide market differentiation to their products), much of the application level software is also sourced through tier-1 and tier-2 suppliers customized to the needs of individual OEMs. Under these conditions access to change metrics is not readily available as the software is developed/customized by supplier and not developed in-house.

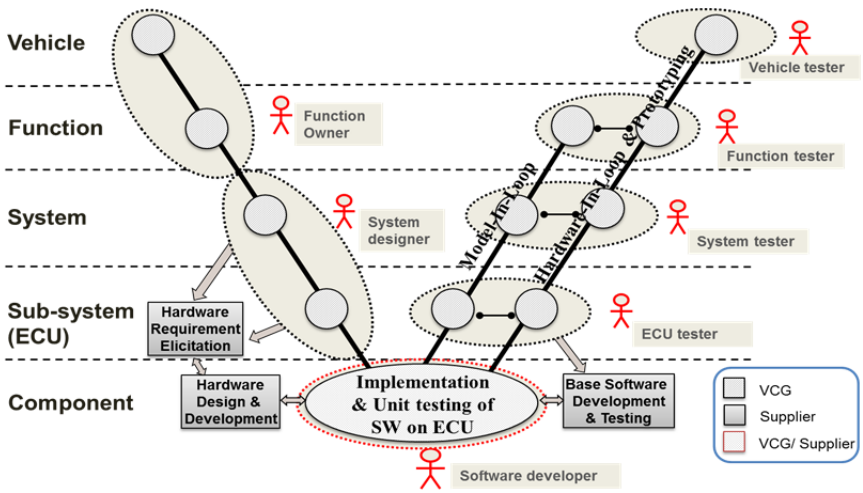

Figure 2: Overview of software development process at VCG

Further software development in the automotive domain often uses combinations of different programming languages and techniques. Use of domain specific languages (DSLs) such as Matlab/Simulink is common among major companies in this sector (both OEMs and their suppliers) and also among other embedded software domains (e.g. aerospace). The production code that runs on a typical ECU today may have mix of code elements that are auto-generated from behavioural models, behavioural model that includes legacy code and hand written code. Figure 3 shows the possible mix of software elements that can be part of production code providing the intended functionality.

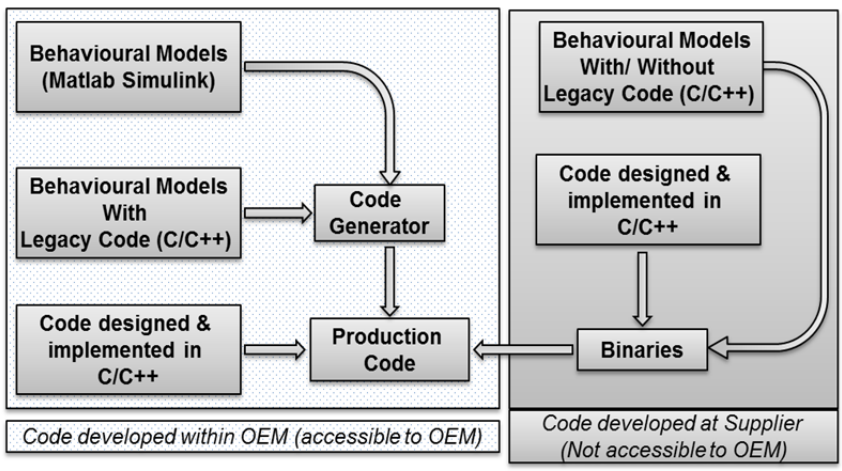

Figure 3: Example combination of software elements in an automotive production software

In such functions and systems obtaining precise and accurate complexity metrics possess challenges for e.g.

- Should we use complexity metrics from behavioural models or from code generated from these models?
- How can we reliably use the complexity metrics for code that is auto-generated and optimized using different Model-to-Code generation tools?
- How can we compare or combine complexity metrics from auto-generated and hand-written code?

Thus software defect prediction techniques based on change and complexity metrics may not always be feasible or easy to apply in certain cases within automotive and other embedded software domains due to:

- Difficulty to calculate the size of software module with good accuracy, thus difficulty calculating and working with defect densities.
- Source code metrics e.g. complexity, size, couplings are readily defined for hand written code, but corresponding metrics for behavioural models are often not validated and in some cases not yet defined.
- Software sourced from suppliers generally is delivered as black-box, which makes it difficult to obtain source code metrics and software evolution information (i.e. change metrics).

In such cases where access to source code and change metrics is unavailable, software defect prediction and defect classification techniques that use code and change metrics may not be feasible.

In the automotive domain, experts have traditionally played their de-facto role in providing their expertise to evaluate the reliability and maturity of software systems/projects. While expert opinions are generally available in large software development companies (like automotive OEMs) and can provide quick predictions, these are based on subjective judgements. Further different experts tend to have different opinions, which make it difficult for managers to take objective decisions. Causal models on the other hand use data from historical projects and characteristics of current project to model and forecast the expected defects found during testing or latent defects. Although causal models provide a more objective and data based predictions, their implementation requires significant effort on first their modelling and secondly on estimating large number of transition probabilities which becomes problematic with increasing number of independent factors and cases where large amount of data is not readily available.

In this thesis the focus has been on software defect prediction techniques that depend on data which is readily available in the automotive software development environment (at OEMs) and models that are easy to implement and use at the companies with low running costs.

1.1.4 Role of Evaluation, Selection and Adoption in Software Engineering

Evaluation

Evaluation of tools and techniques is an important part of software engineering research and development. Software engineers can be divided into two distinct groups, one consisting of those that build and maintain software systems and others

who develop methods and tools for the use of former group [36]. New methods and tools continuously proliferate without much supporting evidence or benefits over existing approaches [36]. Scientifically based and practical approach to evaluation fills this gap by providing the necessary evidence on benefits of given tools/techniques - thus allowing organizations to take informed decisions and helping with the adoption of new tools and methods.

Kitchenham et al. [37] presented methodology for evaluating software engineering methods and tools. The methodology is intended to help organizations plan and execute unbiased and reliable evaluation. It identifies nine methods of evaluation and sets of criteria to help evaluators select an appropriate method. Three important evaluation methods identified under this methodology are: formal experiments, quantitative case studies, and feature analysis exercise; the latter two have been used in this thesis.

Selection

Evaluation can be done for a single tool/method to assess its applicability or performance in a given context or it can be applied for number of similar or competing tools/methods to help select the best one for a given purpose. The evaluation methodology by Kitchenham et al. [37] is comparative, assuming that there exist several alternative ways of performing software engineering task and the main purpose of evaluation is to identify the best alternative for given specific circumstances.

For various decisions to be made with respect to software development, such as which process to use, programming language, tools, or techniques for data analysis - a number of options are usually available. With number of alternative methods and tools available to be applied, selecting appropriate method/tools is a recurring theme in software engineering. Three important methods of selecting the best among competing systems have been described by David et al. [38] as: interactive analysis, ranking & selection and multiple comparison. Ranking & selection has been used in this work specifically for the selection of appropriate software reliability growth models from sub-set of number of competing models.

Adoption

Software managers and practitioners often have to make decisions about which technologies to employ on their projects. An important challenge in making informed decisions about whether to adopt a new technology or not often arise due to lack of objective evidence for the suitability, cost, quality and inherent risks of given technology, tool or method [39]. The transfer of new software engineering techniques and tools from research to industry involves more than just new idea and evidence that it works [40]. Successful technology transfer require good ideas,

generation of evidence of superior characteristics in given contexts, good packaging and support, and careful consideration of the audience for the given technology [40]. Technology acceptance model [41] and technology adoption framework [42] help understand the factors that are important for adoption decisions, while and technology transfer models [43] outlines the process of adoption.

Thus for a comprehensive assessment of a given method, tool or technique – it must be evaluated using scientifically based practical approaches. Also when sets of similar/competing alternatives are present, selecting an appropriate tool/technique for a given purpose is also important. Further when the objective of assessment also includes the transfer to technology from research to industry, factors affecting technology adoption and acceptance are also useful to be studied. We assess software defect prediction techniques in the context of automotive domain mainly in these three dimensions.

1.2 Research Questions

In section 1.1 we reviewed different techniques of software defect predictions, which use different types of input data and can be applied at different stages of software life cycle at different level of granularity. We also provided a brief background on the lifecycle of software development in automotive domain and the development process. Given the specific context of automotive software development using particular development process, tools and other practical constraints when working with third party suppliers – we underline that some of the software defect prediction techniques may or may not be suitable for use in the automotive domain, thus the main goal of this thesis is:

To evaluate how software defect prediction techniques can be effectively applied over the software development life cycle within the automotive domain.

The main research goal addressed in this thesis was broken down to six research questions as following:

RQ1: Which defect prediction techniques are applicable at different stages of software life cycle in automotive domain?

This research question provides a basis for understanding the context of software development life cycle as well as overview of different software defect prediction techniques that are applicable over the life cycle. The answer to this research question is provided in chapter 2. The chapter maps different defect prediction techniques onto when they can be applied over the automotive software life cycle. The chapter also maps SDP techniques to what granularity level they can be applied

and for what purpose. This in turn provides a basis for understanding which defect prediction techniques are applicable at what stage of software development, for what purpose and at what granularity they can be applied.

Next the research question posed was to evaluate how software reliability growth models can be used for making defect count prediction and release readiness assessment in the automotive domain.

RQ2: How software reliability growth models can be used for defect predictions in automotive domain?

The research question two is addressed in chapter 3; in particular we were interested in evaluating applicability and performance of software reliability growth models in the context of automotive software development. Two specific application areas of interest in this chapter where total defect count prediction accuracy and assessment of release readiness. The chapter also deals with practical aspects of applying SRGMs in practice such as which parameter estimation method to use, metrics for evaluating predictive accuracy and how the choice of SRGMs on an on-going project be made more effective.

After evaluation of SRGMs for predicting defect count for live projects, the next research question addressed in the thesis is aimed at understanding the consequences of mispredicting total defect count or the shape of defect inflow.

RQ3: What are the consequences of mispredicting total number of defects and release readiness?

Predictions obtained from any type of abstract models have uncertainties and risk of mispredictions associated with them. When using SRGMs for predicting expected defect count in an on-going project, misprediction scenarios could be over- and under-prediction, early- and late-predictions – and the combination of theses. The research question answered in chapter 4 is about possible consequences of mispredicting scenarios. The consequence or cost model is developed together with industry professionals at Ericsson and VCG and interviews were conducted to reflect which consequences were more or less relevant for two organizational divisions involved in the study.

Following evaluation of SRGMs that have been evaluated in RQ 2 and possible consequences of mispredictions in RQ3, we evaluate another black box defect prediction technique – correlation based models for defect prediction.

RQ4: How can correlation based models be used for defect prediction in automotive domain?

In particular the study was aimed at evaluating if and how can correlation based models be used for defect prediction in the context of EE (Electronics & Electrical system) platform projects in the automotive domain. Since correlation based models also do not require access to process or product metrics (access to source code), they were deemed suitable for the given context. Further correlation based models were also attractive due to their characteristics that they can be applied very easily in the industrial context, requires little (usually available) data and are intuitive to understand by all stakeholders involved in the development and quality assurance of software. The results of this study are presented in chapter 5.

Most SDP techniques either use data from software testing and or use various process and product attributes collected from the analysis of source code and evolution data of software under development. A large part of software developed within the automotive domain uses domain specific modelling languages such as Matlab/Simulink from which code is usually auto-generated. The next research question posed was to investigate how these early software artefacts can be used for reliability evaluations.

RQ5: How to evaluate reliability characteristics of software at early stages of development using only behavioural models?

This research question required a shift from using software testing data for defect count prediction to using behavioural models for making reliability assessment early in the software development process. The framework proposed in this study comprises of utilizing fault injection in combination with mutation testing to assess the efficacy of the test suite. The framework help identify which defects if remained undetected by the available test suite can potentially violate the safety goals according to the ISO 26262 functional safety standard. Thus the proposed framework helps improving the reliability of system by early identification of design and possible implementation defects that can lead to safety goal violations.

Defect prediction and classification models such as machine learning based models that use software evolution and code source based metrics are not feasible in many areas of automotive software development (particularly areas where large part of software is developed using suppliers). Nonetheless other areas within automotive software development that develops software in-house, these defect prediction techniques can be potentially useful. Also in other industrial domains developing embedded software such as telecom - these techniques are particularly attractive. While ML based SDP techniques have been extensively evaluated in research, their adoption in industry is yet far from optimal. The research question posed in chapter 7 is to understand why.

RQ6: What are important factors for industrial adoption of machine learning based defect prediction models?

When evaluating new techniques for software defect prediction, one attribute that is the focus of most research studies is the predictive accuracy. Many research investigations propose new algorithms and prove their usability by showing their superior predictive performance to other algorithms/techniques. Although in industry predictive performance is only one of many attributes that industrial practitioners consider while making adoption related decisions. The last research question addressed in chapter 7 investigates the factors that are considered important by these practitioners for choosing to use new techniques for defect prediction based on machine learning algorithms or adopting new tools based on these new techniques.

1.2.1 Mapping of research questions to chapters and included papers

In this section, the research questions (RQ1 – RQ6) discussed in the previous section are mapped to individual chapters and research questions answered in individual papers included in the thesis. Figure 4, illustrates the overview of this mapping.

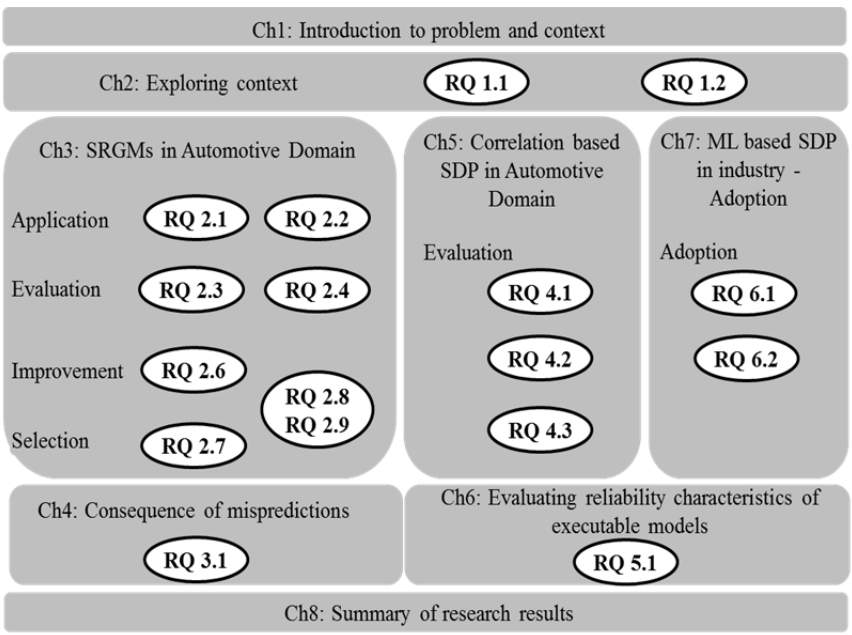

Figure 4: Mapping of main research questions to the chapters in the thesis.

Chapter 1 - Introduction

- *The introduction provides an overview of the goals of this thesis and the context in which the research has been conducted to answer the research questions posed.*

Chapter 2 - Overview of SDP techniques in context of automotive software life cycle

Paper I:

- *RQ 1.1 What are the state of the art methods for software defect predictions?*
- *RQ 1.2 When and at what granularity these methods applicable in the automotive software development life?*

Chapter 3 - SRGMs in automotive domain – selection and evaluation

Paper II:

- *RQ 2.1 Which parameter estimation model is practical for applying SRGMs in automotive domain?*
- *RQ 2.2 What metric to use for assessing the predictive accuracy of SRGM models?*

Paper III:

- *RQ 2.3 Which SRGMs fit best to the defect inflow of large automotive software project?*

Paper IV:

- *RQ 2.4 Which SRGMs are best for assisting with resource allocation?*
- *RQ 2.5 Which SRGMs are best for making release readiness assessment?*
- *RQ 2.6 Does using information from earlier projects improve release readiness assessment?*
- *RQ 2.7 How to make the choice of SRGM (model selection) more effective?*

Paper V:

- *RQ 2.8 Which statistical distribution fit best to the defect inflow from large software projects?*
- *RQ 2.9 How do different information criteria differ for selecting the best distribution fit?*

Chapter 4 - Consequence of mispredictions

Paper VI:

- *RQ 3.1 Given the software quality growth prediction curve, what are the consequences of mispredicting the total number of defects and release readiness?*

Chapter 5 - Correlation based SDP technique in automotive domain - evaluation

Paper VII:

- *RQ 4.1 Is it possible to use defect count in current iteration to predict the defect count in next iteration?*
- *RQ 4.2 Is it possible to predict pre-release defect counts using only defect count data in the intermediate iterations?*
- *RQ 4.3 How can we use correlation based prediction models to identify defect prone modules?*

Chapter 6 - Evaluating reliability characteristics of executable models

Paper VIII:

- *RQ 5.1 How fault injection and mutation testing can be used at model level and how it can be applied within the ISO 26262 verification and validation framework?*

Chapter 7 - Machine learning techniques for SDP in industry - adoption

Paper IX:

- *RQ 6.1 How can we use the technology acceptance and adoption models for developing framework for ML adoption in industry and how to adapt it for software defect prediction?*

Paper X:

- *RQ 6.2 What are the factors that are important for companies to make informed decision to adopt (or not adopt) ML algorithms for the purpose of software defect predictions?*

Chapter 8 – Summary of research results

- *The chapter provides a summary of research results, conclusions and areas of future research.*

1.3 Contributions of the thesis

The main contribution of this thesis comprises of evaluation of software defect prediction techniques in the context of automotive software development. The evaluation is primarily aimed at supporting software developers, testers, quality and project managers to take effective decisions on test resource allocation and assessment of maturity of software under development. In order to structure the research and thesis into sizable pieces, the main research question was broken down into six research questions (RQ1 to RQ6), which are addressed in chapters 2 to 7. The answer to the main research question and conclusions are presented in the final chapter (number 8).

The first research question RQ1 is posed to provide more contextual information about software development life cycle in the automotive domain and mapping software defect prediction techniques with respect to their applicability on this life cycle. Also the purpose different SDP techniques can be used for and at what granularity level they can be applied was also explored under this research question, which is addressed in chapter 2. This study mainly provided:

a. An overview of automotive software development life cycle for large platform EE projects,
b. Mapping of different SDP techniques according to the development phase they can be applied,
c. An overview of input data required, advantages and limitations of mapped SDP,
d. The purpose and application level for different SDP techniques, and
e. Roadmap for increasing the efficiency of defect predictions by using field data.

The next research question, RQ2 addressed in chapter 3 was aimed at evaluating the applicability of software reliability growth models for defect count predictions and software maturity assessment. Related aspects such as the parameter estimation method to use and how the choice of SRGMs could be made more effective were also evaluated in this chapter. The studies included in this chapter provided:

a. Two commonly used maximum likelihood estimation (MLE) and non-linear regression (NLR) were compared and suitability of metric for measurement of predictive accuracy discussed,
b. Commonly applied SRGMs were evaluated on a system level suggesting applicability of SRGMs for the purpose of defect count predictions,
c. A number of SRGMs were further evaluated on defect data from three different companies from embedded software domain and

their predictive performance for defect count forecast and release readiness assessment compared,

d. It was also shown that trend analysis of defect inflow profile on an on-going project can help predict the expected shape of defect inflow for the project which can be useful for selecting the appropriate SRGM, and
e. The statistical distribution family of defect inflow data was also explored where beta distribution was shown to be prominent family among tested distributions.

The results from chapter 2 evaluated the performance of commonly applied SRGMs at system level software and at the level of EE platform project. The question of how to choose the appropriate SRGMs for a given purpose was also addressed.

The next chapter addressed research question RQ3 to answer what are the cost/consequences of making wrong predictions using reliability growth modes. The main outcome of the research presented in this chapter where:

a. Two axis of accuracy of prediction were explicitly identified - (i) the prediction of the asymptote or the total number of defects and (ii) when the total number of defects are discovered or the release readiness, and
b. Different consequences of mispredictions on the two axes were discussed in the study and their impact on organizations was evaluated using case studies at two organizations.

The following research question, RQ4 addressed in chapter 5 was posed primarily to evaluate applicability of correlation based defect prediction models in the context of iterative software development for EE platform projects in the automotive domain. Since correlation based models only needs few attributes as data input and can potentially be used to make predictions early in the development process, they can prove to be potentially useful for industrial practitioners. The answer to research question RQ4 consists of:

a. Total defect count until about half way through the project strongly correlate to the total pre-release defect count suggesting its possible use as early indicator, and
b. It was also shown in the study how correlation based models could be used to identify software modules that may need specific attention.

After evaluating applicability of SRGMs and correlation based models for software defect count predictions and assessment of release readiness, we evaluated specific

opportunity presented in the automotive domain by the use of behavioural models. These models usually developed in domain specific languages such as Matlab/Simulink are detailed at implementation level from which code is commonly auto-generated. Since the development of these models begins early in the development process, using them for reliability analysis can provide early feedback for designers - thus making changes easy to implement and cost efficient. The study presented in this chapter 6 provided:

a. A framework for early identification of design flaws and evaluating the efficacy of test suite to detect potential implementation defects that can potentially violate the safety goals. A combination of fault injection and mutation testing approach is used in the framework, and
b. The framework was also subjected to initial validation to provide a proof-of-concept which encouraging results.

Finally in chapter 7, research question RQ6 is addressed. Machine learning based software defect prediction techniques have found strong support among the research community – a number of techniques have been shown to provide high predictive accuracy, but their adoption in industry has not been widespread. In this chapter the main question of interest was to identify the factors that influence the decision of adoption for such techniques in industry. The study resulted in:

a. A framework for adoption of machine learning based techniques in industry and its adaptation for software defect prediction,
b. The framework identified nine main factors and various sub-attributes that affect the decisions of adoption of new techniques for SDP in the industry, and
c. The initial validation of framework was done at two companies and specific challenges for adoption are also identified.

The results from the study suggested that while predictive accuracy have been the main focus of past research, industrial practitioners were also interested in how the new techniques fit into their existing systems and the magnitude of setup and maintenance costs. The study further discussed how to use the framework where research studies can actively address the concerns on industry practitioner looking to adopt these techniques and how tool vendors can use such information for further development of their tool offerings and marketing of their products. The framework also provides an objective checklist for organizations to evaluate new technologies or to distinguish between two competing techniques or tool offerings.

1.4 Research Methodology

Research methodology describes the systematic process that is undertaken to yield the sought out research results. It outlines the process, steps taken, practices and methods employed to address the questions one wish to explore. Different approaches can be used to meet the objectives of a research, which could be discovery of knowledge and/or solving a specific problem. Possible research models identified by Adrion [44], [45], also identified as research paradigms by Basili [46] are:

- The scientific method
- The engineering method
- The empirical method
- The analytical method

Depending on the questions of interest, different research approach and methods can be employed. While there is no consensus in the field with respect to classification of research techniques [47], a number of approaches and methods are well established in the field, we use research approach classification used by Glass, Vessey and Ramesh [47], they broadly classified research approaches used into descriptive, evaluative and formulative, these are further subcategorized as represented in Table 2.

Table 2: Research approach as classified in [47]

Descriptive	Descriptive system (DS) Descriptive other (DO) Review of literature (DR)
Evaluative	Evaluative-deductive (ED) Evaluative-interpretive (EI) Evaluative-critical (EC) Evaluative-other (EO)
Formulative	Formulative-framework (FF) Formulative-guidelines/standards (FG) Formulative-model (FM) Formulative-process, method, algorithm (FP) Formulative-classification/taxonomy (FT) Formulative-concept (FC)

The studies included in this thesis are classified based on the research approach, refer to Table 4.

1.4.1 Main research types used in this thesis

Research can be classified under various types, which mainly relate to what type of research questions are answered and the process followed. The studies included in

this thesis are classified on the basic types of research using classification provided in Kothari [48], distinguishing between:

- Descriptive Vs. Analytical,
- Fundamental Vs. Applied,
- Quantitative Vs. Qualitative,
- Conceptual Vs. Empirical, and
- Other types of research.

Descriptive Vs. Analytical

The main purpose of descriptive studies is to describe the state of a system, as it is present. In descriptive studies researcher has/exert no control over the variables but only report what has happened or is happening. Descriptive studies are usually used to describe the state or workings of software systems, eco-systems or take the form of review of literature classifying and summarizing the advances in a particular area of interest. On the other hand in case of analytical research, the researcher use facts and information available already and it is analysed to make critical evaluation. According to Glass [45], analytical studies include proposing or using an existing theory or set of axioms, develop that theory deriving results and where possible comparing the results using empirical observations. Analytical studies can be done using correlation/regression analysis. Analytical studies usually use quantitative research methods which is defined as [49]: *"Explaining phenomena by collecting numerical data that are analysed using mathematical based methods (in particular statistics)"*. Both descriptive and analytical research types have been used in various studies included in this thesis.

Fundamental Vs. Applied

Fundamental (or basic/pure) research is mainly concerned with formulating theories and generalization of phenomenon. This type of research is mainly driven to expand the knowledge within a specific research area. On the other hand applied research is mainly driven to solve specific problems with immediate practical implications. The overall research goal of this thesis is applied in nature and thus all studies in this thesis were applied research type.

Quantitative Vs. Qualitative

Quantitative research is based on objective measures and is applicable for phenomenon's that can be expressed in terms of numerical quantities. Quantitative research collect data from number of cases and interesting patterns can be reviled using statistical methods, these methods are not appropriate for gaining deeper understanding of underlying reasons. While quantitative research can be effectively used to evaluate established or proposed theories, it is usually not appropriate for explaining why questions where qualitative research using methods such as case

studies are useful. Quantitative research deals with "when", "where" and "how often"; on the other hand qualitative research generally aims at answering the "why" and "how" questions. In this thesis six out of ten included studies are quantitative in nature while rest are qualitative research type.

Conceptual Vs. Empirical

Research related to abstract idea(s) or theory is usually regarded as conceptual research, while research that relies exclusively on the experience/observations is classified as empirical type. In conceptual analysis an idea or a concept is broken down into its constituent parts to gain better understanding, this research type is more popular in social sciences and philosophy. Empirical research involves collection of data through observations and experiments and it is usually done to test a given hypothesis. Most studies conducted in the course of this thesis can be classified as empirical research.

Other types of research

Types of research not described above are often variations of one or more of above stated research types. Research type can also be classified in other ways such as from the perspective of time, a research can be conducted as one-time study (using cross-sectional data) or it can be conducted over longer time period (longitudinal study).

1.4.2 Research methods mapping to studies included in the thesis

Research methods used in a given study are usually not mutually exclusive; they are generally combined as appropriated by the purpose of research. We use the research methods categories used by Glass, Vessey and Ramesh [47], Table 3 list the research methods listed in [47].

Table 3: Research methods as classified in [47]

AR	Action research	HE	Hermeneutics
CA	Conceptual analysis	ID	Instrument development
CAM	Conceptual analysis/mathematical	LH	Laboratory experiment (human subjects)
CI	Concept implementation (proof of concept)	LR	Literature review/analysis
CS	Case study	LS	Laboratory experiment (software)
DA	Data analysis	MA	Meta-analysis
DI	Discourse analysis	MP	Mathematical proof
ET	Ethnography	PA	Protocol analysis
FE	Field experiment	PH	Phenomenology
FS	Field study	SI	Simulation
GT	Grounded theory	SU	Descriptive/exploratory survey

Table 4 presents the overview of mapping of included studies based on research methodology and methods used.

Table 4: Mapping of research approach, type and methods to papers included in the thesis.

Paper No	Research approach	Research type	Research methods
Paper 1	Descriptive, Review of literature	Descriptive, Applied, Qualitative, Conceptual	Conceptual analysis, Case study, Literature review/analysis
Paper 2	Evaluative-deductive	Analytical, Applied, Quantitative, Empirical	Conceptual analysis/mathematical, Data analysis
Paper 3	Evaluative-deductive	Analytical, Applied, Quantitative, Empirical	Case study, Data analysis
Paper 4	Evaluative-deductive, Formulative-guidelines	Analytical, Applied, Quantitative, Empirical	Case study, Data analysis
Paper 5	Evaluative-interpretive, Formulative-model	Analytical, Applied, Quantitative, Empirical	Case study, Data analysis
Paper 6	Descriptive, Descriptive other	Descriptive, Applied, Qualitative, Conceptual	Conceptual analysis, Action research, Literature review/analysis, Case study
Paper 7	Formulative-framework	Analytical, Applied, Quantitative, Empirical	Case study, Simulation
Paper 8	Evaluative-deductive	Analytical, Applied, Quantitative, Empirical	Case study, Data analysis
Paper 9	Formulative-framework	Descriptive, Applied, Qualitative, Conceptual	Conceptual analysis, Action research, Literature review/analysis, Case study
Paper 10	Evaluative-deductive	Analytical, Applied, Qualitative, Empirical	Case study, Field study

The principle research methods used in the included studies are described next.

Case study

Case studies are empirical studies using either qualitative or quantitative data, these are generally used for exploring projects, activities or assignments [50]. According to Yin [51] *a case study is an empirical investigation of contemporary phenomenon within a real-life context*. Although the level of control is lower compared to experiments in a case study, it has a strong focus on empiricism and thus effective for tracking a specific attribute or establishing relationships between different attributes in real-life situations. The strong emphasis on understanding the context makes case study very suitable for industrial evaluation of software engineering methods and tools [50], while the greatest weakness of this method is the low power of generalizability.

Data Analysis

Data analysis method is used for both quantitative and qualitative research types. In case of quantitative data, the analysis techniques usually include descriptive statistics, correlation analysis, predictive modelling and hypothesis testing [52]. Descriptive statistics help understand the collected data - mean values, standard deviations and various exploratory visual plots aid the researcher in data exploration and visualization. Correlation and predictive models are used to describe relationship between different attributes of a process. Hypothesis testing is typically done to establish if there exists a significant effect of one or more variables on one or more other variables [52].

1.4.3 Mapping research process to experience factory model

The environment, in which research is conducted, commonly referred to as research setting has important consequences for type of research that can be conducted. Research settings for example affect the choice of experimental design, the type of data that can be collected, research methods used and overall goals of conducted research. The research settings used in this thesis can be mapped to the experience factory model described by Basili et al. [53]. The Quality Improvement Paradigm/ Experience Factory Organization setup provides an appropriate setup for conducting applied research in close collaboration with industrial partners, which fits well with the main objectives of this project.

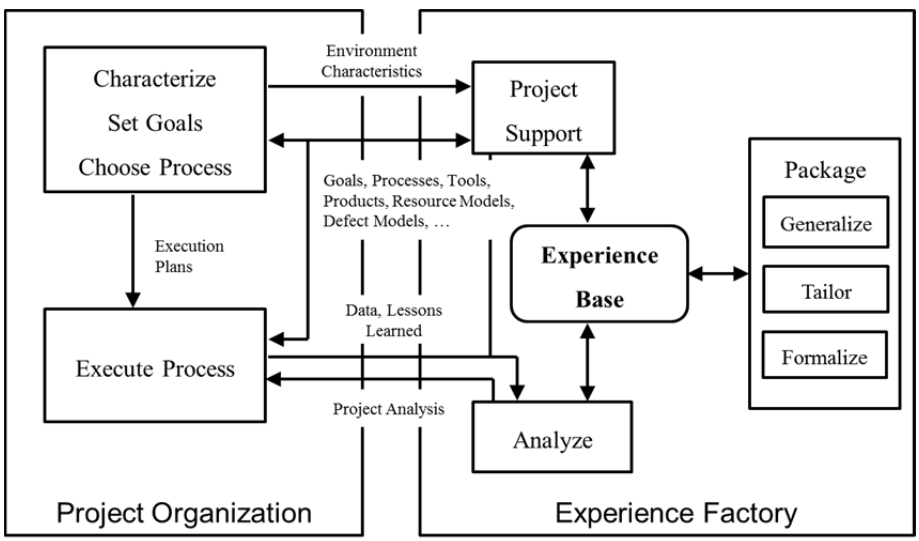

Figure 5: Illustration of Quality Improvement Paradigm/Experience Factory Organization as presented in Basili et al. [51]

The experience factory can be a physically separate and/or logical organization within the project organization. The correspondence between this thesis project setup and experience factory model can be described as follows:

The project organization: This role was fulfilled mainly by the industrial partner of the project i.e. VCG. While some studies included in the thesis have been conducted in co-operation of other companies including Ericsson and Saab electronic defence system, the main partner organization has been VCG. The project was co-supervised by the industrial supervisor at the project organization. The development organization provided the analysis organization with the environment characteristics (context of automotive software development and testing), development data (software defect data), process, quality, resource utilization etc.

Experience factory: The project researcher together with main academic supervisor from the division of software engineering at Chalmers/University of Gothenburg were the primary participants of the analysis organization (experience factory). The analysis organization engaged in data collection (interviews, observations etc.), analysis of collected data and returned direct feedback on the lessons learned to the project organization. The role of support organization was fulfilled by academic and industrial co-supervisors of the project who facilitated the interaction between developers, testers and managers within the project organization and with the efficient retrieval of information.

- The data collected was analysed based on the goals of project, which were setup in the beginning of the project and continually reviewed following the project progress.

The data was mainly used to:

- Characterize and understand, (e.g. the paradigm of software development and verification and validation environment within automotive domain, the constraints and opportunities of the same);
- Evaluate and analyse, (e.g. which SRGMs give the best predictive performance for defect count prediction and for assessment of release readiness);
- Predict and control, (e.g. predicting expected defect count in next iteration based on defect found until a given iteration, using defect count data to identify software modules for further investigation);
- Motivate and improve, (e.g. how to improve the reliability characteristics of software under development using early stage artefacts such as behavioural models).

The package part of experience organization came from the experience of academic and industrial co-supervisors. While the project and analysis organization in the setup of this project were not strictly setup according to the Quality Improvement Paradigm/Experience Factory Organization [53], the implicit setup of the research project was close to this paradigm as outlined above.

1.5 Related Papers

The main part of the thesis excluding current and last chapter comprise of papers and articles that have been published or are under submission. Some of papers that were published but not included in the thesis are listed separately in section 0.

1.5.1 Papers included in the thesis

The following papers were included in the thesis:

Chapter 2: R. Rana, M. Staron, J. Hansson and M. Nilsson, "Defect Prediction over Software Life Cycle in Automotive Domain", In the proceedings of 9th International Joint Conference on Software Technologies - ICSOFT-EA, Vienna, Austria, 2014

Chapter 3: This chapter consist of four papers

R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, and F. Törner, "Comparing between Maximum Likelihood Estimator and Non-Linear Regression estimation procedures for Software Reliability Growth Modelling", In the proceedings of 23rd International Conference on Software Measurement, IWSM-Mensura, Ankara, Turkey, 2013

R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, F. Törner, and N. Mellegård, "Evaluation of standard reliability growth models in the context of automotive software systems", In the proceedings of 14th Product-Focused Software Process Improvement, PROFES, Paphos, Cyprus, 2013

R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, F. Törner, W. Meding, and C. Höglund, "Selecting software reliability growth models and improving their predictive accuracy using historical projects data," Published in Journal of Systems and Software, vol. 98, pp. 59–78, Dec. 2014

R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, and W. Meding, "Analyzing Defect Inflow Distribution of Large Software Projects", Submitted to a Journal

-This paper is based (revised and extended) on paper "Analysing Defect Inflow Distribution of Automotive Software Projects", Published in the proceedings of 10th International Conference on Predictive Models in Software Engineering, PROMISE, Turin, Italy, 2014

Chapter 4: M. Staron, R. Rana, W. Meding, and M. Nilsson, "Consequences of Mispredictions of Software Reliability: A Model and its Industrial Evaluation", In the proceedings of 24nd International Conference on Software Measurement, IWSM-Mensura, Rotterdam, The Netherlands, 2014

Chapter 5: R. Rana, M. Staron, J. Hansson, M. Nilsson, and F. Törner, "Predicting Pre-Release Defects and Monitoring Quality in Large Software Development: A Case Study from the Automotive Domain", Submitted to a Journal

Chapter 6: R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, and F. Törner, "Early Verification and Validation According to ISO 26262 by Combining Fault Injection and Mutation Testing," Published in Software Technologies, Springer, 2014, pp. 164–179.

Chapter 7: This chapter consist of two papers

R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, and W. Meding, "A framework for adoption of machine learning in industry for software defect prediction", In the proceedings of 9th International Joint Conference on Software Technologies, ICSOFT-EA, Vienna, Austria, 2014

R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, and W. Meding, "The adoption of machine learning techniques for software defect prediction: An initial industrial validation", In the proceedings of 11th Joint Conference On Knowledge-Based Software Engineering, JCKBSE, Volgograd, Russia, 2014

1.5.2 Papers not included in the thesis

The following papers and technical report are not included in the thesis:

R. Rana, "Defect Prediction & Prevention in Automotive Software Development", Ph.D. Licentiate Thesis (Technical Report No 108L), Chalmers/ University of Gothenburg, Sweden, 2013

R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, and F. Törner, "Evaluating long-term predictive power of standard reliability growth models on automotive systems", In the proceedings of 24rd IEEE International Symposium on Software Reliability Engineering (ISSRE), Pasadena, USA, 2013

R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, and F. Törner, "Increasing Efficiency of ISO 26262 Verification and Validation by Combining Fault Injection and Mutation Testing with Model Based Development", In the proceedings of 8th International Joint Conference on Software Technologies, ICSOFT-EA, Reykjavík, Iceland, 2013

R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson and F. Törner, "Improving Dependability of Embedded Software Systems using Fault Bypass Modeling", In the proceedings of Software-Based Methods for Robust Embedded Systems (SOBRES) Workshop at Informatik, Germany, 2013

R. Rana, M. Staron, C. Berger, J. Hansson and M. Nilsson, "Analysing Defect Inflow Distribution of Automotive Software Projects", In the proceedings of 10th International Conference on Predictive Models in Software Engineering, PROMISE, Turin, Italy, 2014

M. Holmén, E. Nivorozhkin, and R. Rana, "Do anti-takeover devices affect the takeover likelihood or the takeover premium?", Published in The European Journal of Finance, vol. 20, no. 4, pp. 319–340, Jul. 2012

1.6 Thesis outline

The thesis is structured according to the research questions presented in section 1.2. The first chapter provides the introduction to the thesis, providing an overview of research questions addressed, mapping them to individual chapters and research methodologies used and providing a summary of thesis contributions. In chapter 2 software development life cycle in automotive domain is introduced providing the contextual information and different software defect prediction techniques are mapped to the phases of this life cycle. The next chapter makes an evaluation of applicability of SRGMs for the purpose of defect count predictions and release readiness assessment. The questions on how to apply SRGMs in practice and how appropriate models can be selected are also addressed in the same chapter. The following chapter (number 4) provide details on the possible consequences of mispredicting the defect count or mispredicting the timing of when the expected defects would be found.

Next in chapter 5 we evaluate another black box software defect prediction technique namely the correlation based prediction models. The chapter evaluates if the number of defects found until a given iteration can be used to predict the expected defect count in the next iteration and/or the total pre-release defect count. Chapter 6 shifts the focus on how behavioural models developed under the paradigm of model based development can be used for early identification of potential design defects and to assess the adequacy of test suite to provide early feedback to software designers and testers to improvise the reliability characteristics of software under development. In chapter 7, factors that play an important role in adoption of machine learning based techniques for software defect prediction are identified and validated in the industrial context. The chapter also provides guidelines on the use of adoption framework developed for different purposes. Finally chapter 8 concludes the thesis with the summary of research results and directions for future research.

Chapter 2:
Overview of software defect prediction techniques in context of automotive software life cycle

Included Publication:

I. R. Rana, M. Staron, J. Hansson and M. Nilsson, ***"Defect Prediction over Software Life Cycle in Automotive Domain"***, In the proceedings of 9th International Joint Conference on Software Technologies - ICSOFT-EA, Vienna, Austria, 2014

2 DEFECT PREDICTION OVER SOFTWARE LIFE CYCLE IN AUTOMOTIVE DOMAIN: STATE OF THE ART AND ROAD MAP FOR FUTURE

Abstract— Software today provides an important and vital role in providing the functionality and user experience in automotive domain. With ever increasing size and complexity of software together with high demands on quality and dependability, managing software development process effectively is an important challenge. Methods of software defect predictions provide useful information for optimal resource allocation and release planning; they also help track and model software and system reliability. In this paper we present an overview of defect prediction methods and their applicability in different software lifecycle phases in the automotive domain. Based on the overview and current trends we identify that close monitoring of in-service performance of software based systems will provide useful feedback to software development teams and allow them to develop more robust and user friendly systems.

Keywords— Defect Prediction; Software Life Cycle; Automotive Software; Test Resource Allocation; Release Readiness

2.1 Introduction

Software is now an important part of automotive products, over 2000 software functions running on up to 70 Electronic Control Units (ECUs) provide a range of functionality and services in modern cars [3]. With premium segment cars today carrying about 100 million lines of code, which is more than fighter jets and airliners [54]. Automotive software development projects at full EE (Electronics & Electrical System) level usually are large and span several months. Given the size, complexity, demands on quality and dependability, managing such projects efficiently and tracking the software evolution and quality over the project lifecycle is important.

Defects in software provide observable indicators to track the quality of software project/product under development. Different methods for analysis of software defect data have been developed and evaluated; these methods have also been used to provide a range of benefits such as allowing early planning and allocation of resources to meet the desired goals of projects. The different methods of software defect analysis and predictions have different characteristics. They need different types of input data, are only appropriate to be applied at specific granularity levels and for certain applications. In this paper we summarize the state of the art methods for software defect predictions. We place these methods where these are applicable on the automotive software development life. The methods are mapped to their appropriate level of granularity and application type. We also contend for the position that with technology enabling collection and analysis of in-operations data efficiently will enable software designers and developers to use this information to design more robust and user-friendly features and functions.

2.2 Background

2.2.1 Automotive Software Development Life Cycle

Most automotive Original Equipment Manufacturers (OEMs) follow Model Driven Development (MDD). And since car/platform projects are often large and spread over several months, they are executed in number of iterations. Software development in this domain has been illustrated as variants of iterative development based on spiral process model [55] and approaches based on V-model [34], [35].

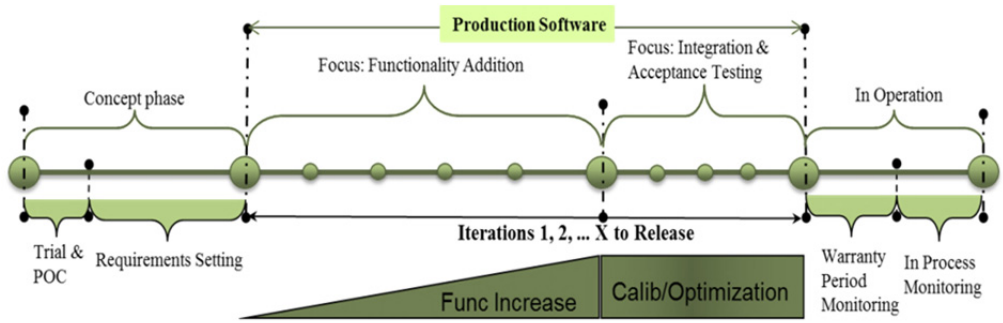

Figure 6: Time Line of Automotive Software Development Life Cycle

The full EE (Electronics & Electrical System) development constitutes the complete development of software and hardware (Electronic Control Units). Different stages of software development process in the automotive domain (illustrated by Figure 6) are:

1. Concept Phase: Where a new functionality is designed and tested on prototypes and Proof of Concept (POC) is demonstrated.
2. Production Software: The main requirements (on vehicle level) are set for the upgrade and new functions approved for market introduction. Software and hardware intended to be included in production automobiles is developed in iterative manner following V-model or spiral development process. The first part of developing production software is dominated by the addition of the new functionality. Unit, integration and function testing are also part of each iteration. In the second part, also carried out in number of iterations – the focus is shifted to integration and acceptance testing.
3. In Operation: Once the new vehicle model is released into the market, the performance of software and hardware is monitored (through diagnostics) during its operation.

2.2.2 Methods for Software Defect Predictions (SDP)

Early estimations of software defects can be used effectively to do better resource planning and allocations. It can also help to track the progress of given software project and improve release planning.

A number of methods have been used for predicting software defects. These methods differ from one another based on the type of input required; the amount of data needed, prediction made and sensitivity to give stable predictions varies. Based on their characteristics, the models can be categorized as:

- Causal Models,
- Using Expert Opinions,
- Analogy Based Predictions,
- Models based on Code and Change Metrics,
- Software Reliability Growth Models (SRGMs), etc.

2.3 Related Work

Expert opinions were used and their performance compared to other data based models in a study by Staron and Meding [15]. Long term predictive power of SRGMs within the automotive domain was studied in authors earlier works [56], [57], demonstrating their usefulness in making defect and reliability predictions.

Number of software metrics based on code characteristics such as size, complexity etc., has been successfully used to classify defect prone software modules or estimate software defect densities. Khoshgoftaar and Allen [29] used logistic regression for classifying modules as fault-prone, while Menzies, Greenwald and Frank [31] used static code attributes to make defect prone forecasts. Methods that use code and change metrics as inputs and use machine learning methods for classification and forecasting have also been studied by Iker Gondra [32] and [58].

Fenton and Neil [20] critique the use of statistical based software defect prediction models for their lack of causal link modelling and proposes use of Bayesian Belief Networks (BBNs). Bayesian Nets have been used to show their applicability for defect forecasting at very early stages of software projects [21]. Our study complements earlier studies in defect predictions by illustrating when different methods of SDP are most appropriate over a software development life cycle.

2.4 Defects Prediction over Automotive Software Life Cycle

Applicability of various methods for software defect predictions over the life cycle phases of automotive software development is represented in Figure 7 and the characteristics of each method are summarized in Table 5. At earliest (concept) phase models that can be applied (given the availability of data about requirements, designs and implementation) are:

- Causal Models
- Using Expert Opinions
- Analogy Based Predictions
- COnstructive QUALity MOdel (COQUALMO)

Models applied at this (concept) phase usually also use information from similar historical projects. Experts in the company draw on their experience to make such forecasts, while data based models require the data to be supplied as inputs. The larger the amount of information available on similar historical projects, the higher is the likelihood for these models to make accurate and stable predictions.

Other SDP methods require data from the development/testing phase. Examples of such methods are:

- Correlation Analysis
- Methods based on Code & Change Metrics
- Software Reliability Growth Models (SRGM)

Correlation analysis models uses number of defects discovered in given iteration (and possibly more attributes) to predict number of defects for following iterations or defect count at project level. Methods based on code and change metrics require access to source code/functional models to measure characteristics such as size, complexity, dependencies etc., which are then used to make the defect proneness classification or forecasting of defect counts/densities. Thus methods based on code and change metrics can only be applied when access to source code/functional models is available. After end of iteration 1, such data is usually available and can be used for making such forecasts. In some cases, which is often the situation in automotive software development, access to source code may be an issue when software is sourced through a sub-supplier. Further since the software development in automotive domain pre-dominantly uses MDD, functional/behavioural model metrics alternatives to code metrics may need to be used where their applicability and performance is currently not well investigated/documented.

Table 5: Software defect prediction models, characteristics and applicability over Automotive SW life cycle

Method	**Input Data Required**	**Advantages and Limitations**
Causal Models	Inputs about estimated size, complexity, qualitative inputs on planned testing and quality requirements.	Causal models biggest advantage is that they can be applied very early in the development process. Possible to analyse what-if scenarios to estimate output quality or level of testing needed to meet desired quality goals.
Expert Opinions	Domain experience (software development, testing and quality assessment).	This is the quickest and most easy way to get the predictions (if experts are available). Uncertainty of predictions is high and forecasts may be subjected to individual biases.
Analogy Based Predictions	Project characteristics and observations from large number of historical projects.	Quick and easy to use, the current project is compared to previous project with most similar characteristics. Evolution of software process, development tool chain may lead to inapplicability or large prediction errors.

COnstructive QUALity MOdel	Software size estimates, product, personal and project attributes; defect removal level.	Can be used to predict cost, schedule or the residual defect density of the software under development. Needs large effort to calibrate the model.
Correlation Analysis	Number of defects found in given iteration; size and test effort estimates can also be used in extended models.	This method needs little data input which is available after each iteration. The method provides easy to use rules that can be quickly applied. The model can also be used to identify modules that show higher/lower levels of defect density and thus allow early interventions.
Regression Models	Software code (or model) metrics as measure of different characteristics of software code/model; Another input can be the change metrics.	Uses actual code/models characteristic metrics which means estimates are made based on data from actual software under development. Can only be applied when code/models are already implemented and access to the source code/model is available. The regression model relationship between input characteristics and output can be difficult to interpret – do not map causal relationship.
Machine Learning based models	Software code (or model) metrics as measure of different characteristics of software code/model; Another input can be the change metrics.	Similar to regression models, these can be used for either classification (defective/not defective) or to estimate defect count/densities. Over time as more data is made available, the models improvise on their predictive accuracy by adjusting their value of parameters (learning by experience). While some models as Decision Trees are easy to understand others may act like a black box (for example Artificial Neural Networks) where their internal working is not explicit.
Software Reliability Growth Models	Defect inflow data of software under development (life cycle model) or software under testing.	Can use defect inflow data to make defect predictions or forecast the reliability of software based system. Reliability growth models are also useful to assess the maturity/release readiness of software close to its release These models need substantial data points to make precise and stable predictions.

SRGMs on the other hand do not need access to source code/model metrics data; these are black-box techniques that only use defect inflow data during development/testing to model the reliability of software systems. While these models can be applied when the software is under development/testing – they need substantial data points (defect inflow) to make stable predictions.

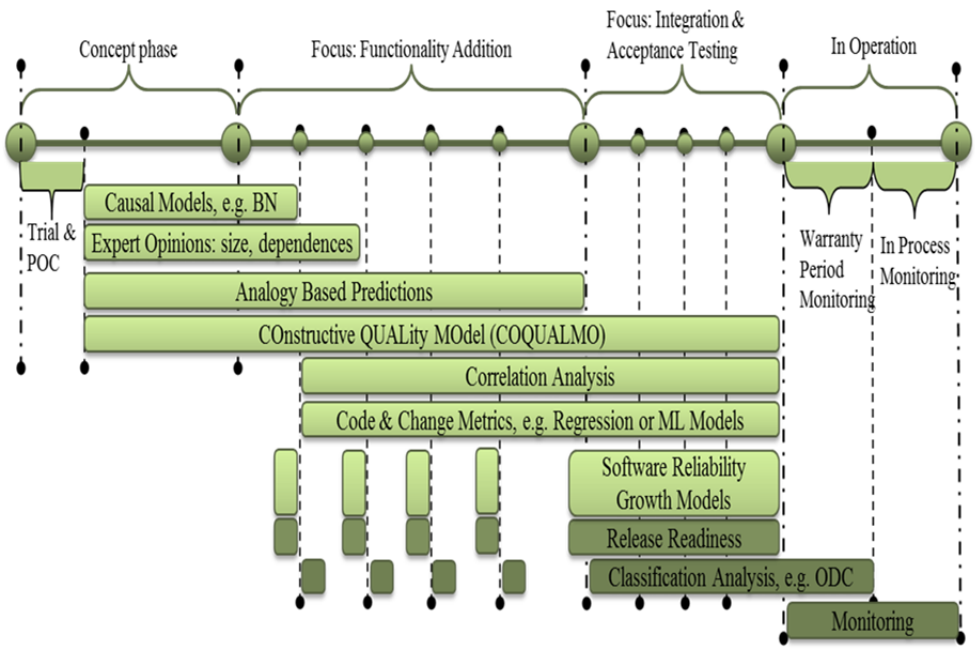

Figure 7: Methods for software defect predictions, applicability over SW life *cycle in automotive domain*

2.5 Analysing defects data over software life cycle

Another characteristic of defect analysis methods is at what level they can be applied. Based on the type of method and input data needed different models provide optimal results at different granularity levels. They can also be used for variety of different purposes. Table 6 summarizes the levels and appropriate applications for each model type. The granularity level at which analysis can be done are:

- Product Level (PL),
- System Level (SL),
- Sub-System level (SSL),

- Functional Unit level (FU),
- MOdule (MO), or at the
- File Level (FL)

And the applications where analysis of software defect data can be useful are:

- Resource Planning and Allocations (RPA),
- What-IF analysis (WIF),
- Release Readiness Assessment (RR),
- Root Cause Analysis (RCA), or for
- Identification of Defect Prone units (IDP)

Table 6: Application level and useful purposes

Model	Application level	Application area
Causal Models	PL, SL, SSL	RPA, WIF
Expert Opinions	PL, SL, SSL, FU	RPA, RRA, RCA, WIF
Analogy Based Predictions	PL, SL, SSL, FU	RPA, RRA
COQUALMO	PL, SL, SSL, FU	RPA
Correlation Analysis	SSL, FU, MO, FL	RRA, IDP, WIF
Regression Models	SSL, FU, MO, FL	RRA, IDP, WIF
ML based models	SSL, FU, MO, FL	RRA, IDP, WIF
SRGMs	PL, SL	RPA, RR, RCA

2.6 Roadmap for increasing efficiency in combining defect prediction methods with field data

In the software domain, the post release monitoring have been fairly limited as software is not regarded same as hardware (software do not degrade or break down with age). Another major reason for lack of monitoring of software in-operation performance in the past has been the un-availability of necessary skills at the service end to retrieve the data and easily feed it back to OEMs for analysis.

But with the advancements of new technology such as high speed data transfer, cloud storage and highly automated computer based diagnostics equipment's available across most of the service points - offers unprecedented opportunity to collect and retrieve the data from the in-operations phase. This feedback information can further enhance the capabilities to design and develop even better, higher quality and safe automotive software.

We contend that the current technologies make it possible for OEMs to collect and analyse in-operations performance of software based systems very much like it has

been the case for hardware components in the past. And much like how such monitoring helped design better hardware components, increase their life and reliability – monitoring the in-operations data of software systems performance will help design more robust, reliable and user friendly software functions in the future.

For example, following and analysing detailed performance metrics of software based system during their life-time operations will:

- Provide in-operations performance metrics of software based systems.
- The qualitative and quantitative robustness and reliability measures from in-operations data will provide input (feedback) for experts and causal models on which software characteristics lead to most reliable performance.
- The current evaluation of performance of code & change metrics SDP models is based on their performance compared to defects found during development and testing. Using in-operations performance data and using code & change metrics data from their source code will help identify "best practices" for the software designers and developers to avoid actions that may lead to sub-optimal performance during operations.
- Insights from the in-operation phase are already used by certain OEMs for effective optimization/calibration. For example functional units such as powertrain use in-operations data to calibrate engines for achieving optimal balance between power and efficiency.
- Active monitoring and analysis of in-operations performance (of software based systems) will help isolate any potential performance related issues and offer quick updates whenever needed. This will further enhance the overall dependability of automotive products.
- Further in future where in-operation monitoring and feedback cycle is shortened would also enable OEMs to identify user satisfaction and usefulness of different features within their cars. This will allow for design and development of more user-friendly features that will benefits the end customers.

2.7 Conclusions

The role and importance of software in automotive domain is rapidly increasing. The size, complexity and value - software provides in modern automotive products is ever increasing and expected to grow further. With trends moving towards more

software enabled functions, autonomous vehicles and active safety systems – ensuring dependability of software based systems is highest priority.

Software development in automotive domain is a long and complex process, various software defect predictions models offer possibilities to predict expected defects thus providing early estimations that are useful for resource planning and allocations, release planning and enabling close monitoring of progress of given project.

In the paper we reviewed that different methods for SDP need different forms of input data, they also have different capabilities and limitations when it comes to their ability to make accurate and stable forecasts. Thus given at what phase of software development life cycle we are in and what kind of data is available, certain defect prediction models may be more appropriate than others and thus should be preferred.

We also show that unlike past, the present technology enables close monitoring, collection and analysis of detailed performance data of software based system during in-operations phase. This data now and in future will be much easy to collect, store, retrieve and analyse. We contend that analysis of such data will lead to development of more robust software based systems that will further help to enhance the reliability of automotive products and aid in development of features that provide superior overall user experience.

Chapter 3:
Software reliability growth models in automotive domain – selection and evaluation

Included Publications:

II. R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, and F. Törner, ***"Comparing between Maximum Likelihood Estimator and Non-Linear Regression estimation procedures for Software Reliability Growth Modelling"***, In the proceedings of 23rd International Conference on Software Measurement, IWSM-Mensura, Ankara, Turkey, 2013

III. R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, F. Törner, and N. Mellegård, ***"Evaluation of standard reliability growth models in the context of automotive software systems"***, In the proceedings of 14th Product-Focused Software Process Improvement, PROFES, Paphos, Cyprus, 2013

IV. R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, F. Törner, W. Meding, and C. Höglund, ***"Selecting software reliability growth models and improving their predictive accuracy using historical projects data",*** Published in Journal of Systems and Software, vol. 98, pp. 59–78, Dec. 2014

V. R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, and W. Meding, ***"Analyzing Defect Inflow Distribution of Large Software Projects"***, Submitted to a Journal
-This paper is based (revised and extended) on paper "Analysing Defect Inflow Distribution of Automotive Software Projects", Published in the proceedings of 10th International Conference on Predictive Models in Software Engineering, PROMISE, Turin, Italy, 2014

3 COMPARING BETWEEN MAXIMUM LIKELIHOOD ESTIMATOR AND NON-LINEAR REGRESSION ESTIMATION PROCEDURES FOR SOFTWARE RELIABILITY GROWTH MODELLING

Abstract— Software Reliability Growth Models (SRGMs) have been used by engineers and managers for tracking and managing the reliability change of software to ensure required standard of quality is achieved before the software is released to the customer. SRGMs can be used during the project to help make testing resource allocation decisions and/ or it can be used after the testing phase to determine the latent faults prediction to assess the maturity of software artefact. A number of SRGMs have been proposed and to apply a given reliability model, defect inflow data is fitted to model equations. Two of the widely known and recommended techniques for parameter estimation are maximum likelihood and method of least squares. In this paper we compare between the two estimation procedures for their usability and applicability in context of SRGMs. We also highlight a couple of practical considerations, reliability practitioners must be aware of when applying SRGMs.

Keywords— Software reliability growth model (SRGM), Asymptote prediction, Predictive relative error (PRE), unbiased, BPRE, Non-linear Regression, Maximum likelihood estimation.

3.1 Introduction

Software is plying an ever increasing role in our day today life. Most of the products and services we consume are now based on software or uses software in certain ways [59]. Over the years the complexity of software artefacts has been growing rapidly, while at the same time the demands for dependability of software systems have also increased. The link between complexity and software faults have been suggested for long, studies as early as 1980s such as [60] suggest that software complexity often affects its reliability. Thus while it is important to keep the complexity of software under check, it is also important to tack and monitor their reliability growth.

Software testing is still the main source of ensuring reliability and quality of software systems. Testing in the area of software products is highly resource intensive exercise, some of the estimates put it around 50% of overall development cost [61]. But testing resource consumptions can be much more resource/cost efficient, if project managers are able to plan testing activities well [62]. Software reliability growth models have been used to estimate the reliability change in software products and use the reliability growth predictions for making testing resource allocation decisions. Since the software can rarely be made fully error free, project managers need to balance costs associated with software testing to cost of fixing bugs after release [63].

Software reliability can be modelled using reliability models, which can be based on Non-Homogeneous Poisson Process (NHPP), Markov process or Bayesian models. One of the major difficulty faced when using Markov and NHPP models is with their parameter estimation [64].

A number of difficulties that may be encountered when applying SRGMs to defect data; in this paper we explore practical considerations when using two types of estimators – Non-Linear Regression and Maximum Likelihood Estimator. We compare between the two and introduce a measure for assessing the predictive power of reliability models. The data used for this study is time-domain failure data for a real-time control system provided in [65] and used in many earlier studies including [66], [67]. In the data 136 faults have been reported with their time between failures (TBF). In the next section we describe the basics of SRGMs and list related work, section 3 outlines the research questions and methodology while the following section (4) is used to present the results. The paper is summarized in section 5 with conclusions and directions for future work.

3.2 Background

3.2.1 SRGMs: Software Reliability Growth Models

Software reliability engineering tends to focus on using engineering techniques for assessing and improving the reliability of software systems during development and post development. A roadmap on the software reliability engineering is presented in [68]. Application of empirical reliability engineering techniques have led to two basic categories, the first class of models called software reliability models (SRMs) are static models that uses attributes of software source code to assess or predict its reliability, while the software reliability growth models (called SRGMs) or the dynamic models generally uses statistical distributions of the defect inflow patterns to estimate/predict the end-product reliability [69]. The SRMs and SRGMs could also be differentiated based on their access to source code which former being a white box models while the latter being black box modelling of software reliability. We focus on SRGMs in this study.

3.2.2 Model Selection

Since the start of reliability modelling within software domain in early 1960s [70], a number of SRGMs have been proposed and evaluated [65]. With so many models which generally differ from one another on their assumptions about underlying software development and testing process, model selection has been a critical challenge. Studies such as by Goel [71] and Musa [72] have shown that different models/families of models are better suited than others for certain applications. A number of studies have also looked into the questions of model selection and suggested various solutions. Sharma [73] recommends that different models should be first compared and evaluated before making a selection. Stringfellow and Andrews [74] presented an empirical method for selecting the SRGM using a proposed criteria and iteratively applying different models, while Khoshgoftaar and Woodcock [75] supports using Akaike Information Criteria (AIC) which is based on the log-likelihood function as a tool to select the best model for given application/data.

3.2.3 Comparing between SRGMs

One common way to understand the differences between different models and their ability to fit and predict given defect data is to do comparative studies. A number of NHPP based SRGMs have been reviewed and compared on their fit and predictive power by Pham [66]. Ullah et al. [76] also present a study comparing eight SRGMs onto large dataset consisting of fifty defect data from industrial and open source projects. Other studies have also evaluated and compared different SRGMs on industrial data, Wood [13] made comparison of eight SRGMs on defect inflow data

and found it correlated with post release defects. Staron and Meding [15] evaluated different SRGMs using large software projects from telecom sector, while in [77] seven SRGMs have been evaluated for their applicability within automotive software projects and long-term predictive power. SRGMs comparison and use in practice for embedded software in consumer electronics is also presented in [14]. Although a number of studies have compared and evaluated different SRGMs within different context, we are still far from making a consensus on how to select SRGMs for given purpose and which models are best for given process characteristics. The situation with different SRGMs comparison is very well summarised by Stephan Kan as: *"Some models sometimes give good results, some are almost universally awful, and none can be trusted to be accurate at all times."* [69].

3.2.4 Parameter Estimation

Two practical and important challenges faced when applying SRGMs in practice/industry are the process to be followed and how to estimate the parameters. IEEE standard 1633: recommended practice on software reliability [78] provides a 13-steps procedure on assessing and predicting the software reliability. The standard also lists three methods commonly used for parameter estimation when using SRGMs as: method of moments, least squares and the maximum likelihood estimation. Maximum likelihood estimation is the recommended approach by the standard and by the various studies introducing new SRGMs [71], [79], [80].

Parameter estimation using Maximum likelihood estimation requires solving sets of simultaneous equations to maximize the likelihood of defect data coming from given function (model equation) to find the parameters. Although MLE fulfils number of important statistical properties of optimal estimator and thus considered the best estimator for large data, unfortunately the set of equations used to find parameters using MLE are very complex and usually need to be solved numerically [13], [72], [81]. This is a practical issue that limits the use of MLE by industrial practitioners who may not be trained to use sophisticated statistical modelling required to use MLE for different SRGMs. The problem of using MLE widely for parameter estimation is further compounded either due to SRGM models with complex log-likelihood functions and cases where MLE does not converge to give unique estimation of unknown parameters. Meyfroyt [82] provides necessary and sufficient conditions for ensuring unique, positive and finite estimation of parameters using MLE for Goel-Okumoto, Yamada S-shaped and Inflection S-shaped models. Use of MLE in industry is further restricted due to lack of commercial tools that can provide reliable MLE parameter estimation for different SRGMs.

On the other hand the least square estimation uses curve fitting to the observed data for making estimation of unknown parameters. Parameters values are estimated for

curve that gives minimum sum of square of errors, i.e. curve that fits best (with respect to sum of squared errors). Given the nature of common SRGMs the least square estimation usually leads to using non-linear regression (NLR) for estimating the unknown parameters. Contrary to MLE, least square estimation is easy to apply, and NLR is often available as standard routine in most commercially available statistical packages.

Wood [13] applied both MLE and least square estimation and found least square predictions to be more stable and better correlated to field data although MLE results were more reasonable. He also noted major difference between the confidence intervals where least square estimates were unsatisfactory, and while MLE confidence interval estimates were realistic they were too wide to make practical conclusions.

It can be safely assumed that statistically MLE is much better parameter prediction procedure than least square, but the least square is much easier and provide consistent results in wider data sets and thus a preferred method of choice by industrial practitioners. Also in certain cases where MLE cannot provide the parameter estimations, least square approach is the natural alternative. Thus the least square estimator/NLR is also used more often than MLE for studies evaluating different SRGMs over large datasets [76], [77]. Given the differences between the two estimators the need to understand the applicability and performance differences of these two estimators is quite apparent.

3.3 Research Context and Method

3.3.1 Research Objectives

In this study we take a look at some of the practical considerations and questions faced by software reliability practitioners. The objective is mainly to document these aspects and mark their importance. Mainly we look at:

- Comparing MLE verses NLR procedure for estimation of unknown SRGM model parameters.
- Assessing predictive accuracy using predicted relative error metric.
- Working with un-grouped data.
- We also comment on reproducibility of earlier studies from literature and provide directions for further research.

3.3.2 SRGMs and Data

In this study we use three of the very early and widely used software reliability models, the SRGMs used and their mean value functions are listed below in Table 7.

The main reason for their selection is their wide familiarity and availability of MLE simultaneous equations. The mean value functions have parameters a, which refers to total number of predicted defects and b, which is generally the shape parameter or growth rate parameter. Parameter β in Inflection S-shaped model is assumed to be 1.2 following the earlier studies [66].

Table 7: Summary of SRGMs used in this study

No	Model Name	Mean Value Function	Ref.
1	Goel-Okumoto (GO)	$m(t) = a(1 - e^{-bt})$	[79]
2	Delayed S-shaped model	$m(t) = a(1 - (1 + bt)e^{-bt})$	[80]
3	Inflection S-shaped model	$m(t) = \frac{a(1 - e^{-bt})}{(1 + \beta e^{-bt})}$	[7]}

The data used for this study is time-domain failure data for a real-time control system provided in [65] and used in many studies including [66], [67]. In the data 136 faults have been reported with their time between failures (TBF). For practical reasons we also assume 136 to be the real asymptote of given data, i.e. actual total number of defects. Cumulative time obtained by successively adding TBF is used for fitting the cumulative distribution functions to different SRGMs. 122 failures are used for fitting the data and making parameter estimates, while the rest are used to evaluate the predictive power.

3.3.3 Data Analysis Techniques

To ensure high reproducibility we list all the data analysis techniques and equations used for analysis in this study with their references.

1. For parameter estimation using least squares we use Non-Linear Regression routine available in statistical package IBM SPSS, the starting values provided were $(a = 120\ and\ b = 0.0001)$ and iterations were done until successive residual errors difference was less than $1.0E - 08$ (default value in SPSS).

2. For parameter estimation using MLE, we use package maxLik, a package for statistical environment R [83]. The optimization method used was Nelder-Mead (NM) and the starting values provided were same as those used for NLR routine.

3. We also compare the parameter estimations obtained by above methods (NLR and MLE using maxLik) with earlier study by Pham [66] using the same data.

4. To make the two estimators comparison even more robust, we further use the non-linear simultaneous equations for getting the analytical solution using MLE. The equations are available for Goel-Okumoto model and Delayed S-shaped model described in [84] and reproduced below:-
   For GO model:
   $\frac{n}{a} = 1 - e^{-bs_n}$ ……..…..……(1)

   $\frac{n}{b} = \sum_1^n s_i + as_n e^{-bs_n}$ ……..……(2)

   For Delayed S-shaped model
   $\frac{n}{a} = 1 - (1 + bs_n)e^{-bs_n}$ ……(3)

   $\frac{2n}{b} = \sum_1^n s_i + abs_n{}^2 e^{-bs_n}$ ……..……(4)

   Where n represents number of failures reported; time between failures is represented as $\{t\_k;\ k = 1,2,\ldots,n\}$ and where time to k^{th} failure is given by $s_k = \sum_1^n t_k$; for details refer to [84].

   Equations (1) & (2) or (3) & (4) can be solved simultaneously to obtain the point estimates of parameters $a\ and\ b$. We used Matlab fsolve to solve system of non-linear equations given above.

5. To make comparison of asymptote prediction accuracy, we use the metric Predicted Relative Error (PRE), which is described in the IEEE standard 1633 and also used in earlier studies as measure of prediction accuracy [76].

   PRE is defined as ratio between predicted error (predicted minus the actual asymptote) to the predicted number of failures.
   $PRE = \frac{(Predicted - Actual)}{Predicted}$ ……..……(5)

   PRE resolves a common problem with using relative error for comparing between different models prediction, the relative error is the ratio of prediction error over actual value and thus if the predicted value is much larger (in multiples) than the actual value, relative error can be greater than 100%. PRE provides a comparative scale between [-1 1] or [-100% 100%], where value close to zero means better predictive accuracy and closer to +/- 100% is as worse prediction as it can get.

Although we identify one major problem with PRE, which is: It provides asymmetric value based on over or under prediction. The problem can be easily understood using a simple example.

Let us assume actual value be a and case1: the predicted value is 20% higher than actual i.e. 1.2a; for case2: the predicted value is 20% lower than actual (i.e. -20% of actual or 0.8a).

Now applying PRE to case1 and case2, gives PRE values:-

$PRE\ (case1)$: $\frac{(1.2a-a)}{1.2a} = \frac{0.2}{1.2} = 0.16667\ or\ 16.667\%$,

While for

$$PRE\ (case2): \frac{(0.8a-a)}{0.8a} = \frac{-0.2}{0.8} = -0.25\ or -25.00\%$$

To make PRE symmetric and thus give consistent value for over and under estimation we define BPRE, referring to Balanced Predicted Relative Error, as follows (equation (6)):

$$BPRE = \frac{Predicted - Actual}{\eta * Predicted + (1-\eta)(2*Actual - Predicted)},$$

$$where\ \eta = \begin{cases} 1\ if\ Predicted > Actual \\ 0\ if\ Predicted < Actual \end{cases} \quad \text{.....(6)}$$

Now applying above defined BPRE to same $case1\ and\ case2$, gives BPRE values:-

$PRE\ (case1)$: $\frac{(0.8a-a)}{0.8a} = \frac{-0.2}{0.8} = -0.25\ or -25.00\%$, and

$$PRE\ (case2): \frac{(0.8a-a)}{0.8a} = \frac{-0.2}{0.8} = -0.25\ or -25.00\%$$

Miyazaki et al. [85] defined a balanced relative error metric R_i, also referred as Balanced Relative Error, BREbias defined as given by equation (7):

$$R_i\ or\ BRE_{bias} = \frac{(Actual-Predicted)}{(min(Actual,Predicted)} \quad \text{.....(7)}$$

Our metric BPRE is similar to R_i, but different in the sense that while R_i is unbounded on both sides, BPRE is bounded [-0.5, 1) which is useful to make comparisons when deviations are particularly large compared to actual values.

6. To compare the model fitting to data for both fit and predicted values, we use another widely used metric, Mean Square Error (MSE). Mean square error measures the average deviations

between the predicted and actual values [86], thus a measure of fit, it is given by equation (8):

$$MSE = \frac{\sum_1^n (a_i - p_i)}{k-q} \quad \text{.....(8)}$$

Where a_i is actual values, p_i predicted values for data set of size k and q is the number of parameters.

3.4 Results

3.4.1 Parameter estimation using MLE and NLR estimation

Parameter estimation using maximum likelihood and non-liner regression procedure are summarised in Table 8. The table also provides comparison of parameters values obtained in study using same data by Pham [66] and also by solving MLE simultaneous equations provided in [86].

Table 8: Comparing Parameters With Different Estimators

Asymptote	**MLE**	**NLR**	**Pham**	**Equation**
Goel-Okumoto	132	114.05	125	139.37
DelayedS	132	103.33	140	125.16
InflectionS	132	107.60	135.5	
Growth Rate	**MLE**	**NLR**	**Pham**	**Equation**
Goel-Okumoto	3.80E-05	6.07E-05	6.00E-05	3.65E-05
DelayedS	9.73E-05	1.66E-05	7.00E-05	9.76E-05
InflectionS	5.79E-05	1.07E-05	7.00E-05	

Form Table 8 it can be observed that the asymptote (a, total number of predicted defects/failures) predictions obtained in this study using maximum likelihood estimator utilizing package maxLik gives very consistent results for all three models. While the asymptote predictions using non–linear regression routine (NLR) varies much more with minimum prediction being 103 for Delayed S-shaped model and 114 for GO model. It is further interesting to note that significant differences are also observed between our predictions using (MLE) and values obtained by earlier study by Pham, although in both case the estimator used is the same (MLE). The difference observed here may be attributed to difference in tools used or the starting values predicted. Given that the tool used and starting values details are not available for earlier study, it is difficult to verify the source of this observed difference.

Predictions for growth rate parameter (b) with different estimators are also listed in Table 8. While there are variations between different models growth rates obtained

in this study using MLE and NLR. The growth rate is predicted to have highest value for Delayed S-shaped model and lowest for GO model using both (MLE & NLR) estimators in our study. The growth rates predicted in Pham study are closer to each other. It can also be noted that for both asymptote and growth rate, our estimates using MLE are very close to the parameters estimates obtained using MLE simultaneous equations described earlier.

The fitting of predicted models using different estimators to actual data is also represented in Figure 8, Figure 9 and Figure 10.

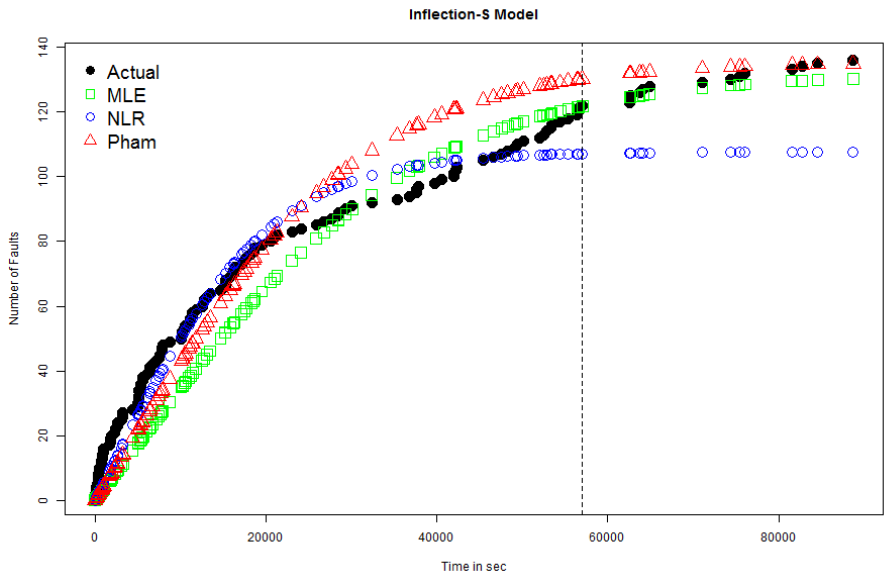

Figure 8: Goel-Okumoto model fitting to data with different estimators

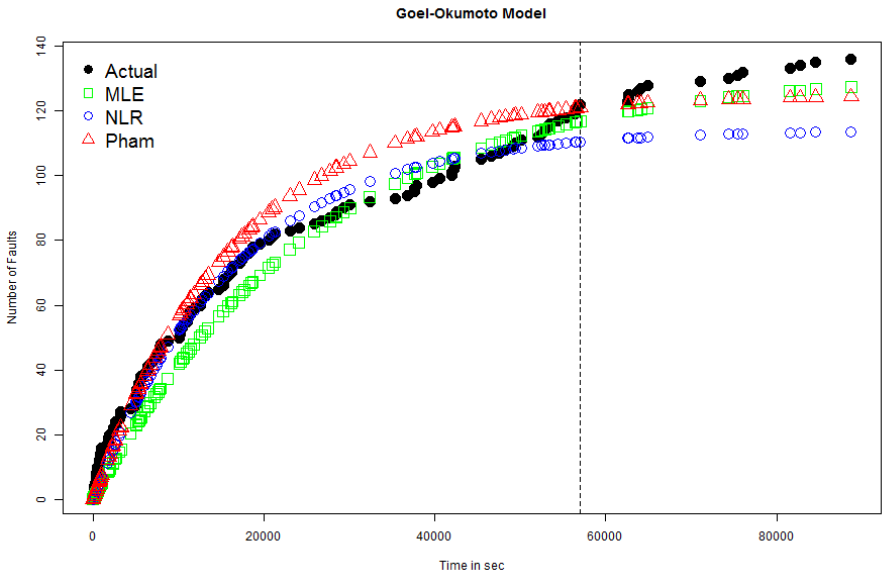

Figure 9: Delayed S-shaped model fitting to data with different estimators

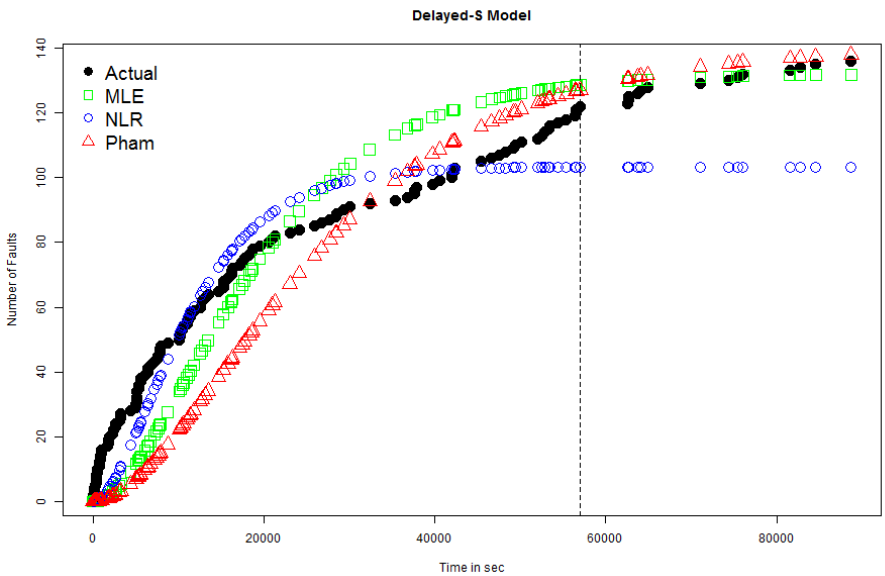

Figure 10: Inflection S-shaped model fitting to data with different estimators

3.4.2 Predictive Accuracy using Predicted Relative Error (PRE) and unbiased PRE (BPRE)

We now compare the predictive accuracy of asymptote values obtained using MLE estimator to NLR estimators.

Table 9: PRE and BPRE for different estimators and models

Asymptote, PRE	**MLE**	**NLR**	**Pham**
Goel-Okumoto	-3.0%	-19.2%	-8.8%
DelayedS	-3.0%	-31.6%	2.9%
InflectionS	-3.0%	-26.4%	-0.4%
Asymptote, BPRE	**MLE**	**NLR**	**Pham**
Goel-Okumoto	-2.9%	-13.9%	-7.5%
DelayedS	-2.9%	-19.4%	2.9%
InflectionS	-2.9%	-17.3%	-0.4%

It is interesting to note from Table 9 that all but one estimate under predicts for given dataset. Using PRE and BPRE values for same parameter predictions we can also see that BPRE gives better and more accurate representation of undervalued asymptote prediction as described in the section 3. The BPRE values for asymptote predictions using MLE and NLR are also presented below in Figure 11. We also add two more models using NLR procedure to make further check.

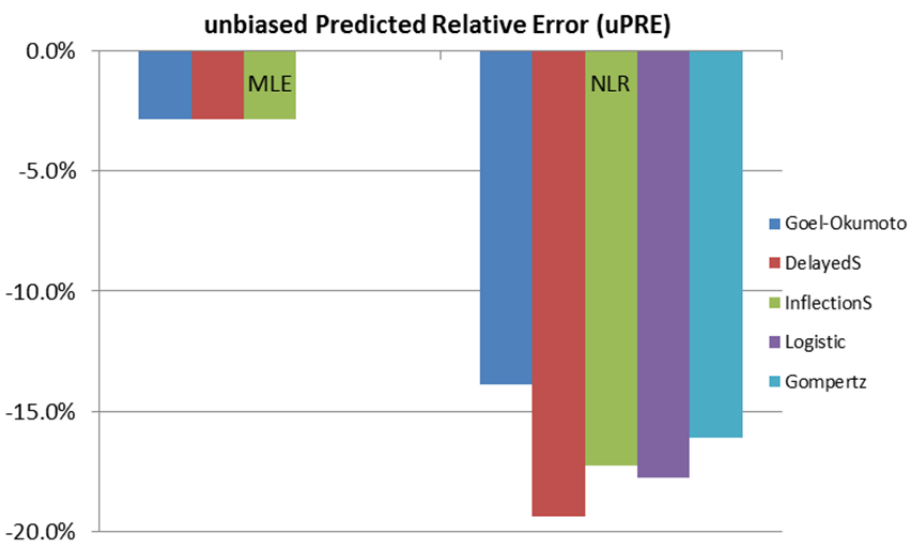

Figure 11: Comparing between BPRE values for MLE and NLR estimations

Figure 11 shows that in our study although MLE estimators also under predict asymptote values the prediction is consistent for all models and prediction accuracy much higher (BPRE lower than -5%). While the unbiased predictive relative error value for NLR estimators is comparatively higher closer to but under negative 20% for different models tested here.

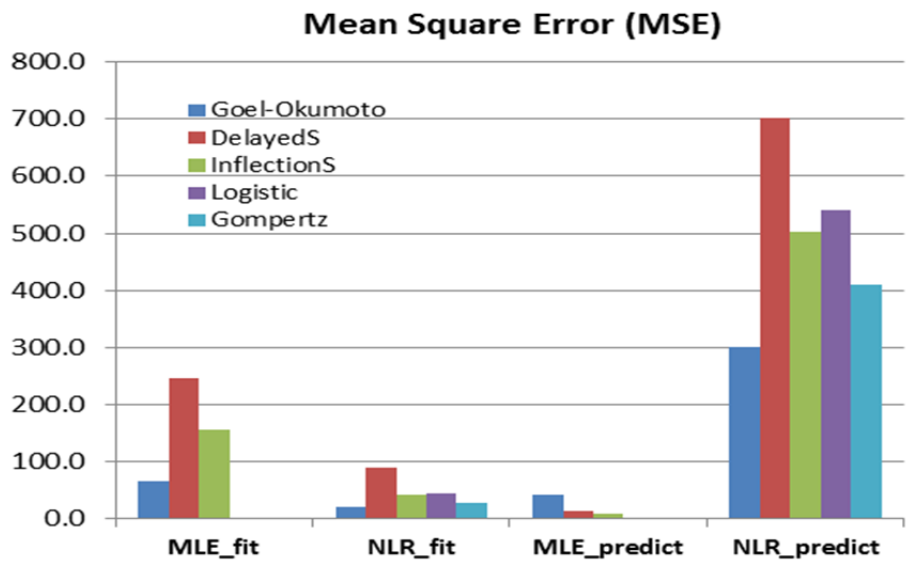

Figure 12: Comparing between MSE fit and predict for MLE and NLR estimation

From Table 10 and Figure 12 we can observe that MSE fit values using NLR are much better compared to values obtained using MLE. This is not surprising given that least square procedure actually minimizes the sum of square of errors between the observed data and used model. On comparing MSE values using MLE obtained in this study to earlier study by Pham and by using equations, we can see that in all but one case MSE values obtained in this study are much smaller than those presented in earlier study and they are also closer to values obtained using MLE simultaneous equations.

Further the interesting point to note from the comparison is that despite NLR giving very good fit values, it does comparatively worse for the MSE values for the predicted values. Mean square error using MLE are significantly smaller to ones obtained using NLR which confirms that MLE is a better estimator for prediction purposes.

Although as described earlier that SSE (Sum of Squared Errors)/MSE is not a fair comparison parameter between MLE and NLR for the fitted data points, but since

MSE for the predicted data is not optimized for both estimators (MLE & NLR), it serves the purpose of comparing between the two estimators on evaluating fit of given model to observed data and goodness-of-fit to predicted data.

3.4.3 Which Estimators give better Fit to data and Predicted values

Another widely used parameter to compare different models and their estimators for their performance is their ability to fit the observed defect/failure data and to the predicted the data. Mean Square Error (MSE) is often used to compare the fit of observed and predicted values. MSE is described in section 3 and values obtained for MLE and NLR estimators are provided in Table 10. The MSE values using MLE and NLR estimation using additional Logistic and Gompertz model (for NLR estimator) is also presented in Figure 12.

Table 10: Comparing MSE fit and predict values for different estimators and models

MSE fit	**MLE**	**NLR**	**Pham**	**Using Equation**
Goel-Okumoto	67.0	20.8	62.7	65.4
Delayed S-shaped	246.6	89.2	420.4	223.8
Inflection S-shaped	155.7	42.3	132.1	
MSE predict	**MLE**	**NLR**	**Pham**	**Using Equation**
Goel-Okumoto	42.7	301.6	50.4	1.6
Delayed S-shaped	12.8	702.0	22.5	40.9
Inflection S-shaped	9.3	501.6	23.0	

3.4.4 Working with un-grouped data

A further practical consideration that needs to be accounted when working with un-grouped data is as follows: in some cases the time between failures is zero for example in the data set used in this study it occurs at fault numbered 33, 61 and 104, highlighted in Table 11. When using MLE estimators with log-likelihood function for NHPP process as given in [66] and using MLE packages such as MaxLik the failures where mean time between failures (MTBF) is zero need to be grouped, else the package can returns NaN errors. And when using the MLE simultaneous equations for GO and Delayed S-shaped model as given in [84], the data used should be un-grouped including the failures with MTBF values equal to zero, else it's equivalent to not considering those failures in the analysis which is also not correct.

3.5 Conclusions

In this study using data from literature we have compared between two of the most widely recommended and used methodology for estimating parameters for the purpose of applying SRGMs to defect/failure data. It is noted in the study that while MLE is the recommended estimator with superior statistical properties, its usability and applicability in all situations is questionable. Further MLE is difficult to apply which limits its use in industry, especially due to lack of tools support.

Although external validity of work presented here may be considered low due to use of only single dataset, the study provides important results that point towards different results obtained using different estimation procedures. The study provides useful and practical insights for industry practitioners and early researchers applying reliability modelling to defect/failure data.

We further provide an improvised metric (BPRE) for comparing the predictive accuracy that is symmetric for over and under prediction addressing the problem identified in this study with widely used metric PRE (predicted relative error).

With results in this study suggesting that the fit, predict and predictive accuracy obtained using MLE and NLR estimators may be much different from one estimator to another, more research in this direction is needed to establish these differences in different contexts and thus helping to resolve the dilemma faced by reliability practitioners of which estimator to use and in which conditions a given estimator is better than other. Initial results presented here and properties of MLE and NLR estimators suggest that while NLR is good estimator for fitting the data to observed failure data, MLE is better estimator for making reliable predictions.

Table 11: Data used in this study, provided in [65] and used in earlier studies including [66], [67]

F	TBF	Cum	F	TBF	Cum	F	TBF	Cum	F	TBF	Cum	F	TBF	Cum	F	TBF	Cum
1	3	3	24	68	2676	47	6	7843	70	379	16185	93	2930	35338	116	122	53443
2	30	33	25	422	3098	48	79	7922	71	44	16229	94	1461	36799	117	990	54433
3	113	146	26	180	3278	49	816	8738	72	129	16358	95	843	37642	118	948	55381
4	81	227	27	10	3288	50	1351	10089	73	810	17168	96	12	37654	119	1082	56463
5	115	342	28	1146	4434	51	148	10237	74	290	17458	97	261	37915	120	22	56485
6	9	351	29	600	5034	52	21	10258	75	300	17758	98	1800	39715	121	75	56560
7	2	353	30	15	5049	53	233	10491	76	529	18287	99	865	40580	122	482	57042
8	91	444	31	36	5085	54	134	10625	77	281	18568	100	1435	42015	123	5509	62551
9	112	556	32	4	5089	55	357	10982	78	160	18728	101	30	42045	124	100	62651
10	15	571	33	0	5089	56	193	11175	79	828	19556	102	143	42188	125	10	62661
11	138	709	34	8	5097	57	236	11411	80	1011	20567	103	108	42296	126	1071	63732
12	50	759	35	227	5324	58	31	11442	81	445	21012	104	0	42296	127	371	64103
13	77	836	36	65	5389	59	369	11811	82	296	21308	105	3110	45406	128	790	64893
14	24	860	37	176	5565	60	748	12559	83	1755	23063	106	1247	46653	129	6150	71043
15	108	968	38	58	5623	61	0	12559	84	1064	24127	107	943	47596	130	3321	74364
16	88	1056	39	457	6080	62	232	12791	85	1783	25910	108	700	48296	131	1045	75409
17	670	1726	40	300	6380	63	330	13121	86	860	26770	109	875	49171	132	648	76057
18	120	1846	41	97	6477	64	365	13486	87	983	27753	110	245	49416	133	5485	81542
19	26	1872	42	263	6740	65	1222	14708	88	707	28460	111	729	50145	134	1160	82702
20	114	1986	43	452	7192	66	543	15251	89	33	28493	112	1897	52042	135	1864	84566
21	325	2311	44	255	7447	67	10	15261	90	868	29361	113	447	52489	136	4116	88682
22	55	2366	45	197	7644	68	16	15277	91	724	30085	114	386	52875			
23	242	2608	46	193	7837	69	529	15806	92	2323	32408	115	446	53321			

EVALUATION OF STANDARD RELIABILITY GROWTH MODELS IN THE CONTEXT OF AUTOMOTIVE SOFTWARE SYSTEMS

Abstract— Reliability and dependability of software in modern cars is of utmost importance. Predicting these properties for software under development is therefore important for modern car OEMs, and using reliability growth models (e.g. Rayleigh, Goel-Okumoto) is one approach. In this paper we evaluate a number of standard reliability growth models on a real software system from automotive industry. The results of the evaluation show that models can be fitted well with defect inflow data but certain parameters need to be adjusted manually in order to predict reliability more precisely in late test phases. In this paper we provide recommendations for how to adjust the models and how the adjustments should be used in the development process of software in the automotive domain by investigating data from an industrial project.

Keywords— Software Reliability Growth Models, Automotive Software, Model Based Development, ISO 26262

3.6 Introduction

Software plays a significant role in modern cars. In past few decades the amount and importance of software in cars has increased exponentially [87], to the extent that today's premium cars carry more than 70 ECUs and software of the order of over ten million lines of code (SLOC) [3]. Software is not only replacing traditional models of control systems but today it is at the heart of providing new functionality and driving innovation. With the rapid growth in significance of software in automotive industry there are a number of challenges the industry faces in developing and maintaining good software for modern cars [3][88].

Automotive software differs from software in other sectors due to stringent demands for rapid development, need for cost effective development, and high demand for innovation and need of high quality and reliability, especially for applications, which are deemed safety critical. To ensure that cars are safe for drivers, occupants and other road users as well as to maintain the consumer confidence, the quality and reliability demand for safety critical software is very high. Functional safety standards such as ISO 26262 [35] provide strict guidelines for the development of software for safety critical applications with significant emphasis on ensuring reliability.

Software reliability growth models (SRGMs) have been used to assess the maturity of software for number of years. Efficient estimation of latent defects in software is valuable information, test managers can use this information to make important decisions not only to ensure optimal resource allocation but also to decide when the given software is ready for release [74]. Applying SRGMs for estimating reliability in industrial applications needs careful consideration to the applied model assumptions, data availability and predictive power, but proper use of SRGMs provides several benefits for developing high quality and reliable software.

3.7 Related Work

Over the years, a number of SRGMs has been presented [68], although similar extent is lacking in the comprehensive evaluation of these models on industrial domain specific applications. This is especially true for the automotive sector. Different industrial domains have very different demands for its software and the development process also varies to a large extent, not all SRGMs would be suited for every sector. Woods [13] applied eight SRGMs on software products from industry and showed that defects predicted based on cumulative defects matches well with after release defects. Staron & Meding [15] evaluated SRGMs on large software projects in the Telecom sector and proposed a new model based on historic

trends data. In this paper we apply common SRGMs on a large project from the automotive sector and evaluate it on simplest fit measure. The applications of SRGMs in automotive software projects are very scarce and with increasing dominance of software in the automotive industry, the need and importance of such studies is very apparent.

In [66], authors present a review of common Non-Homogeneous Poison Process (NHPP) based software reliability models and compare their performance on real time control system. We evaluate SRGMs with only two and maximum three parameters, which are easy to implement and intuitive to understand, this also means that these models can be easily adopted in the industry.

The automotive domain in itself is quite unique, firstly the industry due to various reasons including the historic factors is driven by the "V" development model with high dependence on suppliers, this has also became true to a large extent for the development of software within this domain. Secondly automotive unlike some other industries and like many other similar sectors have widely adopted the model based development approach. Additionally within the Original Equipment Manufacturers (OEMs) there exist numbers of different departments/teams (for example Power-train, Central Electric Module, Infotainment etc.), which develops quite different type of software products and works in quite different working environments. Currently there is also significant trend in automotive domain towards being more agile in their software development process. All these factors affect the defect inflow profiles and the use of SRGMs needs to take these factors into consideration for successful application. In this paper we give a way forward for effective implementation of SRGMs in the automotive sector, what needs to be emphasized and what would lead to optimal software reliability modelling in this domain.

3.8 Research context and method

We use data from a large project within the development of an active safety function from our industrial partner, Volvo Car Group (VCG) from the automotive sector. Department of Active Safety within VCG develops functions/features such as driver alert control, collision warning, lane departure warning etc. The defect data has been used earlier in [89] in a study that introduced a new lightweight defect classification scheme LiDeC. We use dynamic software reliability growth models that have been reported in many earlier studies and are summarized in Table 12.

Table 12: Software Reliability Growth Models used in the study.

Model Name	Type	Mean Value Function	Ref.
Models with 2 parameters			
Goel-Okumoto (GO)	Concave	$m(t) = a(1 - e^{-bt})$	[79]
Delayed S-shaped model	S-shaped	$m(t) = a(1 - (1 + bt)e^{-bt})$	[80]
Rayleigh model		$m(t) = a\, e^{-\left(\frac{b}{t}\right)^2}$	[69]
Models with 3 parameters			
Inflection S-shaped model	S-shaped	$m(t) = \frac{a(1 - e^{-bt})}{1 - \beta e^{-bt}}$	[66]
Yamada exponential imperfect debugging model (YExpI)	S-shaped	$m(t) = \frac{ab}{\alpha + b}(e^{\alpha t} - e^{-bt})$	[90]
Yamada linear imperfect debugging model (Y-LinI)	S-shaped	$m(t) = a(1 - e^{-bt})\left(1 - \frac{\propto}{b}\right) + \propto at$	[90]
Logistic population model	S-shaped	$m(t) = \frac{a}{1 - e^{-b(t-c)}}$	[29]
Gompertz model	S-shaped	$m(t) = a\, e^{-be^{-ct}}$	[91]

To fit the models to our data we used non-linear regression (NLR) routine of the commercially available statistical software package, IBM SPSS. The starting values we used are same for all models and iterations are done until the reduction between successive residuals errors is less than $1.0 * E - 08$. Models with two and three parameters were used in fitting of the curves as these parameters could be interpreted empirically (for instance with respect to the testing effort or maximum number of defects). The models were built based on the data set from all the development phases of the system - starting at requirement analysis and ending with vehicle production testing (i.e. excluding the post-release defects).

3.9 Results and interpretation

The fitting of different SRGMs (two and three parameter models) on actual data is presented in Figure 13 and Figure 14, due to confidentiality reasons the Y-axis scale is not presented and time scale is trimmed at beginning and end representing only partial data for illustrating the fit of the used models. For fitting the model, however, the full data set was used.

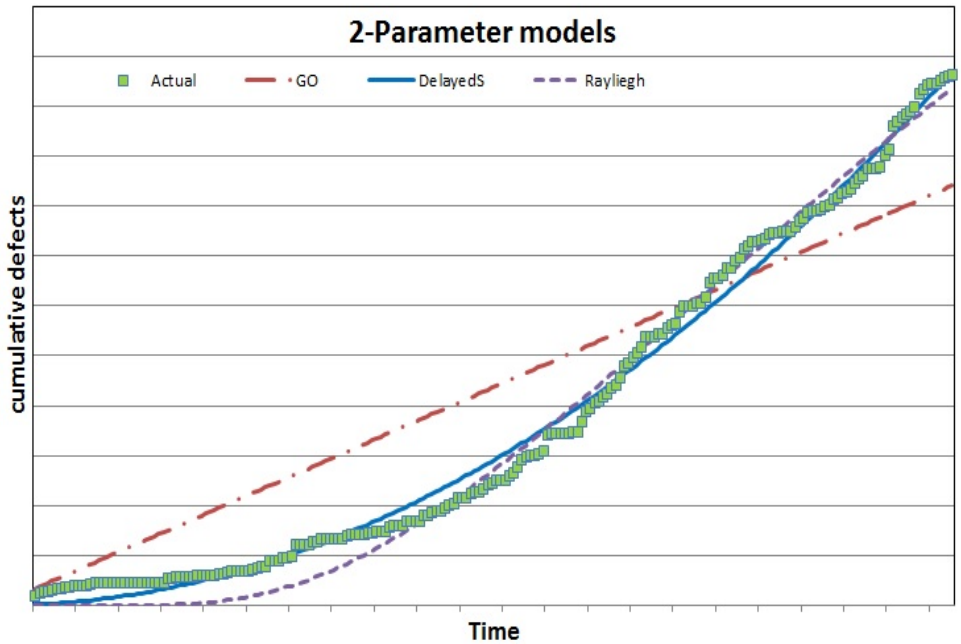

Figure 13: Two parameter software reliability growth models applied to data set from automotive software project[2].

Although (as shown in Figure 13) the models fit the data, they have a tendency of growing exponentially. The exponential growth gives unrealistically high values of asymptotes (maximum predicted defects), such growth is not possible in practice – the number of defects discovered late in the projects decreases over time, thus giving the well-known S-shape of the cumulative defect inflow profile. This shortcoming can be overcome by using three parameter models which include the $\propto (t)$ parameter. The additional parameter is meant to describe the function of test progress over time, and therefore provide more accurate results with logical empirical explanations. Figure 14 presents these models.

[2] Scales on X and Y axis have been removed due to confidentiality reasons. The time domain is also trimmed at the beginning and end to show only the partial data, however full data was used to fit the models.

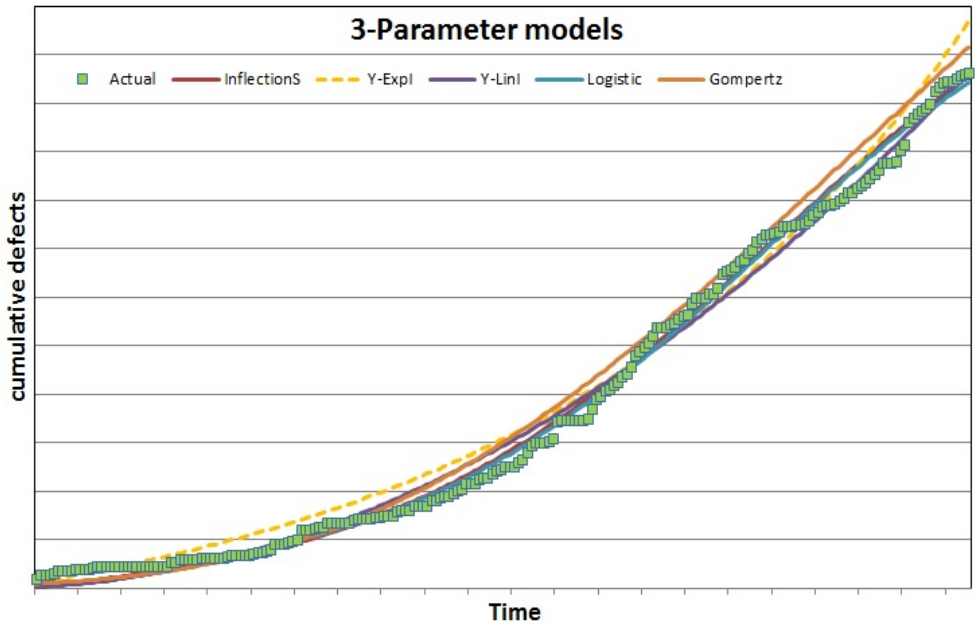

Figure 14: Three parameter software reliability growth models applied to data set from automotive software project.

The analysis of the models and their fit, as shown in Figure 14, suggests that the $\propto (t)$ parameter is promising and will be used in our further analyses. Using the Mean Square Error (MSE) measure to analyse the goodness-of-fit of the models (shown in Figure 15) we observed that the most accurate model was the InflectionS model and the logistic model (used to model population growths in general [92] [93].

MSE presented in Figure 15 for the simplest and one of the earliest Goel-Okumoto (GO) model was approximately 10 times larger than the rest of the models thus we excluded it from the chart to rescale it and focus on the remaining models. As expected, the three parameter models generally fit better than two parameter models, but we observed one exception - the DelayedS model fits better than the Yamada exponential imperfect debugging model (Y-ExpI) and Yamada linear imperfect debugging model (Y-LinI), both of which attempts to account for the testing effort using a third parameter. This means that our initial results should be complemented with more accurate model of the testing effort.

Another significant observation is with respect to the three parameter general logistic model, which performs best among models used in this study with respect to minimum MSE criteria, despite this model not being widely used for software reliability modelling. The three parameters general logistic model is used in many

applications and domains but not as widely in the software reliability modelling although it yields relatively accurate results. Our observation suggests that traditional three parameter models such as logistic and Gompertz model provides superior fit to our data from automotive domain software project. InflectionS model also does very well in MSE fit criteria with MSE only higher than logistic and lower than that using Gompertz model.

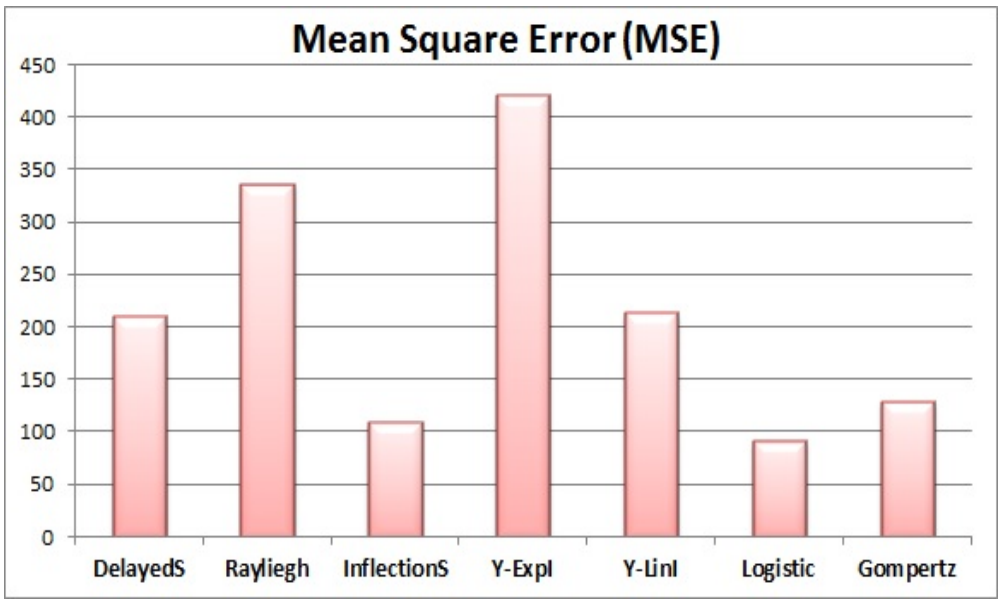

Figure 15: The mean-square error for each of the models. Note that the GO model is excluded in this figure.

3.10 Conclusions

A number of SRGMs have been proposed and evaluated over time. It is noted here that despite software being dominant in modern automotive industry there is a gap in studies evaluating the application of SRGMs in this domain. In this paper we take a step in direction of addressing this gap by applying eight common SRGMs on defect data from a large automotive software project and evaluating their fit using MSE criteria. We further-more, provide a way forward for effective application of SRGMs in automotive software reliability modelling which are as follows:

- It was observed that simple two parameters models can provide good fit (with exception of the GO model), but the asymptotes obtained might be unrealistic;
- Logistic and InflectionS models had the best fit to our data among the different models tried;

- Since one of the important factors for successful use of SRGMs is to use appropriate time scale, we identify that modelling the change of testing effort over time (generally done using parameter $\propto (t)$ will be critical in applying SRGMs within automotive sector;
- Using parameter estimates from two parameter models and based on historic values one could also model/predict the testing effort i.e. $\propto (t)$ for the current project which would give useful insight to project managers for optimizing the resource allocation going forward.

Realistic accounting of testing effort will help us to fit the SRGMs to actual defect inflow data. Finding the models, which provide the best fit, have superior predictive power, and use the data in its available form will significantly enhance the adoption of software reliability modelling in industries where software is starting to play a critical role. And customizing the SRGMs to conform to given industrial domains such as automotive sector will provide a powerful tool to test and quality managers within these industries to use them for optimal resource management, increasing the quality and reliability, and ensuring timely delivery of high quality software.

SELECTING SOFTWARE RELIABILITY GROWTH MODELS AND IMPROVING THEIR PREDICTIVE ACCURACY USING HISTORICAL PROJECTS DATA

Abstract— During software development two important decisions organizations have to make are: how to allocate testing resources optimally and when the software is ready for release. SRGMs (Software Reliability Growth Models) provide empirical basis for evaluating and predicting reliability of software systems. When using SRGMs for the purpose of optimizing testing resource allocation, the model's ability to accurately predict the expected defect inflow profile is useful. For assessing release readiness, the asymptote accuracy is the most important attribute. Although more than hundred models for software reliability have been proposed and evaluated over time, there exists no clear guide on which models should be used for a given software development process or for a given industrial domain.

Using defect inflow profiles from large software projects from Ericsson, Volvo Car Group and Saab, we evaluate commonly used SRGMs for their ability to provide empirical basis for making these decisions. We also demonstrate that using defect intensity growth rate from earlier projects increases the accuracy of the predictions. Our results show that Logistic and Gompertz models are the most accurate models; we further observe that classifying a given project based on its expected shape of defect inflow help to select the most appropriate model.

Keywords— Software Reliability Growth Models; Embedded Software; Defect Inflow; Automotive; Test Resources Allocation; Release Readiness; Automotive; Telecom; Defence Industry

3.11 Introduction

Embedded software is today an integral part of most products, on which we depend for smooth functioning of our daily life. Embedded software does not only provide functionality, it also drives innovation in mobile phones, satellite systems, home appliances, and aircrafts. Reliability is an important attribute of such systems and one way of evaluating their reliability is to use Software Reliability Growth Models (SRGMs). SRGMs are the result of applying reliability engineering theory to the software development domain. The defect inflow data is modelled using mathematical models that quantify the change in reliability of the given software artefact during its development and testing. SRGMs help to answer an important practical question as to when the given software quality is good enough and thus, when can we stop testing [94]. The good-enough quality is also referred to as release readiness of a given product [95]. From the reliability standpoint, one of the most important factors for deciding if a software is ready for release is the number of remaining defects (latent defects). By comparing the predicted total number of defects (asymptote of SRGMs) and the number of defects discovered and resolved to date, software managers can decide if the software is ready to be released [96].

Apart from answering the important release readiness question, SRGMs can also be used to make the software testing process more efficient [97]. However, requirements for the successful application of SRGMs for optimal resource allocation and the assessment of release readiness of software differ. Models which can be applied early in the project and have higher ability to accurately forecast the expected shape of the defect inflow profile are useful for optimizing test resource allocations. While SRGMs that are accurate in forecasting total expected defects in a software product (asymptote) late in the development/testing phase are better suited for assessing the release readiness of a given software system.

Although more than hundred SRGMs have been proposed and evaluated in the literature [68], many of the earlier studies evaluating SRGMs have focused only on how well they could fit to the observed defect inflow data. The evaluation of the predictive power of SRGMs in the literature has generally been limited to only the last few data points (typically last 10% of data) [57] [66]. The difficulty of applying SRGMs in industry is compounded with the lack of studies focusing on specific industrial domains [98] and scarce guidelines to select the best SRGMs for a given software process/application. We focus on the following research questions that are important for reliability practitioners and project managers in software organizations, denoted RQ1-RQ4 below:

Since software development projects have a planned amount of testing resources, we explore how SRGMs can help to allocate these resources more effectively. We

assess which SRGMs are best for this purpose, i.e. we evaluate the SRGMs' ability to correctly predict the shape of the future defect inflow during an on-going project.

RQ1: Which SRGMs are best to assist decisions for optimal allocation of testing resources?

When the software system has been developed and tested, the most important question is: Is the software ready to be deployed (released) or does it need more testing? We evaluate, which SRGMs are best for assessing the release readiness of software systems from the reliability standpoint.

RQ2: Which SRGMs are best for assessing the release readiness of a software system?

Given that software development organizations usually have data on a large number of historical projects, it is also important to evaluate how we can use this experience to make the reliability predictions for current projects more accurate. This is addressed by the following research question:

RQ3: Does using information from earlier projects improve release readiness assessment?

Further, there exists no agreement on which models are the best for a given software development process or industrial domain especially during the early phases of a software project [76], thus we analyse how to select the best SRGM for a given purpose based on available data on an on-going project:

RQ4: How to make the choice of SRGM more effective?

The answers to these questions are the key to successfully applying SRGMs in industrial settings. Evaluation of long-term predictive power of SRGMs in the automotive domain was done in our earlier work [57]. In this paper we extend the analysis by using additional data from two more large organizations engaged in embedded software development but in different application areas (telecom and defence). With the unique setting of large-scale software projects we are able to answer the research questions with higher generalizability. We are also able to make distinctions between the applicability of different SRGMs based on different project attributes, defect inflow profiles and development processes. We further use trend analysis for predicting the shape of the defect inflow for on-going projects - which provides practitioners in industry a framework for selecting and applying SRGMs for supporting decisions of practical significance, such as test resource management and evaluating whether the software product is ready for release.

The rest of the paper is structured as follows: Background for the research and a brief discussion around related works is presented in section 3.12. In section 3.13, we describe in details the design choices of this study, the data, models, and analysis methods we used. Section 3.14 presents the results and analysis of data with answers to the research questions. Section 3.15 presents recommendations for industry to apply SRGMs while conclusions are presented in section 3.16.

3.12 Background and Related Work

Common terms related to software reliability are defined in IEEE 1633: Recommended practice on software reliability [78], accordingly:

Software Reliability (SR): is *(A) the probability that software will not cause the failure of a system for a specified time under specified conditions, or (B) the ability of a program to perform a required function under stated conditions for a stated period of time.*

Software Reliability Model (SRM) is *a mathematical expression that specifies the general form of the software failure process as a function of factors such as fault introduction, fault removal, and the operational environment.*

IEEE standard 1633 also provides metrics used in reliability modelling and specifies the recommended procedure for software reliability assessment and prediction.

SRMs can be classified as white box and black box models [76] [99]. White box models use source code attributes for making the assessment and predicting the defect proneness of a given software artefact, while black box models use defect inflow data for modelling reliability. Based on the nature of the data in use, white box and black box models are also known as static and dynamic models [100]. Dynamic/black box models are usually referred to as SRGMs and use defect data from development and/or testing phases. The failure or reliability process can be modelled using calendar or execution time. Though the execution time models have been shown to be more accurate, the calendar-time models are easier to apply and more intuitive to interpret.

Different models are based on different assumptions, which make some models better suited than others for a given process. Musa et al. [72] showed that various families of models have characteristics that are better suited for certain applications. The same conclusion is also achieved in the study by Goel [71]. Thus, one of the important questions in software reliability engineering has been which models to use and how to apply them [101]. Khoshgoftaar and Woodcock presented a case study [75] to support the claim that Akaike Information Criteria (AIC), based on the log-

likelihood function, can be used to select the best model. Sharma et al. [73] looked at the model selection problem and proposed a quantitative framework based on the distance-based approach that can be used to rank different models and select the optimal one. Stringfellow and Andrews [74] proposed an empirical method to select a suitable SRGM for making release decisions during the test phase. They iteratively applied different SRGMs and if a given model passed the proposed criteria, it could be used for making release readiness decisions.

In this study we introduce a new approach for selecting the appropriate SRGM, which is based on the observed defect inflow profile. We use the trend of defect intensity to predict the shape of the full defect profile, which is used to select the appropriate SRGM for a given purpose. We also evaluate if using this strategy leads to better model selection.

SRGMs evaluations within specific industrial domains are limited, though some studies have been reported. Wood [13] evaluated eight SRGMs on data from industry concluding that defect predictions based on cumulative defect inflow data from development and testing was well correlated with after-release defects. Popov et al. studied the problem of estimating the reliability of multiple-version software to estimate the bounds on reliability of diverse systems [102]. Staron and Meding [15] studied defect data from the telecom domain to found a reliability model based on moving average giving good predictability for weekly defect predictions. Ullah et al. [76] also did a similar analysis using commonly used eight SRGMs on several sets from industrial and open source software projects. Their study found Musa-Okumoto and Inflection models performing best on industrial datasets, while Gompertz and Inflection were concluded as best for the open source software projects. A number of commonly used SRGMs have been evaluated on software projects from the automotive domain in our earlier study [57] demonstrating the usefulness of trend models (Logistic & Gompertz) in such reliability evaluations. In this study we extend our earlier work from the automotive domain [57] and complement previous works [97], [76] in this direction by evaluating, which SRGM performs best for a given software process in the embedded software domain.

To use SRGMs for testing resource management, the long-term predictive power is an important criterion [97]. It is noted that in existing studies the effort has been more focused on introducing new models with higher goodness-of-fit and the assessment of the predictive power was mostly restricted to short-term (typically using last 10% data, for example [103], [66]). An early work in long-term predictability of SRGMs has been presented by Malaiya et al. [97]; they proposed two predictability measures to characterize long-term predictive power namely average error and average bias and used them to evaluate five SRGMs common at the time. The authors used 18 data sets derived from earlier studies and found that

different models have appreciably different predictive powers. Use of secondary sources of data meant that it was either difficult and in many cases impossible to compare the performance based on software domain/development process characteristics. Also metrics based on averages do not allow examining the predictive power of models during a specific point in the project timeline. In this study, our focus is also on the long-term predictive power of SRGMs. We measure a model's predicted defect inflow fit to the actual defect inflow at four distinct phases of a project starting mid-way through the project timeline. This evaluation helps us in selecting the best model for a given purpose and also evaluate when (in project timeline) these models can be used in practice.

Historical project data is proposed to be used to monitor running project progress and evaluating the time to market [104]. Xie et al. proposed using the growth rate from earlier similar projects to avoid the problem of non-convergence when using maximum likelihood estimation for estimating SRGMs parameters early in the testing phases with less failure data available [105]. Our approach to incorporate the information from past projects is similar to the approach introduced by [105], we apply this method for incorporating past projects' information using non-linear regression and use it to evaluate if the predictive accuracy of the models can be increased to make reliable predictions.

3.13 Case Study Design

Using Robson's classification [106], the study presented here is a case study with the main goal of evaluating the applicability of SRGMs in the context of embedded software development projects for decision support with regard to resource allocation and release readiness. Following the taxonomy and guidelines for conducting and reporting case studies in software engineering by Runeson and Höst [52], the presented study is an interpretive case study using a fixed design principle. The research is organized as an embedded case study with each company being the unit of analysis. The similarities and differences based on different application domains and software development processes are explored and highlighted in this study, which suits the embedded case study design. The case study design overview is presented in Figure 16.

Figure 16: Overview of case study design

Table 13 shows the summary of the characteristics of the process used in the companies and the mapping to the industrial domain.

Table 13: Overview of units of case analysis within this embedded case study

Company (unit of analysis)	**Application domain**	**Software development process for studied projects**
Volvo Car Group	Automotive	V-shaped software development mostly using sub-suppliers for implementation
Ericsson	Telecom	Agile development, mostly in-house
Saab EDS	Defence Equipment	Waterfall development (old projects) with development concentrated in-house

3.13.1 Case and subjects selection

In this study, the selection of case units was primarily driven to capture the variation in software development process and application domain. Two case units although allows for a better comparison between case units, but also limit the generalizability of results. While if four or more case units are chosen it leads to difficulties in conducting cross-case analysis including necessary details, thus three case units were selected for this study. Next we provide brief descriptions of each unit of analysis and their application domain (from the authors earlier work [107]), details about each unit's software development process is also included.

Company A: Volvo Car Group

Volvo Car Group (VCG) is a Swedish car Original Equipment Manufacturer (OEM), based in Gothenburg. VCG develops software and hardware typically in a distributed software development environment, but for a limited number of Electronic Control Units (ECUs) the software is also developed in-house. The development is done by the software development teams who usually also hold responsibility for integrating the software with the hardware developed by suppliers. The majority of the embedded software development in the car, however, is developed by external suppliers who design, implement, and test the functionality based on specifications from VCG ([108], [109]).

The size of the entire automotive project in terms of resources is substantially larger than the projects in the other application domains studied in this case study, due to the fact that both OEM and suppliers (first and second tier) are involved and car development projects are usually conducted using the product line approach with reference architectures ([110]).

Software Development Process

The software development process at VCG predominantly follows the V-model. The projects studied here are so called platform projects, which span for a long period of time and are divided into a number of stages (marked as stages S0, S1, to S7 in Figure 17). Each stage is effectively iterative within a larger project where new functionality is designed, developed, tested, verified, and released into the latest system builds. The model is shown in Figure 17.

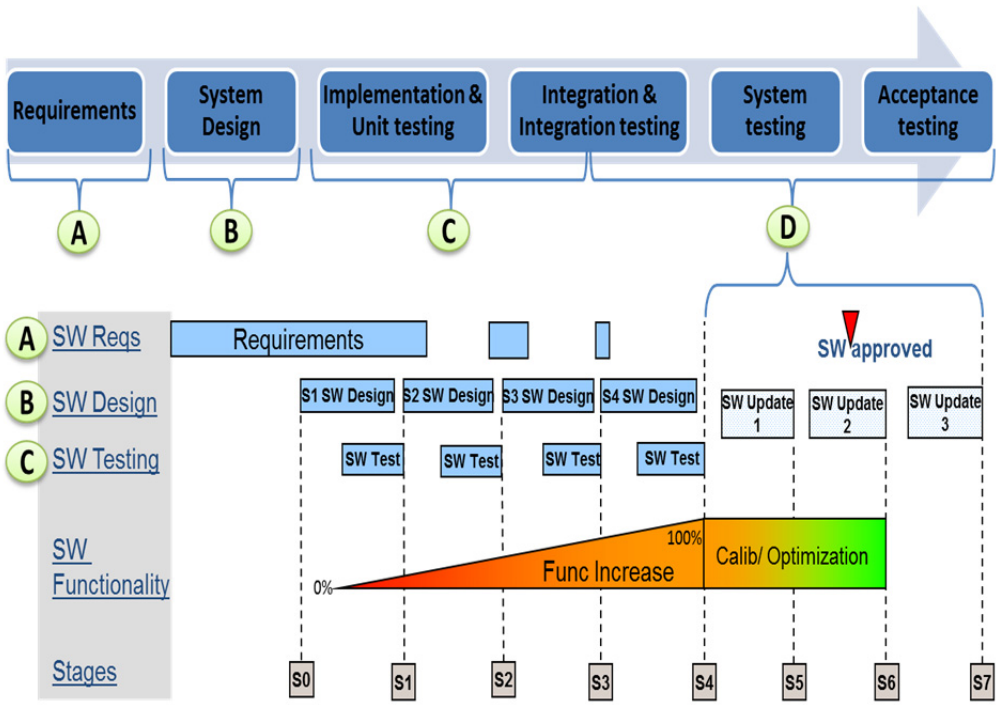

Figure 17: Representation of software development process for case unit 1

A project starts with setting up the requirements, which is followed by design, implementation (in-house or using suppliers) and testing (in each stage). By the end of stage S4, all functionality addition is completed, and the focus is shifted to

calibration and optimization. Defects found during testing of the software are removed as they are detected or patches are provided in the software updates.

Company B: Ericsson

Ericsson develops large software products for mobile telecommunication networks. The size of the projects in this study was up to several hundred engineers. The projects at the company are increasingly often carried out according to the principles of Agile software development and Lean production system, referred to as Streamline development (SD) [111]. In this environment, various teams are responsible for larger parts of the process compared to traditional processes: Design teams (cross-functional teams responsible for complete analysis, design, implementation, and testing of particular features of the product), network verification and integration testing, etc.

Software Development Process

The whole process is dominated with continuous development and testing as expected in a highly iterative agile software development process. The overview of process is presented in Figure 18.

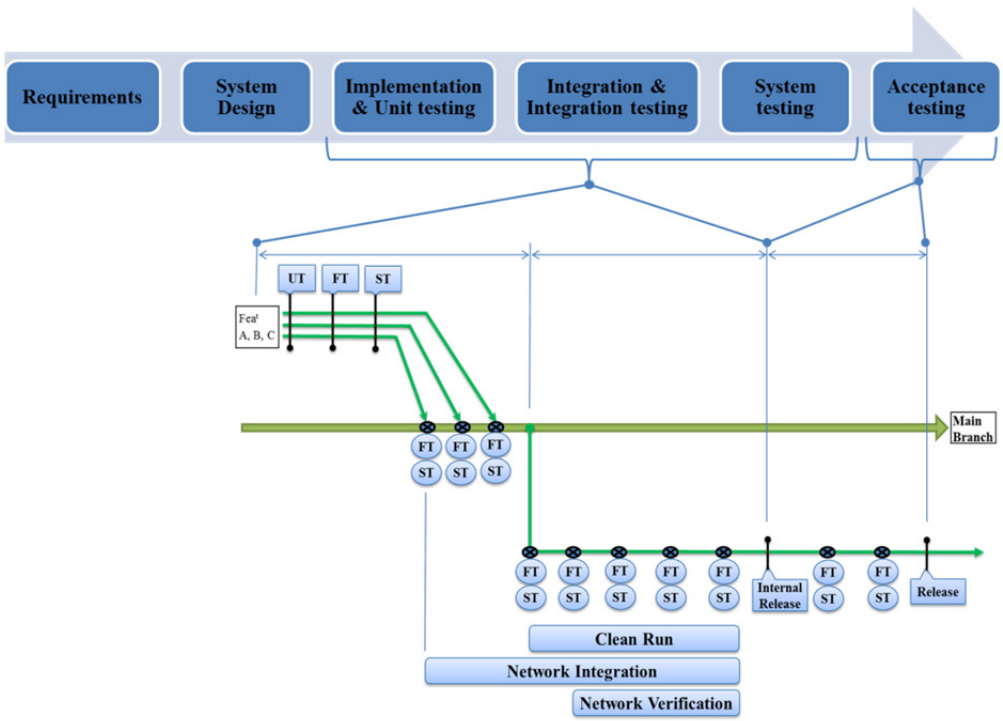

Figure 18: Representation of software development process for case unit 2

Each product has a main branch and for each release, a number of features would be agreed to be developed and released. These features will be developed by separate teams who would also be responsible for unit testing (UT) and preliminary function (FT) and system testing (ST) before releasing it to the main branch. The main branch with the newly developed features is branched out which is subjected to function testing and part of system tests on daily builds with any defects detected being reported to the defect database and resolved in due time. New versions with newly developed features are integrated to network in the Network Integration phase where later phase is concentrated on verification activates (Network Verification and Clean Run using specific test cases for new features). Network integration and verification is completed before the product is released internally. After the internal release, the product is subjected to further function and system testing before making the general release to customers.

Company C: Saab Electronic Defence System (EDS)

Saab EDS develops embedded software and graphical user interfaces for ground based radar systems. The specific project's data used in this study was part of a larger product developed by several hundred developers, designers, testers, analysts etc. The historic project (used in this study) developed the product in waterfall process and did not utilize cross functional teams.

The organization has, since these projects evolved into using more agile processes with increasing use of cross functional teams. A lot of improvements and optimizations have also been applied regarding software build and delivery times. Also to improve customer value, market competitiveness and profit, Saab EDS in Gothenburg is going through a Lean transformation.

Software Development Process

The overview of the software development process at case unit 3 (Saab EDS) is shown in Figure 19.

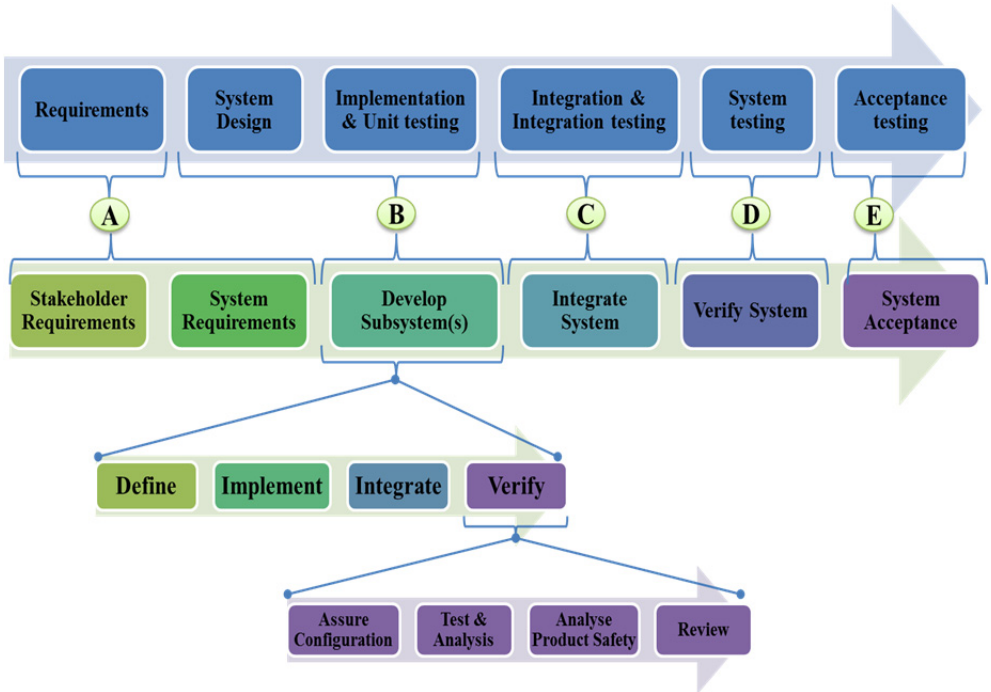

Figure 19: Representation of software development process for case unit 3

The development of software starts with defining the stakeholder's requirements, which are translated to system requirements. The system is broken into a number of sub-systems, which are concurrently developed by different teams; each sub-system is developed, integrated, tested, and verified as individual units. The sub-systems are integrated followed by the verification and validation at system level. The system is then tested for the stakeholder's expectations in acceptance testing and finally made available for release.

3.13.2 Data collection and analysis methods

Based on the existing literature, seven widely used SRGMs were selected to be evaluated in this study as in our earlier study [57]. Selected models are a mix of concave, S-shaped, and trend models. These models are frequently mentioned, evaluated, and applied in software reliability research and practice. Also to account for the highly iterative streamline software development process at Ericsson, we also use a linear model that has been used in prior studies [112] for agile and streamline software development processes. It is noted here that the linear model does not have an asymptote modelled in the equation $(m(t) = g * t + c)$, the model assumes a constant growth rate (g) and the total predicted defects in this case are taken as predicted defects at the time when the project is finished. Since the linear model

does not have a theoretically motivated asymptote prediction, it is excluded for its applicability for conducting release readiness assessment of software in this study. A summary of the software reliability models used in the study with their mean value functions is presented in Table 14.

Table 14: Software reliability growth models used in this study

No	Model Name	Shape	Structure	Mean Value Function	Reference
1	Musa-Okumoto (MO)	Concave	NHPP	$m(t) = a \ln(1 + bt)$	[113]
2	Goel-Okumoto (GO)	Concave	NHPP	$m(t) = a\,(1 - e^{-bt})$	[79]
3	Inflection-S model	S-shaped	NHPP	$m(t) = \frac{a\,(1 - e^{-bt})}{(1 + \beta e^{-bt})}$	[7]
4	Delayed-S model	S-shaped	NHPP	$m(t) = a\,(1 - (1 + bt)e^{-bt})$	[80]
5	Rayleigh model	S-shaped	NHPP	$m(t) = a\,(1 - e^{-\left(\frac{t}{b}\right)^2})$	[69]
6	Logistic model	S-shaped	Trend	$m(t) = \frac{a}{(1 + e^{-b(t-c)})}$	[114]
7	Gompertz model	S-shaped	Trend	$m(t) = a\,e^{-be^{-ct}}$	[91]
8	Linear model	Linear	Trend	$m(t) = g * t + c$	[112]

Software reliability models can be fitted to the observed defect inflow data using statistical techniques such as maximum likelihood parameter estimation or curve fitting techniques like Non-Linear Least Square (NLLS). Following earlier studies [76], [57], [115], we used NLLS for the parameter estimation of models. Non-linear regression routine in commercially available statistical software package (IBM SPSS) is used to make parameter estimations. Since the parameter estimation using NLLS is an iterative process, we run iterations until successive iterations returned less than 1.0*E-08 difference in residual errors, which is the default setting. For the case of Inflection-S model, the β parameter value is assumed to be equal to 1.2 following the parameter estimation procedure given by [7].

To assess which reliability models are best and when they can be applied during a project timeline, we evaluated their respective goodness-of-fit and accuracy of asymptote prediction using full and partial data sets. For each project we divided the data into four sets (p0, p10, p30 and p50), containing all, 90%, 70% and 50% data points respectively – this data is referred to as the observed region of defect inflow profile. These sets correspond to same level of project completion with respect to project timeline (our data is weekly defect count data). The data points in the observed region for each set are used for model fitting (for estimating the parameters of given SRGM using NLLS method), while the remaining data points toward the end of the project (i.e. 0, 10, 30 and 50% referring to p0, p10, p30 and p50 respectively) are used to assess the predictive power of a given SRGMs – this region is referred to as the predicted region.

3.13.3 Data collection

The main source of data for the case study is defect inflow data from defect reporting systems of individual companies. Already collected metrics data from software projects is archival data [52] that offers limited possibility for researchers to control or assess the quality of data. But in mature software development organizations defect reporting is generally a controlled and monitored activity, which is the fact for each case unit used in this study and thus the data used is of high quality. In order to ensure the comparability of data collected for the different case units, a common definition of defect inflow was used as given by [15]: *"defect inflow is the number of non-redundant defects reported in the defect database"*, the definition was cross-checked with each company involved in this multiple case study and appropriate filters were used to collect the data from their defect reporting databases.

In total, we collected the defect inflow data from eleven projects (seven individual projects including one project with five releases) from three companies. Data was collected in close cooperation with the industrial partners; all projects in this study were finished before the beginning of our data collection and thus we had full data at hand. The defect data collected is from development to testing phases excluding the post-release defects.

Stronger conclusions can be drawn by using triangulation i.e. using data from several sources [52]. Therefore we complement the defect inflow data with information obtained through interviews. Semi-structured interviews were conducted with managers and developers in each company in two stages:

a. During the case study design phase, i.e. before the defect data was collected. The main aim of these interviews was to understand the context, to get insights of software development process, and to confirm that the data to be collected matches the definition used in this study.
b. After the analysis of defect data, the initial conclusion and interpretations were again discussed with the same interviewees at each company to transfer the knowledge and also to confirm if the conclusions and interpretations aligned with their experience from the actual projects.
c. To minimize the possible bias due to job roles, two people in each company were selected for interviews. A manager and a developer/tester both involved in the original projects and familiar with the companies' defect reporting system and procedures were selected.

Although it is not possible to share the actual defect data due to confidentiality issues, for each project we provide the partial cumulative defect inflow profile, which helps to visualize the distribution as well as to point out similarities/differences between the projects. The total number of defects reported and the total timespan of projects cannot be disclosed and hence the data is normalized.

3.13.4 Metrics used for the evaluation of research questions in this study

SRGMs in general can be evaluated for their ability to fit the observed defect data (i.e. reproduce the observed behaviour), and to forecast the expected future behaviour based on observed data [116].

Goodness-of-fit criterion help us evaluate the models ability to reproduce the observed behaviour, MSE (Mean Square Error) is a well understood measure for evaluating difference between actual and predicted values [117] [70], a smaller MSE indicate closer fit and thus better performance.

Predictive validity criterion on the other hand help us evaluate the ability of model to predict the future behaviour from past and present observed data [116]. Predicted relative error is one such measure which is useful indicator of error between predicted and actual number of defects discovered by termination time of testing [100]. Other similar measure such as relative error defined by Musa [72] have been used to evaluate predictive validity of models in earlier studies [117] [118].

In this study, the comparison criteria used are MSE for goodness-of-fit and Balanced PRE for evaluating predictive validity, these are explained next.

Mean Square Error (MSE)

MSE is the mean of squared error, where error is defined as the difference between actual and predicted value. MSE value can only be used as an interval scale measure to compare between different estimators to rank one model against another. The lower the MSE, the better is the model/estimator goodness-of-fit. MSE measures the variance for an unbiased estimator and is given by equation (1):

$$MSE = \frac{1}{n}\sum_{1}^{n}\left(Y_i - \hat{Y}_i\right)^2 \quad \text{.....(1)}$$

where, Y_i = actual value, $\hat{Y}_i$ = predicted value, and n is the number of data points in use.

Mean square error is used to evaluate the fit of a given software reliability growth model to the observed data and predicted values to the actual values in the predicted region. Other goodness-of-fit metrics such as the R^2 and Theil's statistics have been used in various studies [76] as well as our previous work [57], but given that these metrics are scaled versions of MSE, we only consider MSE to evaluate and compare the fit ability of different models in this study. Another widely used goodness-of-fit metric is Akaike's information criterion (AIC), which is given by equation (2) [67], [75]:

$$AIC = -2 * log(likelihood\ funciton\ at\ its\ maximum\ value) + 2k \quad \ldots\ldots\ldots(2)$$

Under the usual assumptions of Non-Linear Least Square Regression (random error follows Gaussian distribution, which is also same as assuming that random errors are independent and identically distributed), AIC can be redefined [119] [120] by following equations (3) – (4):

$$AIC = n * ln\left(\frac{SSE}{n}\right) + 2k \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(3)$$

$$AIC = n * ln(MSE) + 2k \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(4)$$

where, n is the number of non-missing observations, SSE is sum of squared errors ($MSE = SSE/n$) and k is the number of fitted parameters in the model.

AIC is a better criterion for evaluating effectiveness of fit (lower values imply better fit) as it penalizes those models with greater number of parameters (k). However, in our case since n is large, while the number of parameters for all tested models is small ($k \leq 3$) - ranking of models using either the MSE or AIC criterion is unlikely to be different. Given that the MSE has a more intuitive definition, which is easy to interpret by industry practitioners - we use MSE as the main goodness-of-fit measure in this study.

Balanced Predicted Relative Error

Goodness-of-fit measures one characteristic of reliability models, the other important property is the predicted asymptote accuracy which can be used to assess the release readiness of a product. To measure the asymptote correctness, the Predicted Relative Error (PRE) defined as the ratio of predicted asymptote error to the predicted number of defects is used; $PRE = (Predicted - Actual)/Predicted$, is specified as one possible approach in the IEEE standard 1633 [78] to measure model predictive validity and used in earlier studies [76]. In [121], it is observed that the PRE does not give consistent results for positive and negative deviations and hence, we use Balanced PRE (BPRE) as described in [121], given by equation (5):

$$BPRE = \frac{Predicted - Actual}{Predicted + 2\eta * (Actual - Predicted)};$$

$$where, \eta = \begin{cases} 0 \; if \; Predicted > Actual \\ 1 \; if \; Predicted < Actual \end{cases} \quad \text{.....(5)}$$

BPRE has upper and lower bound of +/-1 respectively; values closer to +/-1 indicate large predicted relative error. While a BPRE value closer to 0 shows that the total number of defects predicted by the model is very close to the actual total number of defects reported in the given project. Corresponding to PRE, BPRE values of +/-10% are usually considered good for making reliable predictions [76] [78].

3.13.5 Analysis methods for the research questions

Which SRGMs are the best to assist decisions for optimal allocation of testing resources?

For assisting decisions to optimize the usage of testing resources such as allocation of human testers, test case scheduling, testing effort etc., a good model should be able to predict the shape of the defect inflow profile in the future, i.e. the fit of model in the predicted region during the project should be as high as possible. Thus, models that give the best fit (minimum MSE values) for the predicted region using 50% and 70% data are best suited for this purpose.

MSE values for all SRGMs that we tested (including the linear model), are presented using average MSE plots (e.g. Figure 24); these plots compare MSE values for all models in the observed and predicted regions for the different sets of data. Lower MSE values on the observed region indicate the superior ability of a given model to fit the observed data, while lower MSE values on the predicted region show that the model is more accurate in predicting the shape of the future defect inflow profile. Figure 20 provides an overview of the analysis procedure in use.

For optimal resource planning and management, the earlier the predictions can be made with high accuracy the better. Thus, we calculate an index MSE value using weights of 50%, 30%, and 20% respectively on MSE values for the predicted region at half-way, 70%, and 90% of a project's completion.

$$MSE_i = 0.5 * MSE_{p50} + 0.3 * MSE_{p30} + 0.2 * MSE_{p10}$$

The index MSE value is used to order SRGMs according to their ability to assist in resource allocation decisions. In the average-MSE plots (e.g. Figure 24), the SRGMs are placed in the ascending order of MSE_i values and thus, the best models are placed on top of the plots. The selection of weights is subjective and can be

customized according to the specific information need of a given organization. For example a company interested in making resource allocation decisions at the mid-way milestone of the project may choose to use 100% weight for MSE_{p50}.

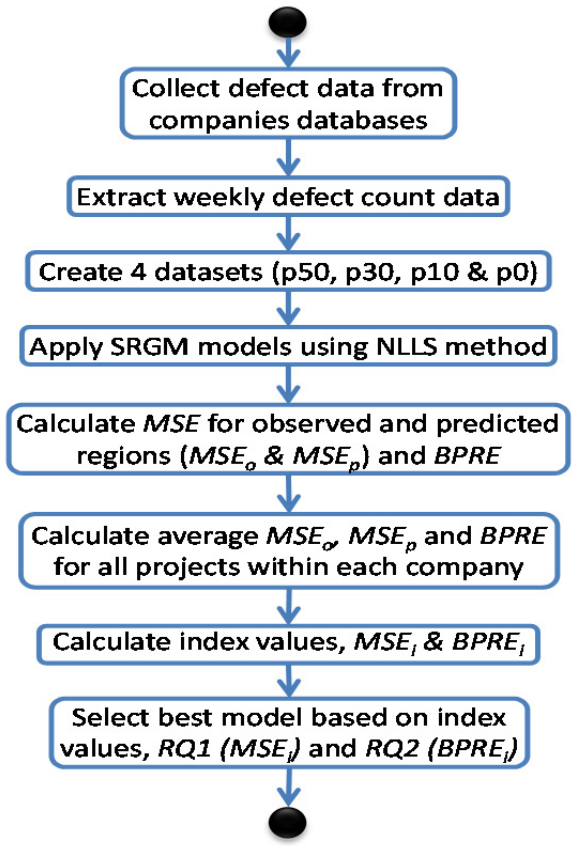

Figure 20: Flow chart for analysis procedure for RQ1and RQ2

Which SRGMs are best for assessing the release readiness of a software system?
To compare between the SRGMs for their appropriateness in determining release readiness of software systems, we calculate the average BPRE ignoring the sign of predictive relative error, giving us Avg+[3]. The Avg+ BPRE values for each tested model are plotted for each case unit (e.g. Figure 25).

[3] Averaging BPRE values can lead to a mean value lower than the actual magnitude of mean due to sign convention for over and under predictions. Since we are only interested in the mean precision accuracy, Avg+ only considers the magnitude.

The criteria for a good model in this case are based on the ability of model to make accurate asymptote predictions towards the end of the project using full or 90% observed data so that latent defects can be estimated with high accuracy. The secondary criterion for selecting the appropriate model for release readiness assessment is its consistency, i.e. the model should make accurate predictions of an asymptote not only late in a project but also when the project is close to completion. We calculate the index $BPRE_i$ value as follows:

$$BPRE_i = 0.5 * BPRE_{p10} + 0.3 * BPRE_{p0} + 0.2 * BPRE_{p30},$$

where the subscripts show which data set is used for BPRE calculation. The weights for calculating index BPRE values reflect the importance a manager assigns, for when such assessment is useful. The selection of weights could be customized by an individual organization according to their information needs. The index BPRE value is used to order SRGMs for their ability to assist with release readiness assessment in Avg+ BPRE plots (e.g. Figure 25). An overview of the analysis method applied is shown in Figure 20.

Does using the information from earlier projects improve release readiness assessment?

Another important and practical question when applying SRGMs in industry is, if and how can we use experiences from earlier projects to make better predictions on an on-going project. The analysis method used for answering this research questions (RQ3) is represented as flowchart in Figure 21.

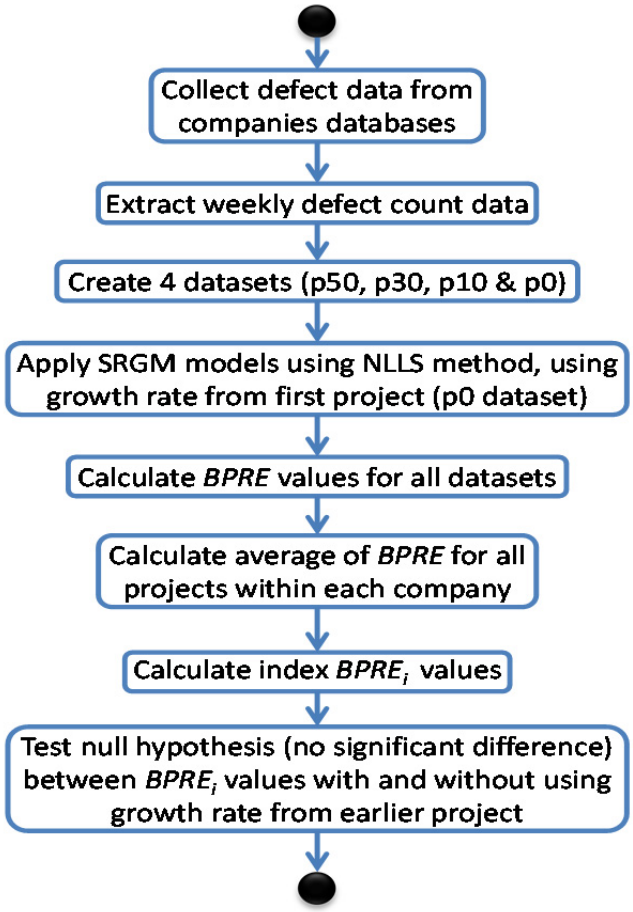

Figure 21: Flow chart for analysis procedure for RQ3

Two important parameters in the SRGM mathematical models are the asymptote value and the growth rate. Since the objective here is to make a better prediction of the asymptote value, we use the growth rate from the first project in each case unit using full data (e.g. p_0 dataset of project A1 in case of VCG) to all other datasets and projects for the given case unit. This is equivalent to assuming that we expect projects in the same organization that are developed and tested by similar teams to have a shape of the defect inflow profile, which is similar to earlier projects.

To evaluate which model performs best when information from earlier projects is used, Avg+ BPRE values for each model tested are plotted for each case unit and ordered using the index BPRE value. Additionally, we also test the following null hypothesis:

H_a: There is no significant difference between the overall performance of asymptote prediction (index BPRE values) with and without using growth rates from earlier projects.

How to make the choice of SRGM more effective?

There are two main shape classes for SRGM models: concave and S-shaped [13] which relate to the general outlook of their shape. Concave profiles bend downwards, while S-shaped curves are first convex and then concave-shaped [13] as illustrated in Figure 22.

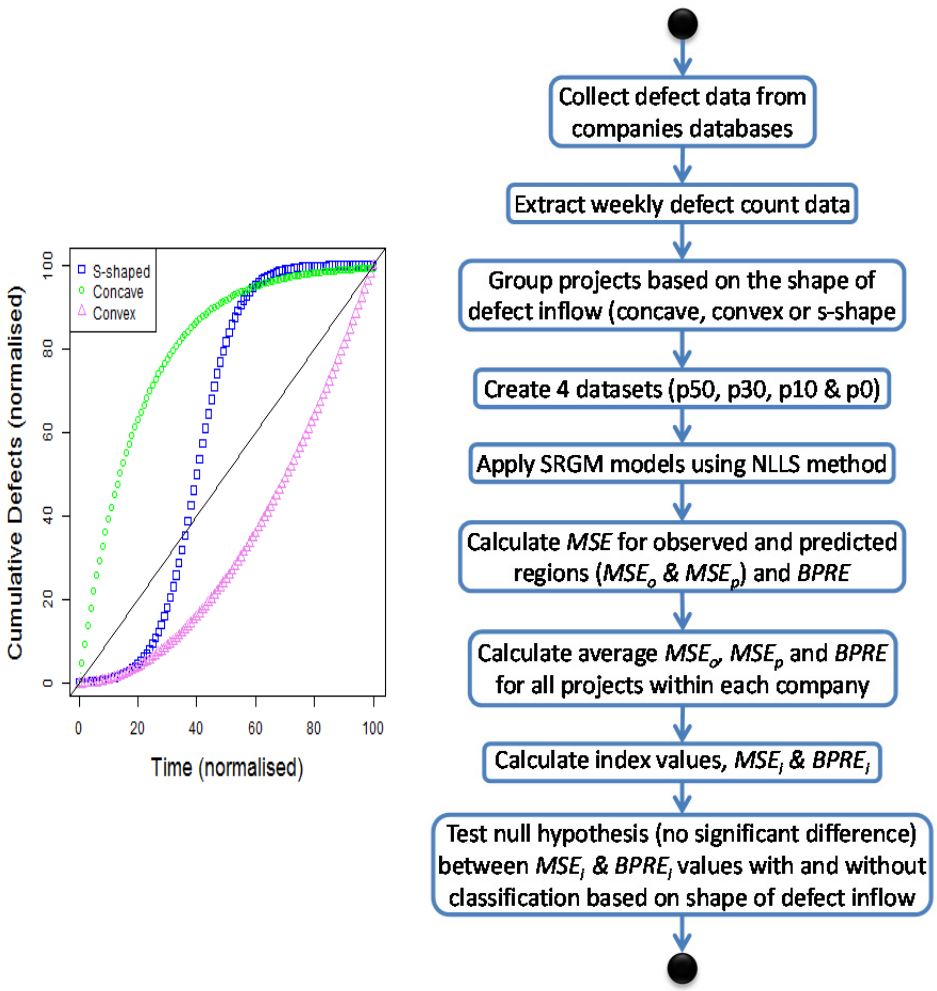

Figure 22: Different shapes of cumulative defect inflow profiles (left) and flow chart for analysis method used for RQ4 (right).

To evaluate if the selection of a model for optimal test resources management and release readiness assessment can be further improved, the projects in this study were classified based on the shape of their defect inflow profiles into the following three categories:

Concave: The cumulative defect inflow profile is concave-shaped if its shape bends downwards, i.e. if you draw a straight line between first and last point and the defect inflow profile is curved outwards for the most part (approximately 70% of time). In these cases, the growth rate for cumulative defects (defect intensity or the tangent to the cumulative defect inflow profile) is at its maximum early in the project, and then drops as project progresses giving a concave-shaped cumulative defect inflow profile.

Convex: The cumulative defect inflows that bent upwards are classified as convex-shaped, i.e. if you draw a straight line between first and last point, the defect inflow profile is curved inwards for the most part (approximately 70% of time). These projects are characterized by a slow growth rate of cumulative defects at the beginning, which stays until late in the project and eventually increasing (refer Figure 22) to give a convex shape.

S-shaped: The S-shaped cumulative defect inflow profile is one that is convex-shaped at the beginning and concave-shaped later. The growth rate of cumulative defect inflow is small at the beginning; it increases to reach a maximum about in the middle of project and eventually slows down to give the characteristic S-shape (Figure 22).

The Average MSE and Avg+ BPRE values are recalculated for the projects with similar shapes of the defect inflow and the best models for a given shape are selected using indexed values as described before. To assess if classifying models into groups based on the predicted shape of the cumulative defect profile improves predictions, following hypotheses are formulated:

H_b: There is no significant difference between the overall performances of a model's ability to fit the predicted region of a defect inflow (indexed MSE predict values) with and without classification into groups based on its shape.

H_c: There is no significant difference between the overall performances of asymptote prediction (indexed BPRE values) with and without classification into groups based on its shape.

To test the formulated hypotheses, we use related samples Wilcoxon signed rank test for H_a and independent samples Mann-Whitney U test for H_b and H_c; the alpha value for all tests is selected at 0.05 level.

3.14 Results and analysis

Since each company has a different software development paradigm, we first analyse each case unit separately following each research question. We also do cross case analysis to highlight how the similarities/differences of the software development process or the shape of the defect inflow affect the selection/applicability of SRGMs within the context of embedded software development.

3.14.1 Case-1: Software Development Processes using V-model: Automotive domain (Volvo Car Group)

In this unit of analysis we study four large[4] software development projects from Volvo Car Group (VCG), a company from the automotive domain. The projects come from the E/E (Electrical and Electronics) integration department within the VCG, which deals with the integration of various software functionalities and which is responsible for the final assessment of the complete EE hardware and software systems. All projects used in this study have been completed during the last decade. Furthermore, they consist of different modules developed by different teams and tested within the development team (unit testing), while further integration and acceptance testing is done by dedicated teams in the integration department. All defects that are detected during all testing phases are reported in the central defect database used by VCG, which is also the primary data source for this study.

3.14.1.1 Defect Inflow Profiles

[4] We define a software project as large software project if it involves at least team of 40 developers and testers working for a minimum period of six months. All projects studied in this study quality this criterion by a large margin.

The cumulative defect inflow profile for the four projects analysed from case unit 1 are presented in Figure 23. From the figure we can observe that most projects (all except project A2) have an S-shape. The cumulative defect inflow profile of project A2 resembles the convex defect inflow. The specific difference of this project to the other projects was no surprise for the interviewees of this company, who explained that the defect reporting strategy was different for this project – project A2 included defect reports from a specific team which applies an agile process and generally reports their defects in another database. Despite this, it is included in this analysis since the specified teams' contribution to the overall project A2 was comparatively large.

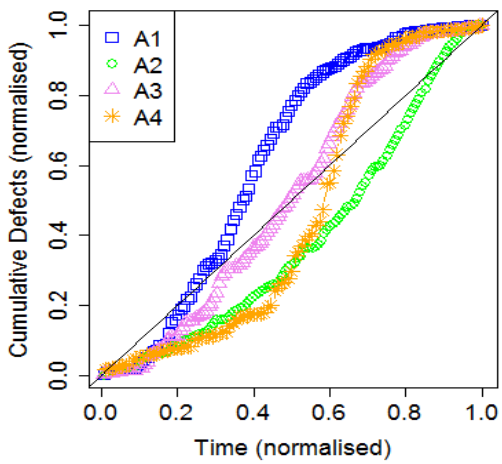

Figure 23: Cumulative defect inflow profiles for case unit-1.

3.14.1.2 Which SRGMs are the best to assist decisions for optimal allocation of testing resources?

The goodness-of-fit measure, the mean square error (MSE) is presented in Figure 24. It is important to note that the values are averages for all four projects from this case unit. In the figure, the horizontal bars on the left side represent MSE values for the observed region, while the bars on the right side within the same figure show the MSE values for the predicted region using 50%, 70%, and 90% data. We do not show the magnitude of MSE values in the figure since the actual magnitude of MSE does not have a physical interpretation because it is measured on an interval scale and is primarily useful for comparative purposes. The fit measure differs in magnitudes on the observed and predicted region – the MSE values are much higher in the predicted region than in the observed region. Thus these values are scaled down by a factor of five (MSE predict values divided by five on right side) in all MSE plots.

The difference in the order of magnitude shows that the fit is superior in the observed region, which is not surprising given that NLLS minimizes the sum of square of errors. We also observe that models where less data is used (e.g. 50% data) fit the observed data better than when using more data (e.g. 90% data), simply due to the fact that it is relatively easy to configure parameters to fit less data points, while fitting the same model to more data points leads to more errors and thus a comparatively inferior fit. At the same time, the opposite is true for the predicted region – using more data gives more accurate parameter estimates making the predicted part fit better as more data points are used (represented by the bars on the right side). The SRGMs are ordered using indexed value of MSE calculated as described in section III part D.

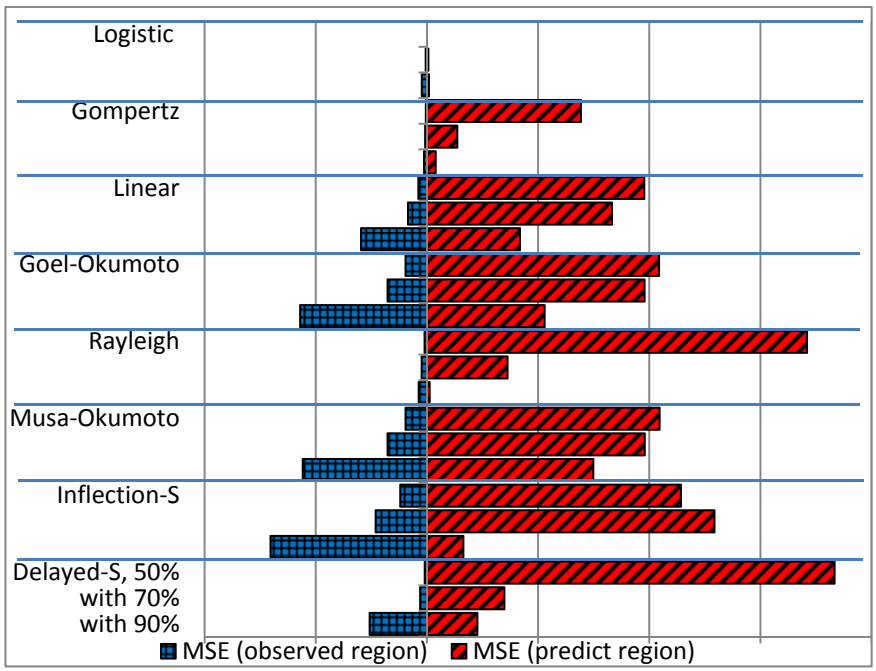

Figure 24: Average MSE for studied SRGMs over full & partial data for case unit 1.

With respect to the goodness-of-fit criterion, we observe that the Logistic model fits best in the observed region. The concave models (Musa-Okumoto and Goel-Okumoto models) do not give good fit overall, and Delayed-S is the worst regarding the fit criterion.

With regard to fit in the predicted region, the Logistic model again performed the best, although it is noteworthy that the Logistic model is unable to converge to the set of 50% data, which shows that this model is not useful with little data. An

alternative to Logistic is the Gompertz model, which converged in all cases and provides good fit with high reliability; however not as good as the Logistic model.

Therefore the Logistic model is best for making test resource allocation decisions as it has superior ability to accurately predict the shape of the defect inflow compared to other models but in this case unit as the Logistic model is unable to give results with 50% data at that point of project timeline, the Gompertz model is recommended.

3.14.1.3 Which SRGMs are best for assessing the release readiness of a software system?

Figure 25 displays the BPRE values for all SRGMs that are used in this study averaged across the four projects. It is obvious that the concave models perform the worst. While S-shaped models perform better than the concave models, trend models are the most accurate in this respect.

Again, the Logistic model gives the most accurate results in predicting the total number of defects in a project when it is close to completion (using all and 90% data). If the projects are half-way, the model is unable to converge for allowing reliable predictions. Gompertz on the other hand has good predictive accuracy and is also consistent in its predictions over different data sets for this software development process.

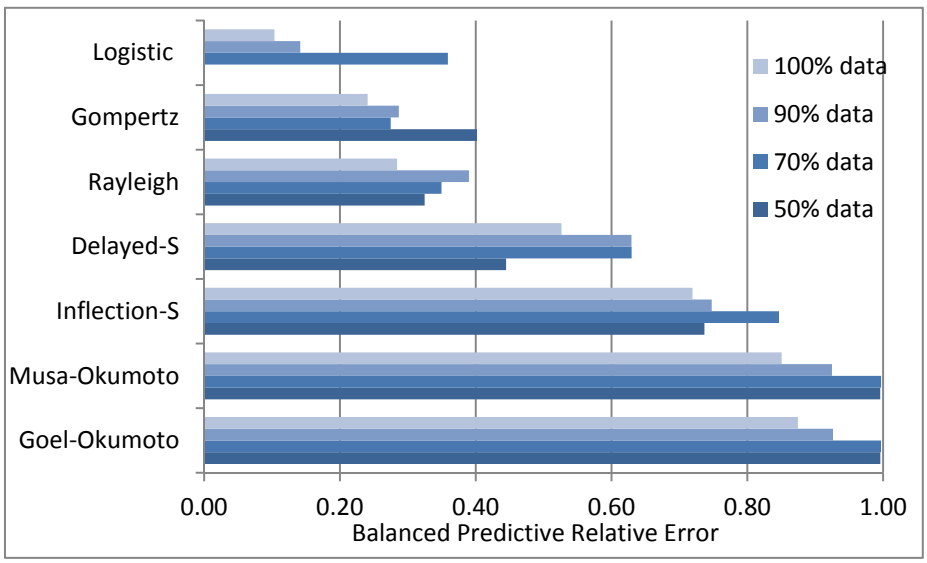

Figure 25: Average+ BPRE for studied SRGMs for case unit 1.

Based on the analysis of the case unit 1, for a software development process that resembles the V-model, the Logistic model is the best model for the assessment of the release readiness.

3.14.1.4 Does using the information from earlier projects improve release readiness assessment?

The estimates for BPRE values using the growth rate from project A1 are presented in Figure 26. The results show a sharp improvement in the prediction accuracy compared to BPRE values that are obtained without using the growth rate from an earlier project (Figure 25).

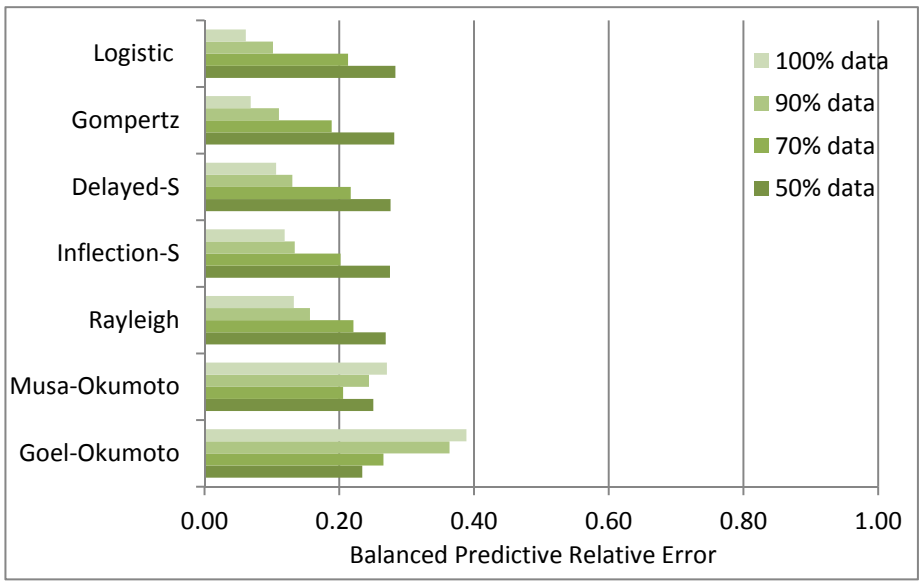

Figure 26: Average+ BPRE for studied SRGMs with growth rate from Project-A1 (100% data), case unit 1.

Except for the concave models, using a growth rate from an earlier project increases the accuracy of the asymptote predictions resulting in BPRE values close to +/-10% with all and with 90% of the data. The results demonstrate that using information from earlier projects is useful for increasing the accuracy of asymptote predictions in on-going projects for embedded software development in the automotive domain.

Testing hypothesis H_a using Wilcoxon signed rank test for this case unit, we are able to reject the null hypothesis (p-value = 0.018) that shows a statistically significant improvement is achieved in the prediction accuracy of SRGMs by using the growth rate compared to applying models using no information from earlier projects.

3.14.2 Case-B: Highly Iterative Software Development Processes: Telecom domain (Ericsson)

3.14.2.1 Defect Inflow Profiles

The defect data from five consecutive releases of a single product is used for the analysis of this case unit, the defects data collected and included in the analysis is from function and system testing starting when new features are delivered to the main branch for the first time until the new release is made available (released) internally, as shown in this case unit's software development process in Figure 18.

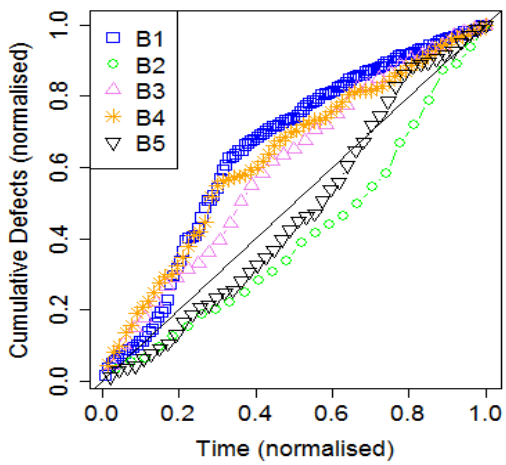

Figure 27: Cumulative defect inflow profiles for case unit-2.

The partial cumulative defect inflow profile for five releases is presented in Figure 27. The defect inflow profiles in this case unit with its highly iterative software development and testing process do not show S-shaped characteristics, which we observed in the previous case unit. Three out of five profiles are concave-shaped while the remaining two are convex-shaped. To accommodate prior studies for the iterative software development paradigm [112], a linear model is also evaluated and compared to seven other models in this study.

3.14.2.2 Which SRGMs are the best to assist decisions for optimal allocation of testing resources?

The goodness-of-fit measured by MSE for this case unit is shown in Figure 28. The first interesting observation is that the ability of the linear model, which is recommended in earlier studies [112] for agile software development process to predict the shape of the defect inflow profile, is lowest among the tested models.

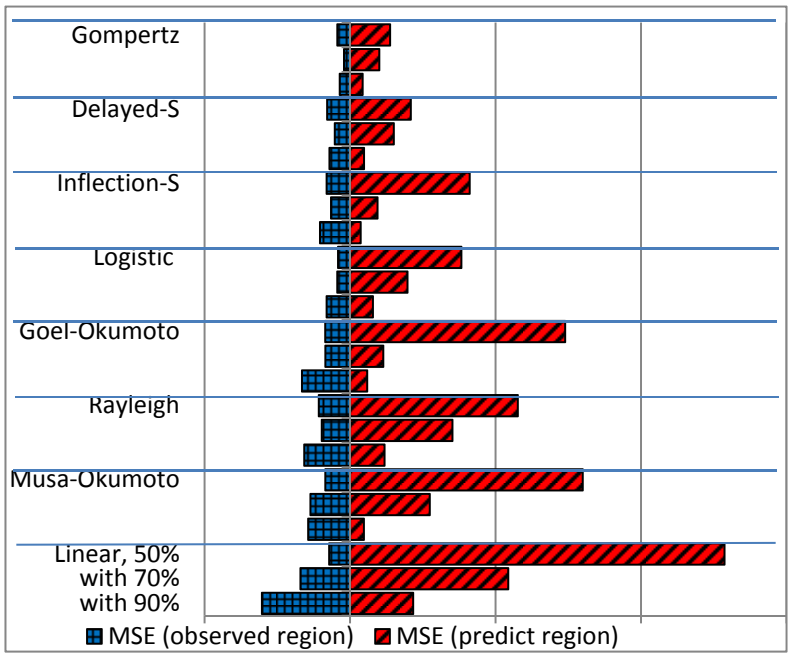

Figure 28: Average MSE for studied SRGMs over full & partial data for case unit 2.

The Gompertz model performs best for this case unit following an iterative development process.

3.14.2.3 Which SRGMs are the best for assessing the release readiness of a software system?

The asymptote prediction accuracy is presented in Figure 29, the concave models and the Inflection-S model performed poorly for this case unit. With respect to accuracy of predictions and consistency to make good predictions with different data sets, the Logistic model is found to be the best model.

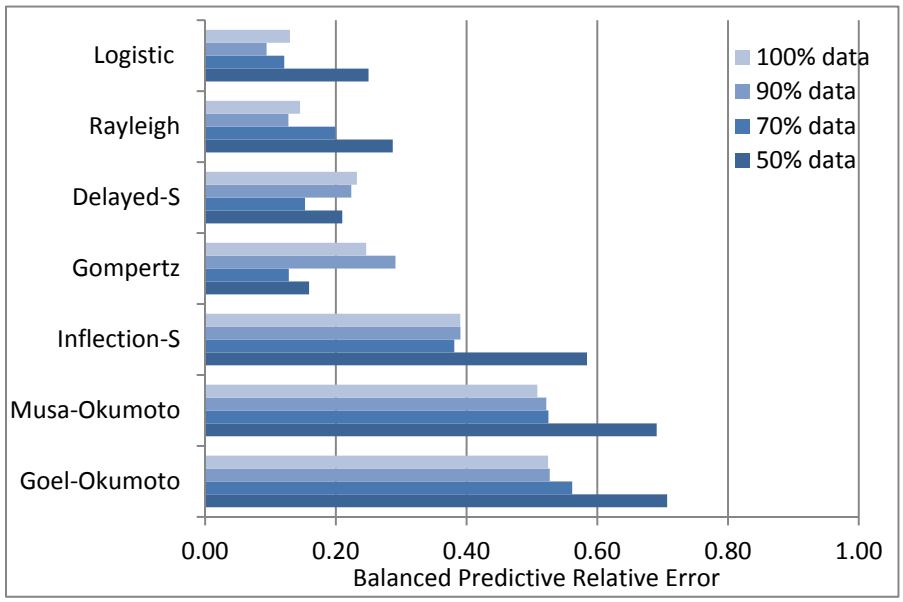

Figure 29: Average+ BPRE for studied SRGMs for case unit 2.

The results here suggest that for making decisions on release readiness where the decisions are made towards the end of the project timeline and where the main criterion of assessment is the likelihood of presence of latent defects ($latent\ defects = Actual\ defects - Detected\ defects$), the Logistic model is the recommended model for a software development process that is highly iterative.

3.14.2.4 Does using the information from earlier projects improve release readiness assessment?

Figure 30 shows the asymptote prediction accuracy (BPRE values) when the growth rate from an earlier project (here release B1) is used for later releases. Comparing the results (Figure 30) to those without using the previous growth rate (Figure 29), we can observe that the prediction accuracy of concave models improves significantly. Using the earlier growth rate, the prediction accuracy of concave models (Musa-Okumoto) is best among the tested SRGMs, the predictions are also consistent among full and partial data sets.

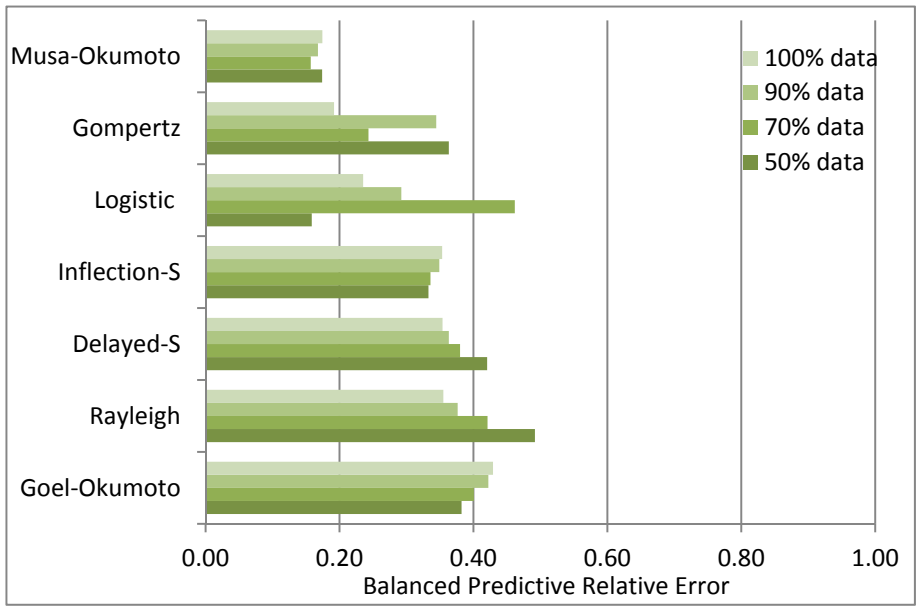

Figure 30: Average+ BPRE for studied SRGMs with growth rate from Release-B1 (100% data), case unit 2.

Another observation is that the asymptote prediction accuracy for S-shaped and trend models on average decreased by a small amount in this case after using the growth rate from release B1. This decrease of predictive accuracy can be attributed to factor that two out of five releases, defect inflow is different from release B1. This point toward an important lesson that the growth rate to be used should be from projects, which is a typical case or some form of weighted average should be used instead, which can be customized to reflect more closely the similarity between current and past projects.

Overall, if the objective is to assess if a software is ready for release and a growth rate from earlier projects is available then the concave model Musa-Okumoto performs best when the software development is streamline (lean and agile) as in this case unit (Ericsson). Testing hypothesis H_a for this case unit, we were unable to reject the null hypothesis (p-value = 0.735 showing that overall there is no statistically significant improvement in the prediction accuracy by using the growth rate compared to applying models using no information from earlier projects for release readiness assessment.

3.14.3 Case-C: Modified Waterfall Software Development Processes: Defence Equipment's (Saab Electronic and Defence Systems)

3.14.3.1 Defect Inflow Profiles

The defect data analysed in this case unit comes from two large software products. For each product, defects reported during the development and testing of all sub-systems and during integration testing and verification are collected (refer Figure 19). The cumulative defect inflow profile for the two projects is represented in Figure 31.

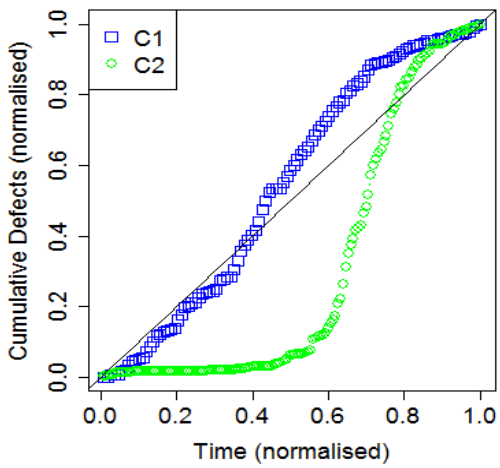

Figure 31: Cumulative defect inflow profiles for case unit-3.

The cumulative defect inflow profiles show signs of an S-shape in project C1, while the second project is closer to convex-shaped; it is also observed that the second project in this case unit had an unusually low growth rate in the beginning which was confirmed by the interviewees for this unit. It is also noted that since the number of projects available for analysis in this case unit is only two, deviations in any of the projects can lead to large deviations on the average value and thus, these aspects need to be considered when analysing the results.

3.14.3.2 Which SRGMs are the best to assist decisions for optimal allocation of testing resources?

The MSE values measuring goodness-of-fit for this case unit are shown in Figure 32.

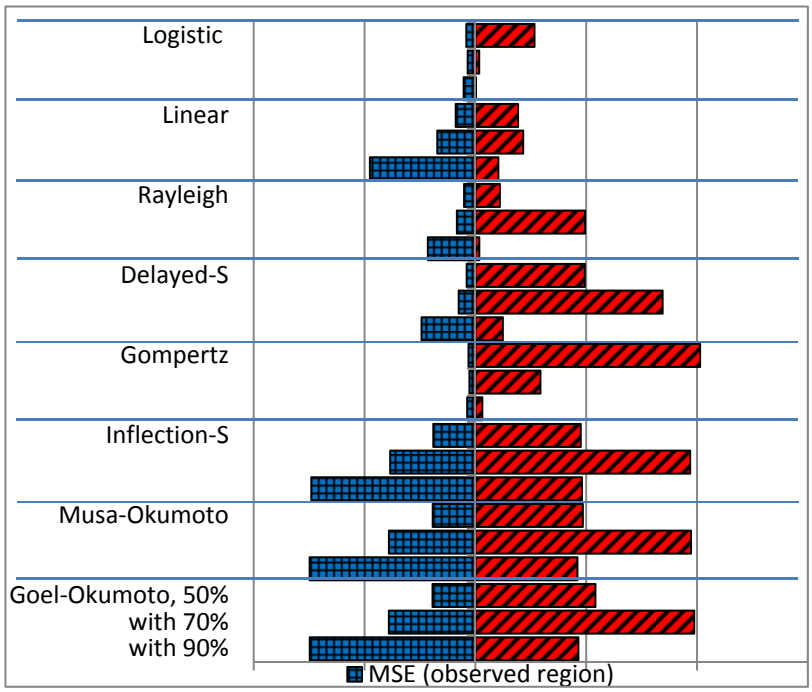

Figure 32: Average MSE for studied SRGMs over full & partial data for case unit 3.

It is observed that the concave models (MO, GO) and Inflection-S model do not fit well, neither on the observed region nor on the predicted region. Delayed-S and Rayleigh models perform comparatively better, while the Logistic models fit best in the observed data and predicted region. The results are similar to the ones obtained in case unit 1. We see the Logistic trend model to be the best among all tested models. Thus, the preferred model for optimal allocation of test resources would be the Logistic model for a software development process that resembles the waterfall model.

3.14.3.3 Which SRGMs are best for assessing the release readiness of a software system?

Again given that we only have two projects in this case out of which one has an unusually low growth rate at the beginning, we see large deviations in performance of models when applied at different times of the project timeline (cf. Figure 33). Nonetheless, on average we confirm the general observations from case unit 1, concave models asymptote predictions are very close to upper limit showing large error in predicted asymptote values in these projects. The prediction accuracy of S-

shaped models is better than concave models but still not close to the desired accuracy levels for use in practice. Trends model such as Logistic and Gompertz perform better when more data is used but unlike the other case units, these models are inconsistent given the large deviation of project C2's defect inflow profile.

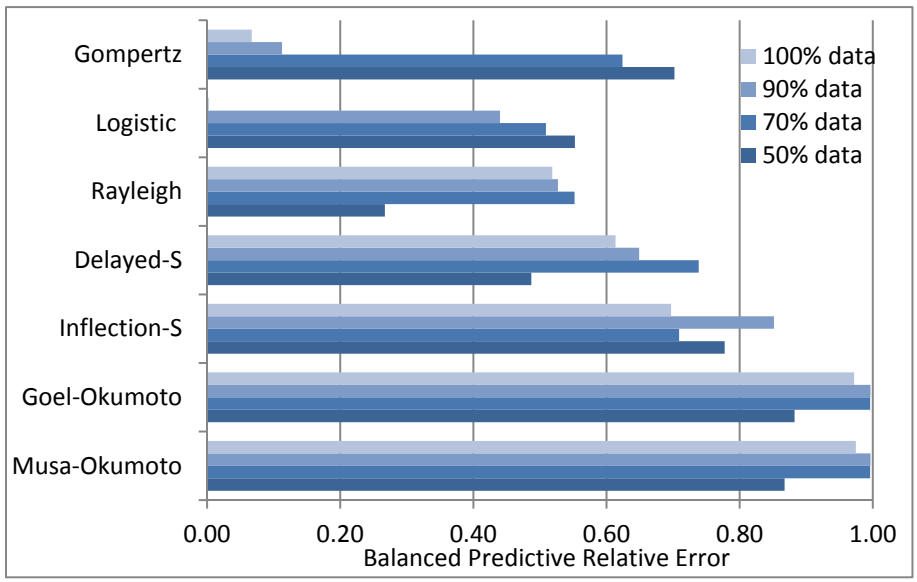

Figure 33: Average+ BPRE for studied SRGMs for case unit 3.

While we observed in case unit 1 as well in case unit 2 that the trend models (Logistic and Gompertz) performed well in asymptote prediction accuracy, results here suggest that if the defect inflow profile deviates heavily from S-shape (such as in project C2 here), the use of these models with partial data sets either should be avoided or thorough care needs to be taken when interpreting the results.

Nonetheless, for this case unit, the Gompertz model is shown to be the best for assessing release readiness of a given software project.

3.14.3.4 Does using information from earlier projects improve release readiness assessment?

shows the asymptote prediction accuracy when we use growth rate from project C1 to improve prediction accuracy for the case unit 3. In this case unlike the case unit 1, we do not see significant improvements in the asymptote predication accuracy for concave models but the prediction accuracy of S-shaped models and trend models is improved. The observation also shows that using information from an earlier project increases the consistency of prediction accuracy.

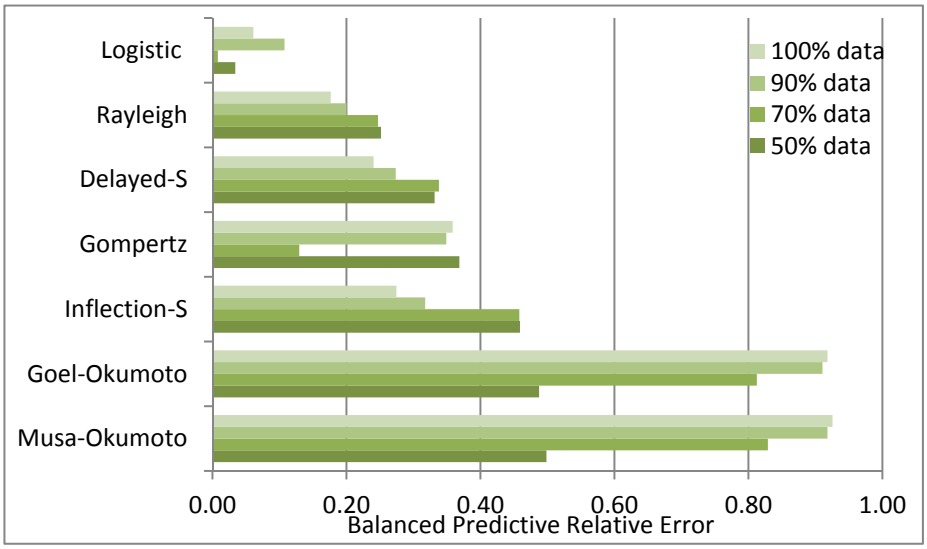

Figure 34: Average+ BPRE for studied SRGMs with growth rate from Project-C1 (100% data), case unit 3.

The result is interesting given the fact that C2's defect inflow profile deviates from the expected standard shape and thus, it is concluded that if one has to use SRGMs for asymptote prediction and data availability is restricted (i.e. when predictions are needed early in development project) one should try to maximize the use of the information from earlier projects. This strategy will improve the chance of making predictions, which are more accurate than making asymptote predictions without using such information. The Logistic model is best suited for release readiness in this case unit when a growth rate from an earlier project is used.

Testing hypothesis H_a for this case unit, we were again unable to reject the null hypothesis (p-value = 0.063) that shows that overall there is no statistically significant difference in the median of prediction accuracy of SRGMs with and without using the growth rates at 5% alpha level. The best model selected using growth rates (Logistic) had a much lower indexed BPRE value than the best model (Gompertz) without using growth rates, 7% compared to 20%, which indicates that the best model using growth rates is much more accurate for making release readiness assessment than using the best model without using a growth rate.

3.14.4 Cross Case Analysis

Analysing the three case units together, we first plot the cumulative defect inflow profiles, also called cumulative defect count for all projects analysed under all case units in Figure 35. For the sake of comparison, the X and Y axis in the figure are normalized to an interval [0, 1]; the project timeline is normalized by total weeks for each project and the defect count by the total defect count observed for each project respectively. A diagonal line helps making a visual distinction of the shape of cumulative defect count profiles. If the defect inflow profile starts under the diagonal line but by mid-way through the project passes over the diagonal, it is S-shaped, while if it starts and stays over the diagonal for most part of the project timeline, the profile is concave-shaped. The projects where the defect inflow profile starts with slow growth rate (i.e. under the diagonal line) and stays below this line for most part of the project are regarded as convex-shaped.

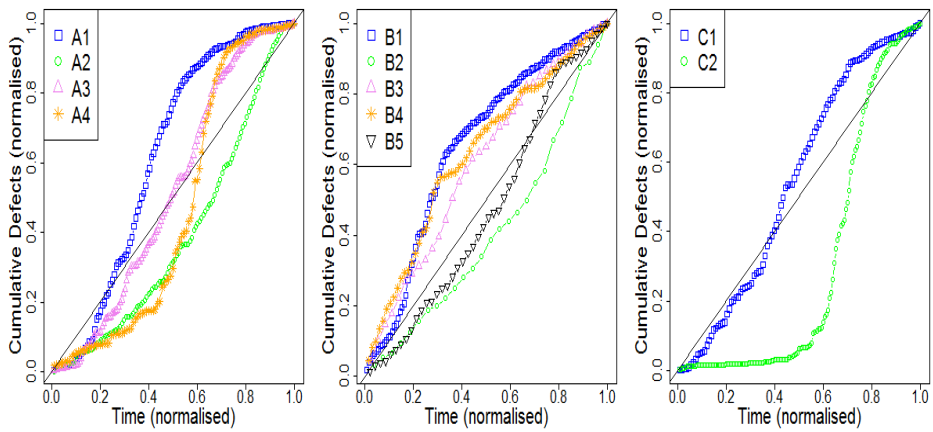

Figure 35: Cumulative defect inflow profile (or cumulative defect count) for case units 1-3.

It is observed from Figure 35 that a given case unit may have projects with different cumulative defect inflow shapes. Notably the case unit 1 has most projects (A1, A3 and A4) following the S-shape, while only one project A2 follows the convex profile. Case unit 2 has three out of five releases following the concave cumulative defect inflow (B1, B3 & B5), while the other two follow the convex profile. In case unit 3, project C1 is S-shaped while the anomalous project C2 is closer to convex shaped than to any other shape. Based on the overall results from the evaluation of SRGMs on each case unit, the software reliability growth models that suit the different objectives for each case unit are summarized in Table 15.

Table 15: Summary of recommended SRGMs for case units 1-3.

Case unit (domain)	Software development process	Observed shape of defect inflow profile	Recommended SRGMs		
			For testing resource(s) allocation	For release readiness assessment	
				Only using current project data	Using historical information
1. Automotive	V-model	S-shape, Convex	Gompertz	Logistic	Logistic
2. Telecom	Lean + Agile	Concave, Convex	Gompertz	Logistic	Musa-Okumoto
3. Defence Equip	Waterfall	S-shape, Convex	Logistic	Gompertz	Logistic

It is noted that:

- For the company developing embedded software in the automotive domain following the V-model for software development (case unit 1), the defect inflow profile of projects is dominantly S-shaped with few exceptions that are convex-shaped. In this case for decisions regarding optimal allocation of testing resources, the Gompertz model performs best, while the Logistic model is recommended for assessing the release readiness with or without using past information.
- In the case of Ericsson using streamline development for developing embedded software in the telecom domain, project releases in general produce defect inflow profiles that are concave and convex-shaped. The best performing model for optimal test resource allocation is Gompertz. For release readiness assessment using current project information, the Logistic model is the best choice while when previous projects information is used, the Musa-Okumoto model gives the best results.
- For Saab EDS developing embedded software in the defence equipment domain following the waterfall development process (case unit 3), the projects defect count profiles are S-shaped and convex-shaped. The Logistic model is recommended for making optimal test resource allocation decisions in this case. While for making the assessment of release readiness with and without using historical project information, the Logistic and Gompertz models are best respectively.
- Overall, for making decisions with respect to optimal allocation of resources during a project, the Gompertz model is the

recommended unless the software development process is waterfall where the Logistic model is better.

- For making an assessment if a product is ready for release or will be ready for release by a given date by using only defect data from the current project, the Logistic model is recommended except in the case of the waterfall development process, where the Gompertz model should be preferred.
- When information in form of a growth rate is used from earlier projects, the assessment of release readiness can be done more accurately by using the Logistic model, except in an agile development process, where the concave-shaped model Musa-Okumoto is more accurate for such assessment.

3.14.5 How to make the choice of SRGM more effective?

Next we investigate how different reliability growth models perform if we apply them based on the observed defect inflow profile rather than evaluating them on company/case unit basis. Based on the shape of observed defect inflow profiles from Figure 23, Figure 27, and Figure 31, the projects/releases at three case units are classified into groups of S-shaped, concave, and convex-shaped defect inflow profiles (shown in Figure 36).

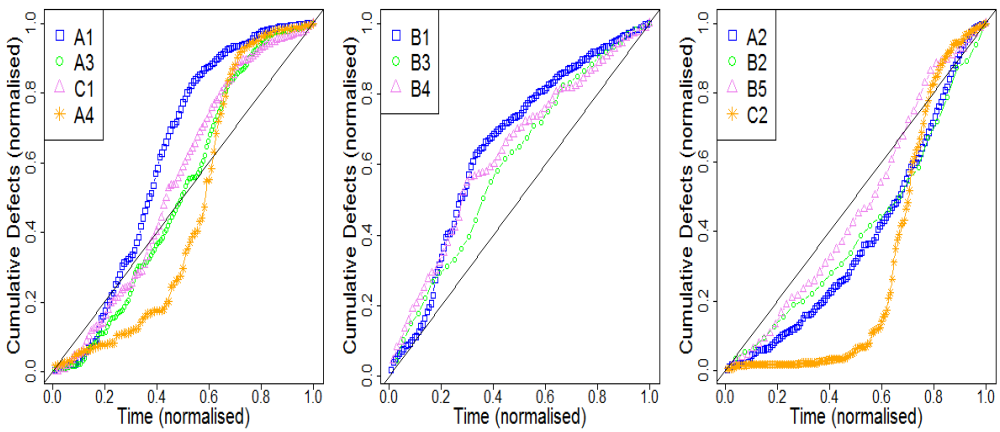

Figure 36: Cumulative defect inflow profile (or cumulative defect count), cases classified into S-shaped, concave and convex-shaped respectively.

Although it is easy to classify the projects based on the observed shape of defect inflow profiles when a project is finished, it is not straightforward to select an appropriate SRGM for a project when the project is on-going.

We propose to use the trend of observed defect intensity to predict the shape of a defect inflow profile, the overall defect intensity of all projects/releases classified according to their shape is presented in Figure 37. A vertical line is also drawn when a software project is half-way through its timeline, where we aim to predict the shape of full defect inflow profile. As it can be seen from the figure, the defect intensity for S-shaped profiles maximizes close to the middle of project timeline before it starts to fall down. For concave-shaped cumulative defect profiles, the defect intensity maximizes very early in the project and then decreases smoothly over the rest of the project. On the other hand, projects that have a convex-shaped cumulative profile, the defect intensity keeps increasing until late in the project timeline before eventually reaching a maximum and then declining. These projects show what in industry is called late defects, which tend to put a lot of pressure on the project teams near the release time.

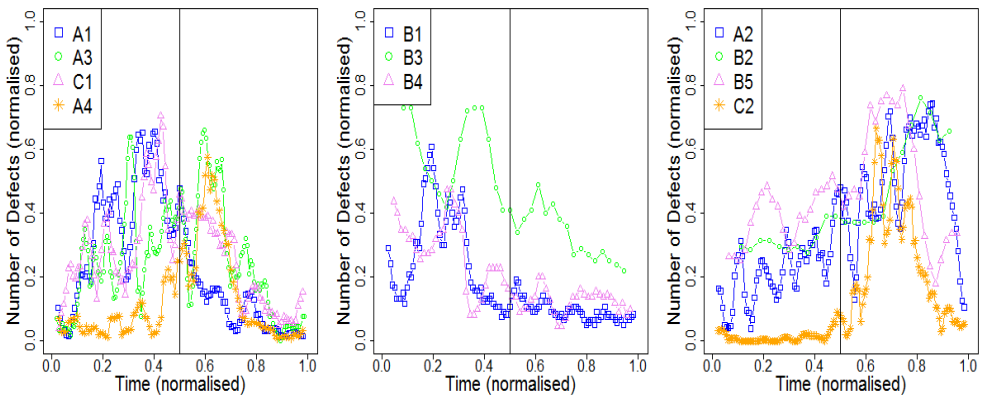

Figure 37: Defect inflow profile (or defect intensity), cases classified into S-shaped, concave and convex-shaped respectively.

To predict the shape of defect inflow during an on-going project (i.e. using partial defect data), we plot only the observed defect intensity of a project (in this case when a project is half-way through its timeline) and plot a linear trend line for this period, as shown in Figure 38. It is clear that for cumulative defect profiles that are S-shaped and convex-shaped, the defect intensity until half-way through the project is increasing, while for concave-shaped cumulative defect inflow, the overall defect intensity trend at the middle of project is decreasing.

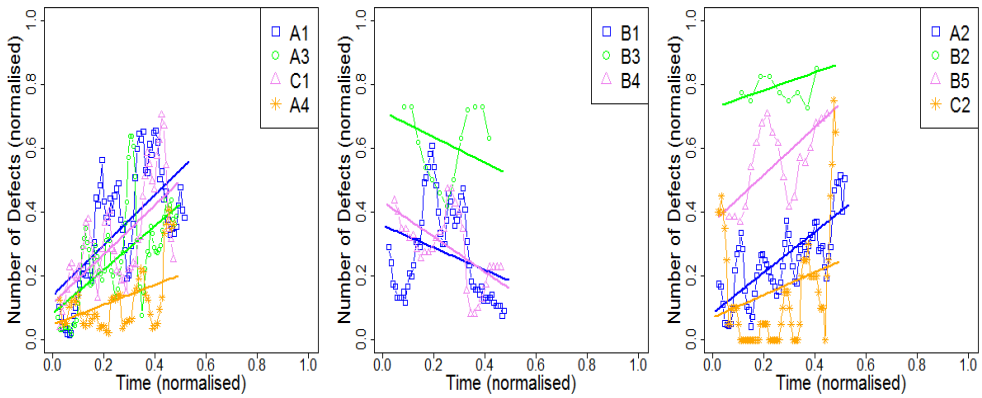

Figure 38: Partial Defect intensity with linear trend line, cases classified into S-shaped, concave and convex-shaped respectively.

Thus observing a linear trend of defect intensity only allows us to identify the concave-shaped defect profiles but not to distinguish between S-shaped and convex-shaped. For this distinction we split the observed region before and after the maximum defect intensity (for the observed period) is reached. Overall if the maximum defect intensity is reached near to half-way though the project and then the defect intensity trend is decreasing, the predicted shape is the S-shaped, while if the trend of defect intensity after it has reached its maximum is increasing then the predicted shape is convex. Only project C2 does not satisfy the classification method described here but it is also the project with an unusually low growth rate for much of the early phases of project.

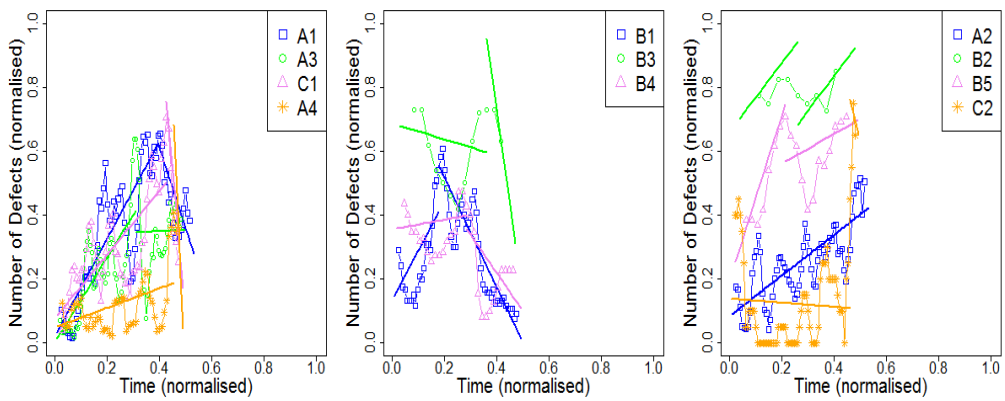

Figure 39: Partial Defect intensity with linear trend line before and after the maximum intensity is reached.

How to predict the shape of cumulative defect inflow by analysing the trend of defect inflow intensity of observed data is summarized in Table 16.

Table 16: Summary of classifying projects based on predicted shape of defect inflow using observed trend of defect intensity.

Projects/ Releases	Defect inflow intensity trend until half-way through the project			Predicted shape of defect inflow profile
	Overall trend	**Trend after reaching maximum**	**Defect inflow intensity trend characteristics**	
A1, A3, A4 & C1	Increasing	Decreasing	Defect inflow intensity first increases, maximizes near to half-way and then decreases	S-shape
B1, B3 & B4	Decreasing	Decreasing	Early defects, defect inflow intensity maximum early then decreases smoothly	Concave
A2, B2, B5 & C2	Increasing	Increasing	Late defects, defect inflow intensity trend is positive throughout half-way of project timeline	Convex

Re-evaluating the results based on classification of predicted shape of defect inflow profiles are presented in Figure 40 and Figure 41. As before SRGMs are ordered based on the indexed value of MSE and BPRE that indicate which models are best for a given purpose: Assisting in allocation of testing resources and assessment of release readiness.

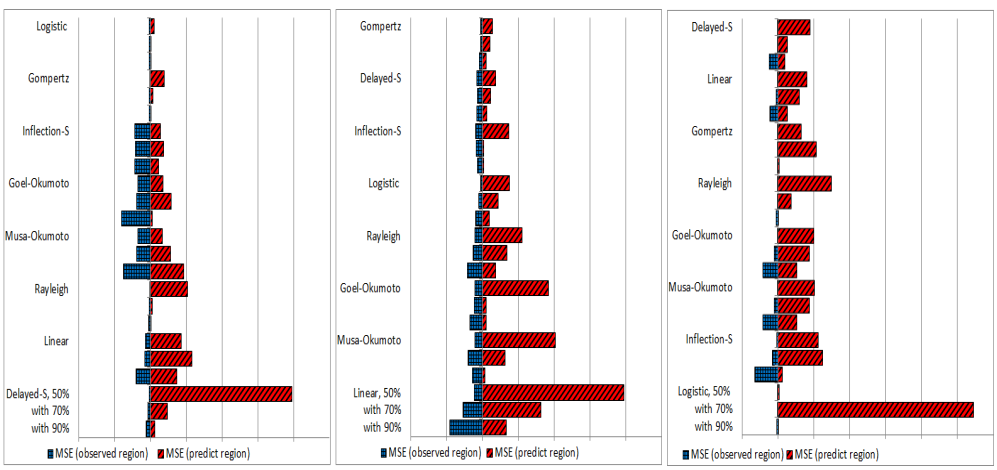

Figure 40: Mean Square Error for classified cases, S-shaped, concave and convex defect profiles respectively.

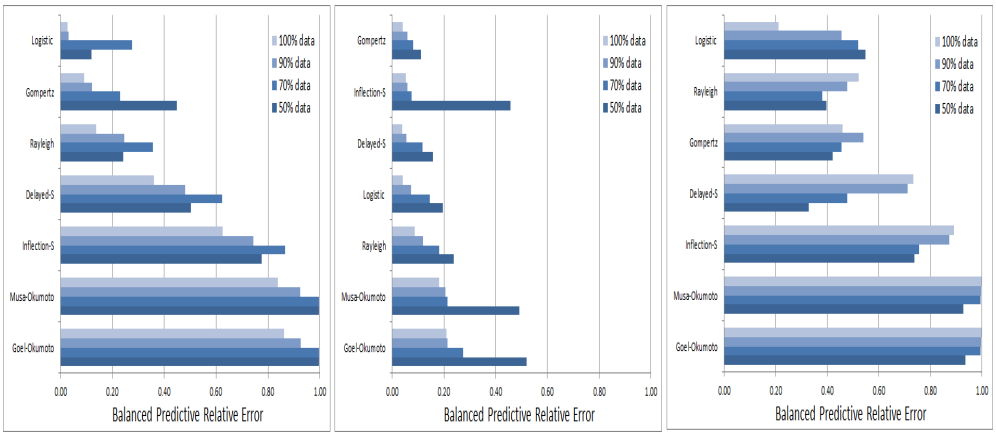

Figure 41: Balanced predictive relative error for classified cases, S-shaped, concave and convex defect profiles respectively.

The recommended model for each (predicted) category of defect inflow shape is summarized in Table 17. After classification, it is observed that:

For S-shaped defect inflow profiles, the Logistic model is best for assisting in decisions regarding optimal testing resource allocation as well as for using SRGMs for assessing release readiness.

If the predicted shape of defect inflow profile is concave, most models give good prediction accuracy but overall, the Gompertz model is best for both purposes.

For projects where the defect inflow profiles are convex-shaped due to late defects, most models fail to make precise predictions. The best model for helping with resource allocation decisions in such projects is Delayed-S, while for assessing release readiness is the Logistic model.

Table 17: Summary of recommended SRGMs for projects classified based on (predicted) shape of defect inflow profiles.

Predicted shape of defect inflow profile	**Recommended SRGMs**	
	For testing resource(s) allocation	**For release readiness assessment using current project data**
S-shape	Logistic	Logistic
Concave	Gompertz	Gompertz
Convex	Delayed-S	Logistic

To test if it actually helps to predict the shape of defect inflow profile and use this information to select an SRGM for test resource allocation or assessment of release readiness, box plots for indexed MSE values (for predicted region) and indexed BPRE values with and without classification are presented in Figure 42. The hypothesis H_b and H_c formulated earlier are also tested.

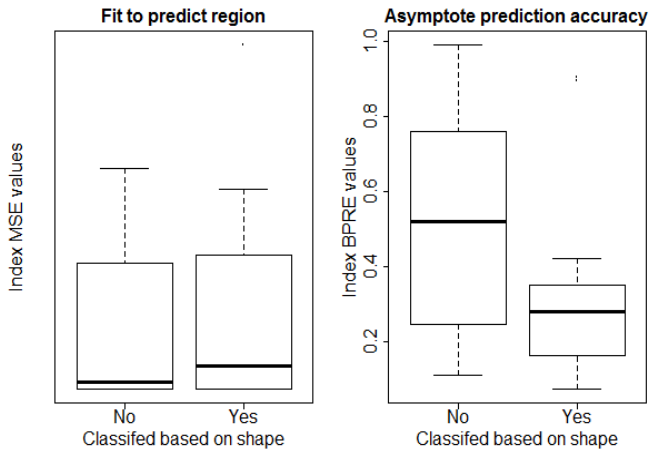

Figure 42: Box plot for index MSE and index BPRE values before and after classification based on predicted shape of defect inflow profile.

Testing the hypothesis H_b for all projects using independent samples Mann-Whitney U test, we are unable to reject the null hypothesis (p-value = 0.564). There is no statistically significant difference between a model's ability to fit the predicted region of defect inflow (indexed MSE predict values) with and without classification into groups based on the shape.

While testing hypothesis H_c for all projects, we are able to reject the null hypothesis (p-value = 0.026) that show that a statistically significant improvement is achieved in the overall performances of asymptote prediction (indexed BPRE values) by classifying projects based on the predicted shape of their defect inflow. Thus, release readiness assessment can be improved significantly if the shape of defect inflow profile is predicted and this information is used to select the most appropriate model.

3.14.6 Threats to validity

We address the threats to validity in the manner as described by Wohlin et al. [50]. There exists a threat to internal validity to this study regarding what is considered to be a defect. To minimize the threat, a common definition of defect was used which was verified at each case unit before the data was collected. Another threat to internal validity of this study arises from using non-linear least square (NLLS) for

curve fitting/parameter estimation, while a widely recommended technique is the maximum likelihood estimation. NLLS has been used in many earlier studies mainly the ones comparing the performance of different models [76], [57], [115], which is also the case here. Also since the main objective of this study is to compare between the models, the parameter estimator method does not pose a serious threat.

A threat to conclusion validity is present due to the assumption that the total reported defects is the actual asymptote, but since all the projects used in this study have been completed at least one year before this study was initiated, the threat of this assumption being invalid is hence minimized. There also exists a conclusion validity threat due to the use of a new metric for assessment. For measuring and comparing asymptote prediction accuracy, the new metric BPRE is used, which is an improvised version of the widely used metric Predictive Relative Error (PRE). Defined in detail in [121], BPRE is used to ensure that there is consistency (symmetry) between the metric values for over and under predictions. Taking only the magnitude value and ignoring the sign when averaging prediction accuracy to give Avg+ further ensures that over and under predictions are treated equally and do not cancel each other's effect. Thus, these changes help us to enhance the conclusion validity of our study.

External validity concerns with the generalizability of results in settings outside of the particular study. In the work presented here, we have evaluated eight well studied and widely accepted SRGMs on industrial embedded software projects. Seven plus one different reliability models with different shapes and structures were chosen to represent the sample; the selected models are based on their widespread use in reliability literature. The data used for this study came from three leading companies with heavy focus on embedded software development but with a different product portfolio and different software development processes respectively. The wide variation in development processes and the defect inflow profiles helped us to keep the study quite general. Thus, the results obtained from this study have wider applicability within the embedded software domain. Results concerning applicability of SRGMs for optimal allocation of testing resources, assessment of release readiness, and usability of historical information for improving the prediction accuracy of given models have also applicability beyond the embedded software domain.

3.15 Recommendations for applying SRGMs in industry for embedded software development

By evaluating SRGMs using real data from industry and addressing practically important research questions in this study, we seek to increase their industrial adoption. Based on the results discussed in this work, we provide some guidelines for companies looking to apply SRGMs for reliability assessment:

a. **Collect the right data from testing**: Understand the software development and testing process at your company. The defect reporting and management process in a company greatly influences which data from which database and from which development/testing phases are best suited for reliability analysis.
b. **Perform exploratory analysis of data:** Visualizing the data is the first and essential step for choosing the right model. Exploratory analysis of past and current projects provides understanding of the shape of the defect inflow and also how similar the past and current projects are with respect to their defect inflow profiles.
c. **Define goals for applying SRGMs**: Reliability growth models can be used to assess the release readiness at the end of a project, and/or to allocate testing resource to ensure the desired reliability/quality is achieved in a given timeline. Some models are better suited for one purpose than others and thus, defining the goals upfront helps the analyst to choose the right model for the given purpose.
d. **Model selection:** It is observed that performance of different SRGMs vary widely. Therefore, instead of selecting a model on ad-hoc criterion, pick a few models based on the observed defect inflow profiles from past projects and current projects, evaluate these models on the data, and select the best model.
e. For the embedded software domain, the current work suggests that irrespective of different development/testing process, trend models (Logistic and Gompertz) perform quite well while concave models were the least accurate.
f. **Classify based on predicted shape:** This study suggests that it is possible to predict the shape of the defect inflow profile during an on-going project. Predicting the expected shape helps to select the appropriate SRGM for a given purpose.
g. **Use of historical information:** Use information from earlier projects to increase the prediction accuracy. When the development and testing process is similar to earlier projects, reducing the model

parameters provides an easy to use methodology for increasing predictive accuracy of these models. When applying these models early in the project timeline, past information can improve the models' prediction accuracy and increase also their prediction consistency.

3.16 Conclusions

In this paper we have evaluated eight established SRGMs on a number of large software projects within the embedded software domain from three different companies. With this unique setting and rich data, we set out to evaluate:

3.16.1 Which SRGMs are the best to assist decisions for optimal allocation of testing resources?

For assisting decisions on how to manage testing resources effectively by evaluating different reliability models across the different projects in the domain of embedded software development, it is observed that the Gompertz model is the best for software development processes that are either V-model based or follow lean and agile software development processes. The Logistic model should be chosen for waterfall development process.

Predicting the shape of defect inflow profile, the Logistic, Gompertz and Delayed-S models were found to be best to assist decisions for optimal allocation of testing resources for S-shaped, concave, and convex-shaped defect inflow profiles.

3.16.2 Which SRGMs are the best for assessing the release readiness of a software system?

Overall, trend models (Logistic and Gompertz) performed best from the perspective of asymptote precision, which is an important property for the assessment of release readiness. The Logistic model proved to be the best among tested models for all except the waterfall processes, where the Gompertz model is found to provide the best results.

When the shape of defect inflow is predicted, for the assessment of release readiness, the Logistic model is shown to be the best for the S-shaped and convex-shaped defect inflow, while the Gompertz is the best model for concave-shaped defect profile.

Another question evaluated in our study is regarding usability of historical information in terms of growth rates from earlier projects for improving long-term asymptote predictions:

3.16.3 Does using information from earlier projects improve release readiness assessment?

Using information in terms of growth rates from earlier completed projects is shown to significantly improve the asymptote accuracy across the models in software development following the V-model. In the case of the waterfall development process, the best model obtained after using the growth rate is comparatively much more accurate than the model without using the growth rate. For the lean and agile software processes, using the growth rate did not improve the overall asymptote accuracy of the models. Thus depending on the software development process, using the growth rate from past projects can significantly improve the asymptote prediction accuracy or help with selecting a better model to assess release readiness.

3.16.4 How to make the choice of SRGM more effective?

It is observed in this study that the shape of a defect inflow profile can be predicted for an on-going project by analysing the trend of the observed defect intensity as early as when a project is half-way through its timeline.

It is further noted that predicting the shape of expected defect inflow profile although do not significantly improve the ability of models to fit the predicted region, but a statistically significant improvement is achieved in asymptote prediction accuracy. Thus, with respect to assessing release readiness it is useful to predict the shape of the defect inflow for the current project and to use this information to select the most appropriate model.

In this study, two of the important and practically relevant questions are analysed in relation to applying software reliability growth models in the context of developing embedded software. Using data from three different leading companies from their respective sectors, the study provides useful and practical information for the application of SRGMs during an on-going project for distributing testing resources more effectively and to better assess their release readiness, which are important decisions from the project management and software quality perspectives. Further studies in a similar direction will ensure that reliability engineering is more widely adopted in the industry and provides useful information to industrial reliability practitioners and project managers to effectively manage the project cost/resources and quality of software.

ANALYZING DEFECT INFLOW DISTRIBUTION OF LARGE SOFTWARE PROJECTS

Abstract— Tracking and predicting quality is a major challenge in large and distributed software development projects. A number of standard distributions have been successfully used in reliability engineering theory and practice, common among these for modelling software defect inflow being exponential, Weibull, beta and Non-Homogeneous Poisson Process (NHPP). Although standard distribution models have been recognized in reliability engineering practice, their ability to fit defect data from commercial software systems is not well understood. Lack of knowledge about underlying defect inflow distribution leads to difficulty in choosing appropriate SRGMs and uncertainty about applicability of different statistical methods for further data analyses. In this paper we explore the defect inflow distribution of total of fourteen large software projects/release from the two industrial domain and open source community. We evaluate six standard distributions for their ability to fit the defect inflow data and also assess which information criterion is practical for selecting the distribution with best fit. Our results show that beta distribution provides the best fit to the defect inflow data for all industrial projects as well as majority of OSS projects studied. Finding the underlying distribution of defect inflow is useful for applying appropriate statistical techniques for data analyses and also for selecting the appropriate SRGMs for modelling reliability. The information about defect inflow distribution is further useful for modelling the prior beliefs or experience as prior probabilities in Bayesian analysis.

Keywords—Software; SRGM; Defect Inflow; Beta distribution; Software reliability growth models; Automotive domain; Telecom; Open Source Software

3.17 Introduction

Tracking and predicting quality is a major challenge in large and distributed software development projects. Software defects found during development provide an observable and useful indicator to track and forecast software reliability, an important measure of quality. Software reliability measures are primarily used for [7]:

- Planning and controlling testing resources during software development, and
- Evaluating the maturity or release readiness of software before the release date.

Software Reliability Growth Models (SRGMs) are widely used methods for quantitative assessment of software reliability [122]. A large number of models (SRGMs) have been proposed over last three decades [65] based on both, the Frequentist and Bayesian statistical approaches. But with no standard method to choose the most appropriate model, and more than 100 SRGMs [68] to choose from, the selection of right model is a major challenge.

In this paper we address the problem of selecting the appropriate SRGM model. The current way of selecting the appropriate SRGMs are either empirical (based on expert opinions) or analytical (based on testing a subset of models and evaluating their performance before picking the best model). The problem with these approaches is that either the selection is subjective or the best model selected via analytical evaluation is only as good as the subset of models evaluated.

Our method is based on using statistical method to find a family of defect distribution first, which reduces the number of models to evaluate by an order of magnitude. We propose that the selection of candidate SRGMs to be done by identifying the underlying distribution family of the defect data. Understanding underlying defect distribution family is important, according to Okamura, Dohi and Osaki [122]

"When the number of total software faults is given by a Poisson random variable, the mean value function of NHPPP-based SRGMs is dominated by only failure time distribution. That is, the essential problem can be reduced to what kind of probability distribution is suitable for representing the failure time distribution."

Knowledge of underlying defect distribution family is also important for Bayesian approaches for modeling software reliability where initial knowledge about software reliability is coded in form of prior distribution.

A number of standard distributions have been successfully used in reliability engineering theory and practice, most common among these for modelling software defect inflow being exponential, Weibull, beta and Non-Homogeneous Poisson Process (NHPP). Rayleigh model is member of Weibull family which has been widely used for software project life cycle defect modelling [9]. Although standard distribution models have been recognized in reliability engineering practice, their ability to fit defect data from real software systems is not well understood. The low understandability of underlying distribution of defect data further leads to uncertainty on which statistical techniques can be uses (for example t-tests) or if a given SRGM is appropriate or not for given data. The research objectives of this study are to:

- *Explore which statistical distribution fit best to the defect inflow from large software projects, and*
- *Explore how different information criteria differ in selection of best distribution fit.*

Using defect inflow data from nine large software projects/releases from two different industrial domains and five large open source software projects we focus on analysing the defect inflow distribution during software development and testing. We explore different distribution families' ability to fit to the defect data from industry.

- Finding out the distribution that fits best to observed defect inflow data is helpful for:
- Enhancing the understanding of defect inflow profile and underlying process of defect discovery process.
- To choose the correct statistical analysis method, it is important to know the distribution of the data. Different statistical methods usually have underlying distributions assumptions. For example to statistically determine if two projects defect inflow differs significantly or not, using a parametric or non-parametric method depends on among others if or not the defect inflow data for projects is distributed normally or not.
- Visualization and simulations, knowing the distribution of given data helps with easy visualization and tracking the difference between different projects data using only a few parameters. It also allows generation/simulation of data for analysis of different scenarios.
- Knowing the distribution of defect inflow data is useful for selecting the right model for modelling/forecasting reliability growth. For example Rayleigh model/function used to model software reliability is a specific instance of one of Weibull family.

Thus if the defect inflow data do not follow Weibull distribution, Rayleigh model is unlikely to give good results for modelling reliability in such a case.

- The information on distribution of data is also useful in Bayesian analysis to describe the initial knowledge provided as prior probability (distribution) which can be updated in light on new data.

This is an extended version of authors earlier work [123] evaluating the defect inflow distribution for automotive software projects. In this paper we extend the analysis by adding five consecutive releases from another large software project from a telecom domain company and further also analyse the defect inflow distribution of five open source software projects. The additional data used adds software projects from different industrial domain as well as open source community which follows different process for software development and testing, thus the external validity of results obtained earlier [123] is further strengthened.

The rest of the paper is structured as follows. Section 3.18 presents the background for our research. Section 3.19 describes the important related works in this area. In section 3.20 we describe in detail the research methodology and data used for this study, while Section 3.21 presents the findings from the study. Finally section 3.22 presents our conclusions.

3.18 Background

3.18.1 Software Defects and Reliability Growth Models

IEEE standard 1044, Classification for Software Anomalies provides common vocabulary for terms useful in this context, according to the standard [9]:

defect: *An imperfection or deficiency in a work product where that work product does not meet its requirements or specifications and needs to be either repaired or replaced.*

error: *A human action that produces an incorrect result.*

failure: *(A) Termination of the ability of a product to perform a required function or its inability to perform within previously specified limits Or (B) An event in which a system or system component does not perform a required function within specified limits.*

fault: *A manifestation of an error in software.*

problem: *(A) Difficulty or uncertainty experienced by one or more persons, resulting from an unsatisfactory encounter with a system in use Or (B) A negative situation to overcome.*

Software defect can be defined as issue or deficiency raised due to use of software product which causes it to perform unexpectedly [8]. Defects can be introduced at different phases of this continuous process and testing is the phase where focus is on discovering and eliminating these defects. If we plot the discovery of defects against time we get the defect inflow distribution over the development cycle. The distribution of defect inflow can be used for various purposes, form enhancing the understanding of defect creation, discovery and fixing process to modelling reliability of software system using software reliability growth models.

Software reliability models can be categorized broadly as static and dynamic models. Static models use attributes from project and the product (source code) to predict the number of defects, while dynamic models uses the defect discovery data during development and/or testing and use statistical models to estimate the reliability of given product. Static models are termed static in the sense that they use parameters, which are based on earlier projects; the current project/product is considered an additional observation from same sample. On the other hand dynamic models estimate their model parameters from multiple data points from the same project and use it to predict/forecast for the same project going forward. Dynamic software reliability models can be further classified in categories based on which data they use, Rayleigh model being an example of first category which model the entire development process while other category being represented by like of exponential model that models the data from back-end testing phase [69].

3.18.2 Software Defect Inflow Distributions and Model Selection

A number of SRGMs have been proposed and evaluated, a roadmap on software reliability engineering is given by Lyu [68] [124]. Different models have different process assumptions and different distributions are better suited for varied defect inflow profiles. Musa [72] and Goel [71] with their work showed that different families of distributions are better suited for applications with different characteristics.

Lyu [8] propose that to assess and predict reliability of software systems, proper measurement and collection of failure data is needed over given system's testing and/or operation. Further the underlying process of software development process

also needs to be understood for selecting the appropriate reliability model. Thus most studies proposing or evaluating SRGMs have applied mathematical models to failure data which is well structure or filtered from specific testing phase. This methodology while works well for assessment of maturity of software artefact once the software is completely developed and tested, but is not optimal for in-process defect prediction; thus there is need for models and their validation on defects from full project development cycles.

Weibull families of curves are one of the best known distributions in reliability engineering; reliability models such as exponential and Rayleigh model are special cases of Weibull distribution. Misra [125] applied exponential model successfully to estimate defect arrival rates for NASA shuttle's ground system software. Rayleigh model have also been used widely to model the defect inflow during software life cycle phases [126], [127], [128]. Other distributions like beta distribution and Logistic population growth models have also been used in reliability engineering; Non-Homogenous Poisson Process (NHPP) based models have been used widely for software reliability modelling.

With number of SRGMs proposed, no distinct way of selecting models and lack of exploratory analysis of data – a common way to differentiate between different models it to apply them to same data sets and do comparative studies. Ullah et al. [76] compared between eight SRGMs on dataset from fifty industrial and open source projects, Pham [66] compared and reviewed common NHPP based SRGMs for their ability to fit data from real-time control system. A number of SRGMs have also been compared for their ability to fit data from telecom domain in study by Staron and Meding [15]. Seven SRGMs have also been compared on their performance on predictive power using partial in-process data from real projects in Rana et al. [57]. Contrary to earlier studies where different software reliability models have been compared and assessed on their ability to fit defect data, in this study we compare between standard distributions such as Weibull, beta, exponential etc. known to do well in reliability engineering and check which distribution fits best to defect inflow behaviour of large software projects form different industrial domains within embedded software development.

Selection of models have been discussed in number of earlier studies, Stringfellow and Andrews [74] propose empirical method fitting data iteratively to different models and selecting a SRGM based on proposed criteria. Akaike Information Criteria (AIC), based on the log-likelihood function is recommended to be used for selecting appropriate model by Khoshgoftaar and Woodcock [75]. Sharma [73] also recommends that before making a selection different models should be first tried, compared and evaluated. Even after certain attempts and currently active research to find a standard selection method/criteria, common agreement on model selection is

not reached which highlight the further need that data be studied properly to understand the process and different distributions/models be tried before making a final selection of reliability model to be used for given defect data and reliability modelling where the current study attempts to make its contribution. In the evaluation of different distributions we also evaluate how different information criteria's differ and if that have any impact on the choice of best distribution.

Evaluation of SRGMs on industrial data and specifically into particular domains is scarce [98]. Wood [13] applied eight SRGMs on industrial defect inflow data and found significant correlation between pre-release defects and post release defects. A comparison of SRGMs and their use in practice within consumer electronics embedded software in is also presented in [14]. The present study complements earlier work in defect data analysis and application of reliability models to industrial defect inflow data.

3.19 Related Work

Different family of distributions has been used for modelling software reliability in previous studies. Zhao [129] proposed to use beta distribution to indicate software testability, the author theoretically prove that testing effort and test values can be simultaneously expressed through the distribution. Mullen [130] shows evidences of lognormal distribution of software failure rates and discuss their origins. In a later study Mullen [131] highlights the dissatisfaction with large number of software reliability models and lack of single flexible general model and introduces software reliability growth model based on lognormal distribution. Gokhale and Trivedi [84] contend that finite failure NHPP models can capture constant, monotonic increasing, and monotonic decreasing failure occurrence rate per fault, but fail to capture cases with increasing/ decreasing nature of failure occurrence rate per fault. The authors propose log-logistic reliability growth model for such cases. Zhou and Davis [132] analysed time related bug reporting patterns of eight popular open source projects evaluated, they showed that open source projects exhibited similar reliability growth pattern as closed projects and Weibull distribution provides a good fit to the defect inflow distribution.

According to Okamura, Dohi and Osaki [122]different statistical distributions such as exponential, gamma, Pareto and Weibull have been used to model the failure time distributions for most of NHPP-based SRGMs. On the contrary models based on normal distribution have not been given more attention and the authors [122] proposes SRGM based on normal distribution. Kharchenko et al. [133] also highlighted the model selection problem for applying SRGMs in practice. The authors classified SRGMs on five different criteria with special emphasis on

distribution family of failure intensity. A method of choice (for selecting SRGM) is presented based on the assumptions of different models and features of software engineering and testing process. Our study contributes in this direction by providing empirical evidence of which distribution family best fits to defect inflow from large software projects from industry and open source community.

Karunanithi, Malaiya and Whitley [134] contend despite applicability of a model can be established for large collection of data sets, a certain degree of stiffness (model's inability to correctly simulate failure process trend for new data set) cannot be completely ruled out. The authors explored the use of neural networks for modelling software reliability. Littlewood [135], in his proposed Bayesian reliability growth model used gamma distribution family to model the prior probabilities justified by its flexibility, correct range of parameters $(0, \infty)$ and mathematical tractability. Kuo et al. [136] presented Bayes inference methodology for NHPP models with S-shaped mean value functions. The authors Bayes methodology for Ohba-Yamada model which assumes mean value function to follow gamma distribution with shape parameter of 2 is proposed and generalized to class of gamma distribution growth curves with known shape parameter and unknown scale parameter. Neil et al. [137] used the Bayesian belief networks for predicting the reliability of military vehicles, in such applications modelling prior beliefs with regard to failure rate is an important step which can be enhances by having better understanding of defect/failure distribution from historical projects/products. By identifying which distribution fits the historical projects and which information criteria can be used for selecting the best fit distribution – our results strengthen the notion that defect inflow distribution be studied for historical projects. In Bayesian approach to modelling software reliability, understanding historical projects defect distribution properties reveal information that is useful in precise modelling of prior probability distribution.

An emerging area which has wide use in software engineering is information visualization, the technology uses graphics techniques for visualizing abstract entities [138]. Visual representation provide an alternative approach for exploring the data for enhancing the understanding of underlying processes and patterns and may also be used as tool to convey information more effectively. Hora et al. [139] argue that while number of tools exist to extract and analyse information regarding evolution of software systems, little is known about the evolutionary behaviour, lifetime, distribution and stability of software defects. A tool named BugMaps is presented that can automate the retrieval and mapping of bugs from relevant databases. The tool provides interactive visualization which is useful source of information for decision support. Empirical software engineering evaluates the applicability and performance of different models and techniques in the practical context with the aim of documenting knowledge and provides support for making

decisions. Garcia et al. [140] observe that using only pre-determined hypothesis and using standard statistical techniques, it might be difficult to reveal the non-anticipated relationships and patterns from the data. In this study we do not presume the defect inflow distribution and use exploratory methods to determine the distributions of historical projects. Understanding the underlying defect distributions for historical projects can form the basis for visualization of defect inflow, exploring the likely effects of process changes on defect inflow and selection of appropriate SRGMs for modelling software reliability.

3.20 Research Methodology and Data

The study is organized as an exploratory case study following classification of Robson [106], the main objective of the study is to explore which standard distribution family(s) are able to fit to software defect inflow data from large scale software projects with wide variations in their distribution characteristics. The research is organized as an embedded case study with the unit of analysis being each project. The similarities and differences based on different software projects are explored and highlighted in this study which suits the embedded case study design. The case study design overview is presented in Figure 43.

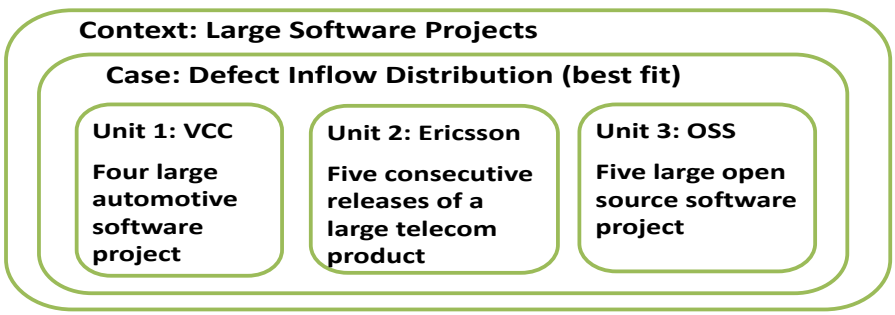

Figure 43: Overview of case study design

Table 18 shows the summary of the characteristics of the process used in the case units and mapping to their application domain.

Table 18: Overview of units of case analysis within this embedded case study

Unit of analysis	Application domain	Software development process for studied projects
Volvo Cars Group	Automotive	V-shaped software development mostly using sub-suppliers for implementation
Ericsson	Telecom	Agile development, mostly in-house
OSS	Open-Source Projects	Open source software development, projects from Apache and Mozilla

3.20.1 Case Units

In this study, the selection of case units was primarily driven to capture variation in the software development process and application domain within large scale software development projects from industry and open source community. Nonetheless limiting units of analysis to a manageable number and accessibility also played a role in the selection of these units. Two case units although allow a better comparison, but limits the generalizability, while if four or more case units are chosen it leads to difficulty in conducting cross case analysis including necessary details, thus three case units were selected for this study.

3.20.1.1 Company A: Volvo Car Group, A company from the automotive domain

Volvo Car Group (VCG) is a Swedish car Original Equipment Manufacturer (OEM), based in Gothenburg. VCG is developing software and hardware in a distributed software development environment. The size of the entire automotive project in terms of resources is substantially larger than the projects in the other application domains studied in this case study, due to the fact that both OEM and suppliers (first and second tier) are involved and car development projects are usually conducted using the product line approach with reference architectures ([110]). The software projects studied here come from the E/E (Electrical and Electronics) integration department within the VCG which deals with the integration of various software functionalities and responsible for the final assessment of full E/E hardware and software systems. All the projects studied have been completed in last decade and consist of different modules developed by different teams and tested within the development team (unit testing), while further integration and acceptance testing is done by dedicated teams in the integration department. All defects detected during all testing phases are reported in the central bug database used by the company which was also the primary source of data compilation for this study.

3.20.1.2 Unit B: Ericsson, A company from the telecom domain

Ericsson develops large software products for the mobile telecommunication network. The size of the organization during the study is several hundred engineers and the size of the projects is up to a few hundreds. Projects are increasingly often carried out according to the principles of Agile software development and Lean production system, referred to as Streamline development (SD) within Ericsson [111]. In this environment, various teams are responsible for larger parts of the process compared to traditional processes: design teams (cross-functional teams responsible for complete analysis, design, implementation, and testing of particular features of the product), network verification and integration testing, etc.

3.20.1.3 Unit C: Open source software projects

We used five large open-source software projects from Apache and Mozilla. While there is no strict software development process followed for all five projects, both Apache and Mozilla projects used in this study generally follow active development with development teams regularly making commits and fixing bugs. For details on the development process of Apache and Mozilla, readers are referred to work done by Mockus et al. [141] [142]. All issue reports marked RESOLVED, CLOSED, or VERIFIED with resolution set to FIXED were retrieved from the bug database - from these issues only ones identified as BUGs [143] are used in this study. The time period and number of defects is summarized in Table 19.

Table 19: Summary of projects time span and number of defects/issues

Case Unit	Project/Release	Time Period	Total number of Defects*/Issues
VCG	Project-A1	NA	6.7X
	Project-A2		14.4X
	Project-A3		2.0X
	Project-A4		X
Ericsson	Release-B1	NA	2.2Y
	Release-B2		Y
	Release-B3		1.3Y
	Release-B4		1.2Y
	Release-B5		1.6Y
OSS	Project- HTTPClient	Nov-2001 – Apr-2012	305
	Project- Jackrabbit	Sep-2004 – Apr-2012	938
	Project- Lucene-Java	Mar-2004 – Mar-2012	697
	Project- Rhino	Nov-1999 – Feb-2012	302
	Project- Tomcat5	May-2002 – Dec-2011	670

**Total number of defect for industrial projects are normalized by project within the case unit with lowest defect and time period is not provided due to confidentiality*

3.20.2 Data Collection and Analysis Methods

The data for this study is collected from the central defect database for each project. For industrial projects we used the definition for defect from IEEE 1044 standard [9], while defect inflow of a project is defined as "defect inflow is the number of non-redundant defects reported in the defect database" [15]. In industrial settings, what is reported and marked as a defect is closely controlled which assures the quality of data collected, but for open source projects, what is classified as a defect or bug[5] is not strictly controlled. The problem of misclassification of issue reports (issues classified as bugs which are actually not bugs) have been highlighted in

[5] Defect and bug are used interchangeably to refer to a non-redundant defect reported in the defect database.

earlier studies [144] [143]. Misclassification of bugs and non-bugs issues can be a serious threat for defect classification where bugs from the database are used to mark files as containing bug or clean. In this study we are interested in the inflow distribution of defects and not on defect classification thus misclassification does not pose a serious threat as long as the distribution of defects in all issues reported does not vary over time. Nonetheless to ensure that the distribution we analyse is of defects/bugs only – for open source projects we use issues which are identified as Bugs using manual classification in earlier study by Kim, Just and Zeller [143]. Using manually validated data not only ensures quality of data used but also provide higher consistency with industrial data where issues classification as defect is closely controlled.

Based on the existing reliability engineering literature, we selected six widely used continuous distributions to be evaluated in this study. A number of reliability growth models such as exponential model, Rayleigh models are based on these distributions. A summary of continuous distributions and their probability density functions (pdf) used in the study are presented in Table 20.

Table 20: Overview of distributions used in this study

No	Distribution	Notation	Parameters	Probability Density Function
1	Exponential	$Exp(\lambda)$	$\lambda > 0;$	$\lambda\, e^{-\lambda x}$
2	Weibull	$Weibull(\lambda, k)$	$\lambda > 0;$ $k > 0$	$\begin{cases} \frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1} e^{-(x/\lambda)^k} & ,x \geq 0 \\ 0 & ,x < 0 \end{cases}$
3	Beta	$Beta(\alpha, \beta)$	$\alpha > 0;$ $\beta > 0$	$\frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha,\beta)},$ $where\ B(\alpha,\beta) = \int_0^1 u^{\alpha-1}(1-u)^{\beta-1}\, du$
4	Gamma	$Gamma(k, \theta)$	$k > 0;$ $\theta > 0$	$\frac{1}{\Gamma(k)\theta^k}\, x^{k-1} e^{-\frac{x}{\theta}},$ $where\ \Gamma(k) = \int_0^\infty x^{t-1} e^{-x}\, dx$
5	Logistic	$Logistic(\mu, s)$	$\mu\ (real);$ $s > 0$	$\frac{e^{-\frac{x-\mu}{s}}}{s\left(1 + e^{-\frac{x-\mu}{s}}\right)^2}$
6	Normal	$\mathcal{N}(\mu, \sigma^2)$	$\mu\ (real);$ $\sigma^2 > 0$	$\frac{1}{\sigma\sqrt{2\pi}}\, e^{-\frac{(x-\mu)^2}{2\sigma^2}}$

To fit the distributions to the observed defect inflow data *fitdistr()* function available in *MASS* package in *statistical environment R* was used. The function uses maximum-likelihood fitting for fitting the distribution and estimating the distribution parameters. To assess which distribution fit best to the given project's defect inflow, we evaluated their goodness-of-fit using six different recommended criteria's listed in Table 21.

We provide a brief overview of commonly used information criteria - to analyse which criterion suits our purpose and does selecting one criterion over another could make difference between selecting one distribution over another.

LogLik (Log–likelihood): Likelihood function is a function of parameters of a function given the outcome. In other words likelihood is defined as "the hypothetical probability that an event that has already occurred would yield a specific outcome. The concept differs from that of a probability in that a probability refers to the occurrence of future events, while a likelihood refers to past events with known outcomes" [145]. The criteria to pick the best fit model is to pick the model with highest likelihood or log-likelihood which is the natural logarithm of likelihood function.

ML (Maximum Likelihood): $\mathrm{ML} = -2 * \mathrm{loglik}$, is same as Log-likelihood, with the difference that instead of casting the criteria as maximizing the logarithm of likelihood function, the criteria is set as a minimization problem with objective function as -2*LogLik. Thus both the criteria would pick the same distribution as best.

AIC (Akaike Information Criterion): $\mathrm{AIC} = -2 * \mathrm{loglik} + 2 * \mathrm{k}$, is another measure for assessing the relative quality of statistical models. The difference from ML being that it includes a penalty for higher complexity of model. By penalizing higher number of free parameters (k), AIC discourages over-fitting. For simple distributions with low number of free parameters or where the selection is between models of same parameters AIC will tend to select the same model as with using ML.

AICc (Akaike Information Criterion, correction): $\mathrm{AICc} = -2 * \mathrm{loglik} + \frac{2\mathrm{kn}}{\mathrm{n}-\mathrm{k}-1}$, AICc is similar to AIC with correction for finite sample size (n). Its penalty is higher than AIC for higher parameters. AICc converges to AIC when n is large or k is small. In other cases (i.e. where n is small or k is large) it is recommended to use the AICc [146]. In our case (for all four projects) as sample size is much greater than the distribution parameters ($\mathrm{k} = 1$ or 2), AICc is expected to give similar results as AIC.

BIC (Bayesian Information Criterion): $\mathrm{BIC} = -2 * \mathrm{loglik} + \mathrm{k} * \log(\mathrm{n})$, is another penalized-likelihood criteria for model selection from a finite set of models. While AIC and BIC have different theoretical assumptions, the practical difference between the two is the size of penalty. BIC applies more penalty for higher model complexity which increases as a logarithm function of sample size. There is always a chance that a big model is chosen using AIC regardless of sample size, while with BIC there is little chance of choosing too big model for sufficient sample size. On the other hand BIC has higher chance of selecting too small model than AIC for any sample size [147].

HQC (Hannan–Quinn Information Criterion): $\mathrm{HQC} = -2 * \mathrm{loglik} + 2 * \mathrm{k} * \log(\log(\mathrm{n}))$, provides a penalty size between that of AIC and BIC which increases with the sample size.

Table 21: Overview of information criteria for selecting distribution with best fit

No	Short	Long Name	Definition
1	LogLik	Log likelihood	Logarithm of the probability of observed outcomes given a set of parameter values
2	ML	Maximum Likelihood	$ML = -2 * loglik$
3	AIC	Akaike Information Criterion	$AIC = -2 * loglik + 2 * k$
4	AICc	Akaike Information Criterion (correction)	$AICc = -2 * loglik + \frac{2kn}{n-k-1}$
5	BIC	Bayesian Information Criterion	$BIC = -2 * loglik + k * \log(n)$
6	HQC	Hannan–Quinn Information Criterion	$HQC = -2 * loglik + 2 * k * \log(\log(n))$

Where $k = number\ of\ parameters$ *and* $n = number\ of\ observations$

3.21 Results

3.21.1 Defect Inflow Profiles

The cumulative defect inflow profile for the four projects analysed is presented in Figure 2 to Figure 46. The data used for all projects in this study is weekly defect count data, thus the horizontal axis in figure represents actual time normalized by total duration of project timeline, while Y-axis show the normalized defect counts. For defect inflow the numbers of weekly defects are normalized by maximum number of weekly defects over the given projects time period and for cumulative defect inflow it is normalized by the total number of defects/issues reported for the studied period.

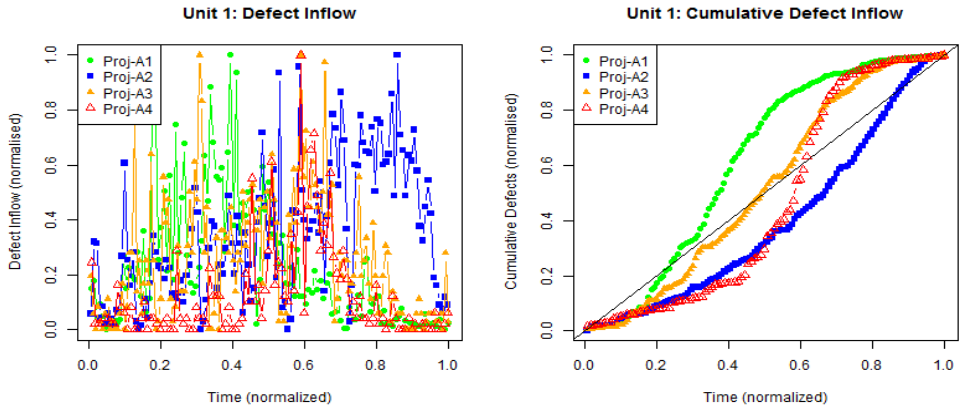

Figure 44: Defect inflow and cumulative defect inflow (normalized) for case unit 1

From the Figure 44 we can observe that most projects (all except project A2) have an S-shape. The cumulative defect inflow profile of project A2 resembles convex defect inflow; the specific difference of this project to the rest was due to the fact that defect reporting strategy was different from other projects - A2 included defect reports from a specific team which works in agile process and generally reports their defects in another database. It was included here since their contribution to this overall project was comparatively large. Software development in highly iterative process (using agile process) integrates new functionality and fixes to previously discovered defects in short cycles (weekly or more frequently), which allows for testing to proceed continuously. Earlier studies [112] have indicated that for such process the defect inflow is better predicted by concave SRGM models and linear models than S-shaped models which indicated toward the defect inflow of these projects not conforming to S-shape.

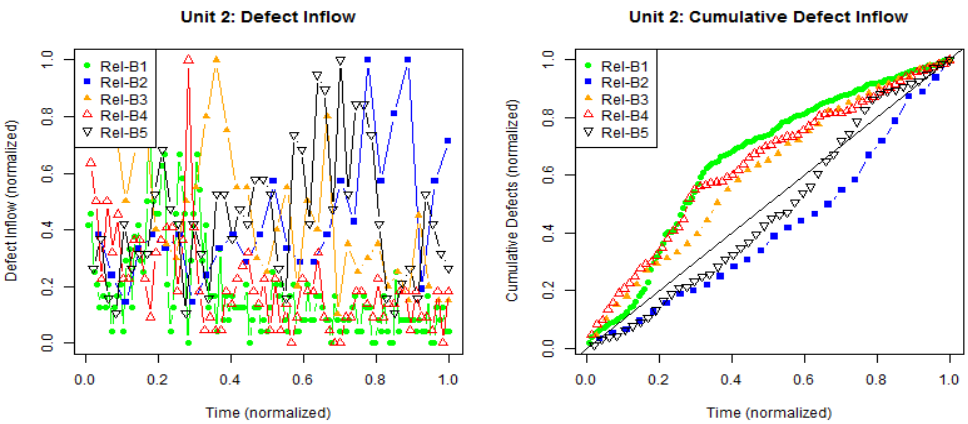

Figure 45: Defect inflow and cumulative defect inflow (normalized) for case unit 2

The defect inflow from case unit 2 (Figure 45) represents the defect inflow from software development process which is highly iterative. We do not see characteristics of S-shape as observed in projects from case unit 1; rather see cumulative defect inflows shaped concave or convex. Three out of five profiles are concave-shaped while the remaining two are convex-shaped. Concave-shaped cumulative defect inflows are characterized by defect intensity (number of defects per unit time, Figure 45 Defect Inflow) maximum early in the project, and then drops as project progresses giving a concave-shaped cumulative defect inflow profile. On the other hand defect cumulative defect inflows that are convex-shaped are characterized by slow growth rate of cumulative defects at the beginning, which stays until late in the project and eventually increasing as seen for Release-B2 and Release-B5 in Figure 45 for case unit 2.

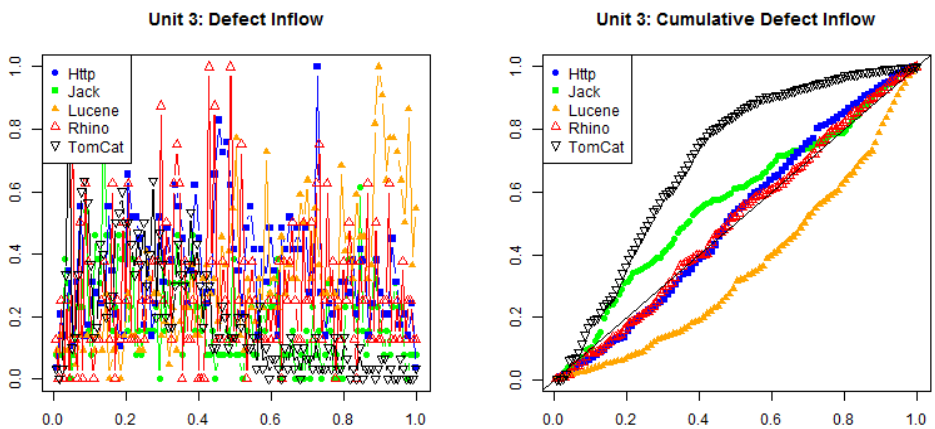

Figure 46: Defect inflow and cumulative defect inflow (normalized) for case unit 3

For projects from the open source community, the issues inflow is quite different across the projects ranging from S-shaped to inflow quite close to following a linear trend. Project-TomCat and Jack show characteristic signs of concave-shaped cumulative defect inflow, while Project-Lucene is convex-shaped. Projects Http and Rhino cumulative defect inflows do not show distinct signs of S-shape, concave- or convex-shape but are more close to following a linear trend, such profile is seen when a project would receive about similar number of issue throughout the studied period. Since software projects in open source community can be released very early in beta releases following development and gaining maturity over long periods to time, issues inflow with linear trend is not unlikely.

3.21.2 Distribution parameters

The distribution parameters for each distribution for all four projects are presented in Table 22 to Table 24.

Table 22: Parameter values for projects A1-A4 for fitted distributions

Project	Exponential	Weibull		Beta		Gamma		Logistic		Normal	
	λ	λ	k	α	β	k	θ	μ	s	μ	σ
A1	4.375	0.809	0.207	0.527	2.318	0.683	2.988	0.193	0.125	0.229	0.232
A2	2.838	1.145	0.366	1.155	2.164	1.032	2.927	0.337	0.146	0.352	0.248
A3	3.982	0.620	0.206	1.408	6.589	0.462	1.839	0.229	0.127	0.251	0.225
A4	8.080	0.446	0.069	0.303	3.617	0.322	2.606	0.089	0.080	0.124	0.176

Table 23: Parameter values for releases B1-B5 for fitted distributions

Release	Exponential	Weibull		Beta		Gamma		Logistic		Normal	
	λ	λ	k	α	β	k	θ	μ	s	μ	σ
B1	5.531	0.807	0.167	0.577	5.342	0.640	3.540	0.150	0.088	0.181	0.176
B2	2.314	2.059	0.490	0.610	2.395	4.200	9.720	0.400	0.120	0.432	0.225
B3	2.188	2.030	0.518	0.953	1.762	3.389	7.416	0.440	0.142	0.457	0.240
B4	4.848	0.954	0.203	1.333	33.081	0.773	3.748	0.186	0.084	0.206	0.167
B5	2.272	2.022	0.498	1.484	3.406	3.348	7.606	0.424	0.132	0.440	0.231

Table 24: Parameter values for open source software projects for fitted distributions

Project	Exponential	Weibull		Beta		Gamma		Logistic		Normal	
	λ	λ	k	α	β	k	θ	μ	s	μ	σ
Http	13.306	0.756	0.069	0.293	2.162	0.569	7.566	0.069	0.033	0.075	0.061
Jack	2.942	2.064	0.384	4.657	84.615	3.527	10.376	0.327	0.097	0.340	0.173
Lucene	5.149	1.343	0.209	0.917	3.092	2.441	-0.015	0.179	0.073	0.194	0.132
Rhino	4.023	0.714	0.224	1.020	4.925	0.527	2.119	0.225	0.106	0.249	0.197
TomCat	4.501	0.593	0.174	0.946	6.219	0.441	1.984	0.192	0.122	0.222	0.221

In Figure 44 to Figure 46, we observed that the defect inflow distributions for different projects across and within case units differ from each other. Following this we can observe from Table 22 - Table 24, that parameter values are different

distribution families tested also span large range. The distribution parameter values not only are different for defect inflows across case units, but also differ for project in case units. For example the shape (λ) parameter of exponential distribution ranges from approx. 2.8 to 8.1 for case unit 1, 2.2 to 5.5 for case unit 2, while the shape parameter for open source project Http is 13.3.

Large difference in parameter values within same distribution family and case units indicate the individual differences between each project. While it is possible that defect inflow form a given company or open source community may follow a particular pattern or distribution, but individual projects also have variations between them. The variations in parameter values further suggest that, while it may be useful to start with the defect distribution information from historical projects; for on-going projects the partial observed defect data should be used (for e.g. using Bayesian approach) to get better forecasts of defect inflow in projects under development/testing.

To visually inspect how the fitted distribution fits the observed defect/bug inflow data, we plot the density and empirical distribution function (eCDF) for observed data and fitted distributions. Figure 47 presents the probability density plots of six evaluated distributions fit to observed defect data for project A1, release B1 and project Lucene. Probability density plots describes the relative likelihood of given data to take a particular value; plotting density plots for observed data and fitted distribution help us visualize the degree of fit over the range of observed data.

Given the observed data, density probability plots is a graphical method to evaluate how well an empirically derived density function fits a theoretical density function for a specified probability distribution [148]. In Figure 47 green (line) represent the probability density of observed defect data while red (dots) show the density function of fitted distribution.

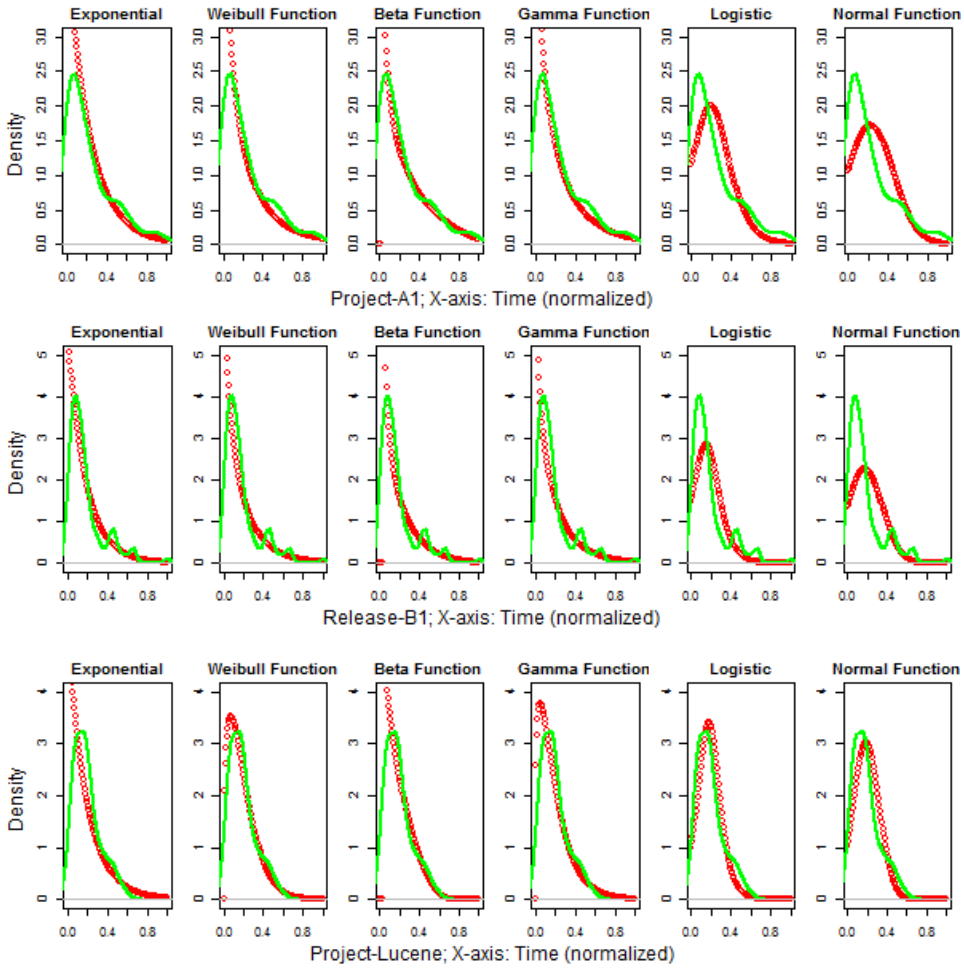

Figure 47: Density probability plots showing Project-A1, Release-B1 and Project-Http data fits to selected distribution (green line show observed data while red dots show fitted distribution)

From Figure 47, following observations are made:

- For Project A1, while the exponential, Weibull and gamma functions gives a good fit over most part, the relative probability density over the left tail fits poorly. The logistic and normal distributions fit is overall poor with underestimation of relative probability over the left tail, while overestimation in right tail. Overall beta distribution seems to fit best over the entire range.
- In Release-B1 again the logistic and normal distributions do not provide good fit. Visually exponential, Weibull and beta are seen to provide good fit to the observe data.

- Visually identifying a best fit distribution based on probability density plot for open source project Lucene is much more difficult with most tested distributions having close fit with observed data. In such cases quantitative criteria of selecting best fit must be used which are discussed later in the study.

Figure 48 presents the density and eCDF plots of six evaluated distributions fit to observed defect data for project A1, release B1 and project Lucene.

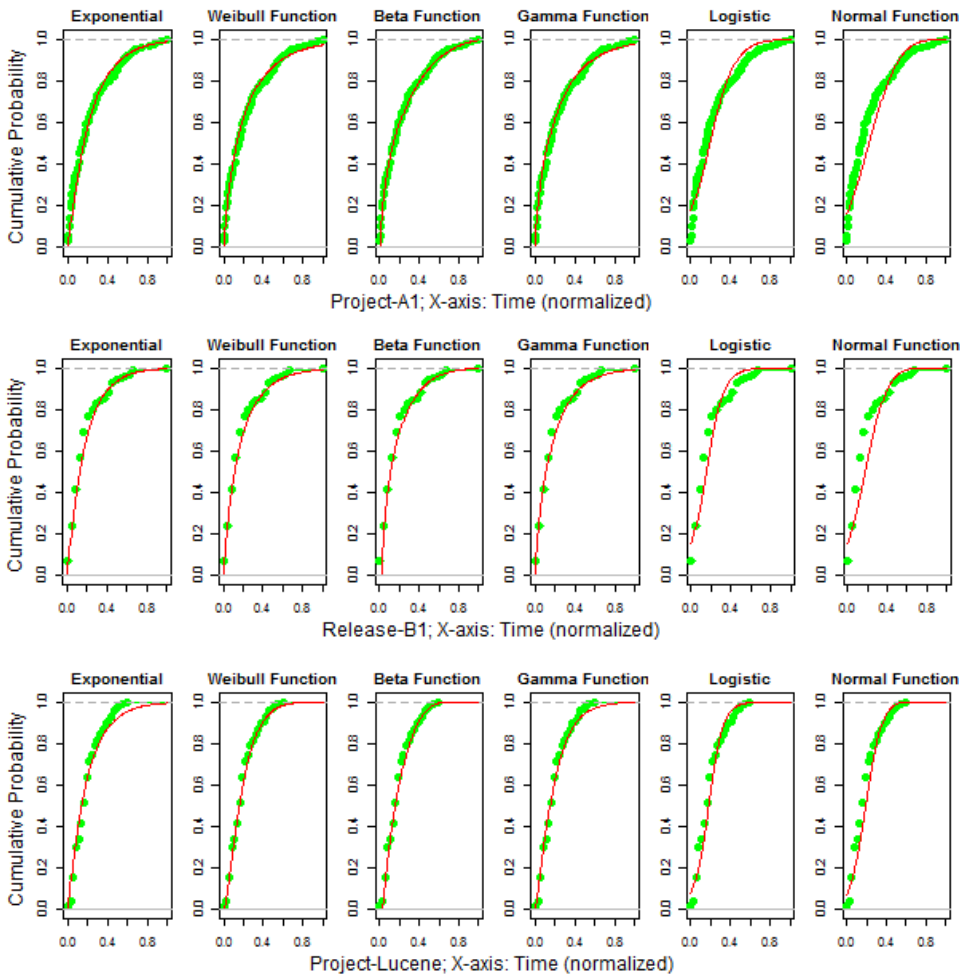

Figure 48: Empirical distribution function (eCDF) for observed data and fitted distributions (green dots show eCDF of observed data while red line show fitted distribution)

Similar to probability density plots, empirical distribution function plots (Figure 48) help visualize the cumulative probability of observed defect inflow and that of fitted distribution. While these plots are also a quick way of visualizing information and can aid in figuring out which distribution family seem to provide good/bad fit for observed data, the plots cannot give precise selection of best fit distribution. The plots can be helpful to visualize the original data as well as eliminate distributions which provide bad fit, for e.g. logistic and normal distribution in case of Project A1 and Release B1 are do not seem promising for further investigation.

3.21.3 Selecting the distribution with best fit

Another popular graphical tool to evaluate the closeness/goodness of fit between observed and specified distribution are the Quantile–Quantile or QQ plots. A quantile is the fraction of points below a threshold, at 0.3 quantile, 30% of the data points fall below the threshold and 70% fall above. QQ-plots compare the fit between two distributions by plotting their quantiles against one another. Although these plots do not allow for visual comparison of probability densities, the fit between distributions is relatively easy to visualize. The QQ-plot for the sample project/release for selected distributions is shown in Figure 49, the $y = x$, green (line) help visualize the fit, the closer the points to the line the better is the fit. The observations from the density probability and eCDF plots are further confirmed for these plots.

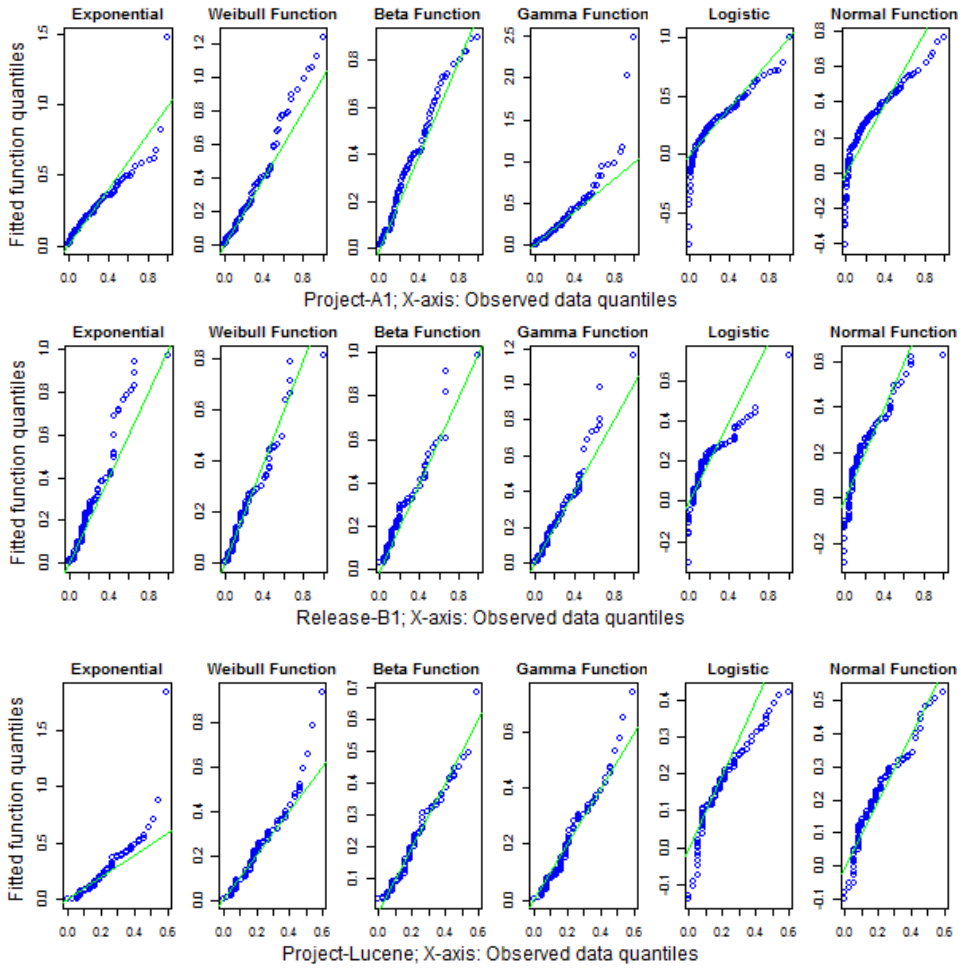

Figure 49: Quantile–quantile plots (QQ-plots) for Project-A1, Release-B1 and Project-Http for selected distributions

QQ-plots are comparatively better graphical method than density plots to assess the fit between observed data and fitted distribution as they are primarily used for this purpose. From these plots (Figure 49) we can make better distinction between which distributions fit the observed data than using probability density plots or the eCDFs (Figure 47, Figure 48). We observe that:

- For Project-A1 from case unit 1, Weibull and beta distributions seem to provide the best fit and arguably it is also clear that beta distribution fit is superior to that of Weibull.

- In case of Release-B1, picking the best fit distribution is not clear from the QQ-plots. Although it seems again the Weibull and beta distributions provide close fit followed by the gamma distribution.
- For Project-Lucene, all but beta and gamma distributions can be seen clearly not providing good fit to observed data.

The graphical methods such as QQ-plots and density probability plots allow us to make a good guess on the fit between the observed data and empirical (fitted) distribution. They also allow for visualizing other characteristics of data such as density probabilities, but it is difficult to pick the absolute best fit function among the tested distributions.

To pick the best fit distribution quantitative assessment can be used. As noted in the section 3, a number of criteria have been proposed to select the best distribution among number of tested distributions. Table 25 show the values obtained for different information criteria for Project-Jack from the case unit 3. In this case irrespective of the information criteria the distribution with best fit was found to be beta distribution.

Table 25: Values of different information criterions for selected distribution for Project-Jack

Project	Distribution	LogLik	ML	AIC	AICc	BIC	HQC
Jack	Exponential	7.29	-14.57	-12.57	-12.53	-10.05	-11.56
	Weibull	36.25	-72.50	-68.50	-68.36	-63.45	-66.46
	Beta	36.72	-73.44	-69.44	-69.31	-64.40	-67.41
	Gamma	36.05	-72.10	-68.10	-67.96	-63.06	-66.06
	Logistic	31.43	-62.86	-58.86	-58.72	-53.81	-56.82
	Normal	30.79	-61.58	-57.58	-57.44	-52.53	-55.54
Selected Criteria		36.72	-73.44	-69.44	-69.31	-64.40	-67.41
Selected Distribution		Beta	Beta	Beta	Beta	Beta	Beta

When fitting distributions to defect inflow data from software projects, the sample size (n) can be different. Depending on the chosen granularity of time and software project's development and testing time span, n could vary from small $(n < 20)$ to very high. In the projects we evaluated, we have weekly count data for industrial projects and monthly count data for open source projects. This together with long time span of these large projects (refer Table 19) give sample size that is large for each project while the tested distributions all have low number of parameters $(k = 1, 2)$, thus for these cases size of penalty do not affect much in model

selection. The penalized likelihood criteria are also not of much use when the choice is between distributions/models with same number of parameters ($k = 2$ for five out of six tested distributions) and for a given project (same sample size). Thus we observe that to select among distributions fit for defect inflow profiles, unless the sample size is small or choice is among models with different number of parameters, most penalized-likelihood criteria for model selection will give similar results as log-likelihood criteria.

Thus while we tested all projects and releases for selected information criteria to select the best fit model we present only the log-likelihood values for all projects for each distribution in Table 26 to Table 28.

Table 26: Log-Likelihood values for selected distribution for case unit 1

Project	Exponential	Weibull	Beta	Gamma	Logistic	Normal
A1	59.0	63.6	105.9	66.0	8.4	5.2
A2	5.5	7.0	19.8	5.5	-6.8	-3.1
A3	56.9	82.9	104.6	98.6	11.6	10.9
A4	119.8	188.3	491.2	199.5	50.3	35.3

Table 27: Log-Likelihood values for selected distribution for case unit 2

Release	Exponential	Weibull	Beta	Gamma	Logistic	Normal
B1	88.1	92.6	167.1	97.9	48.8	39.6
B2	-4.3	4.6	24.8	6.0	2.7	2.0
B3	-7.8	2.8	5.0	2.9	-0.7	0.3
B4	38.8	38.9	86.7	40.4	30.2	24.7
B5	-8.4	5.4	11.9	5.3	1.7	2.2

Table 28: Log-Likelihood values for selected distribution for case unit 3

Project	Exponential	Weibull	Beta	Gamma	Logistic	Normal
Http	195.4	202.1	406.2	211.9	174.6	169.0
Jack	7.3	36.2	36.7	36.0	31.4	30.8
Lucene	62.0	67.6	70.7	71.9	59.4	58.8
Rhino	52.9	63.7	59.4	77.3	31.4	28.1
TomCat	58.5	73.0	76.9	71.9	12.3	10.7

The results from testing fit between selected distributions for their fit to the studied large software projects – we find:

- For all project from two different industrial domains (four large projects form automotive domain and five consecutive releases of large telecom software product), beta distribution was found to fit best defect inflow data

- For open source software projects, beta distribution again provided the best fit in three out of five projects, while gamma distribution was found to fit best to the rest two projects.

The fit between the observed defect inflow data and their respective best fit distributions for all projects/releases is visualized using QQ-plots in Figure 50.

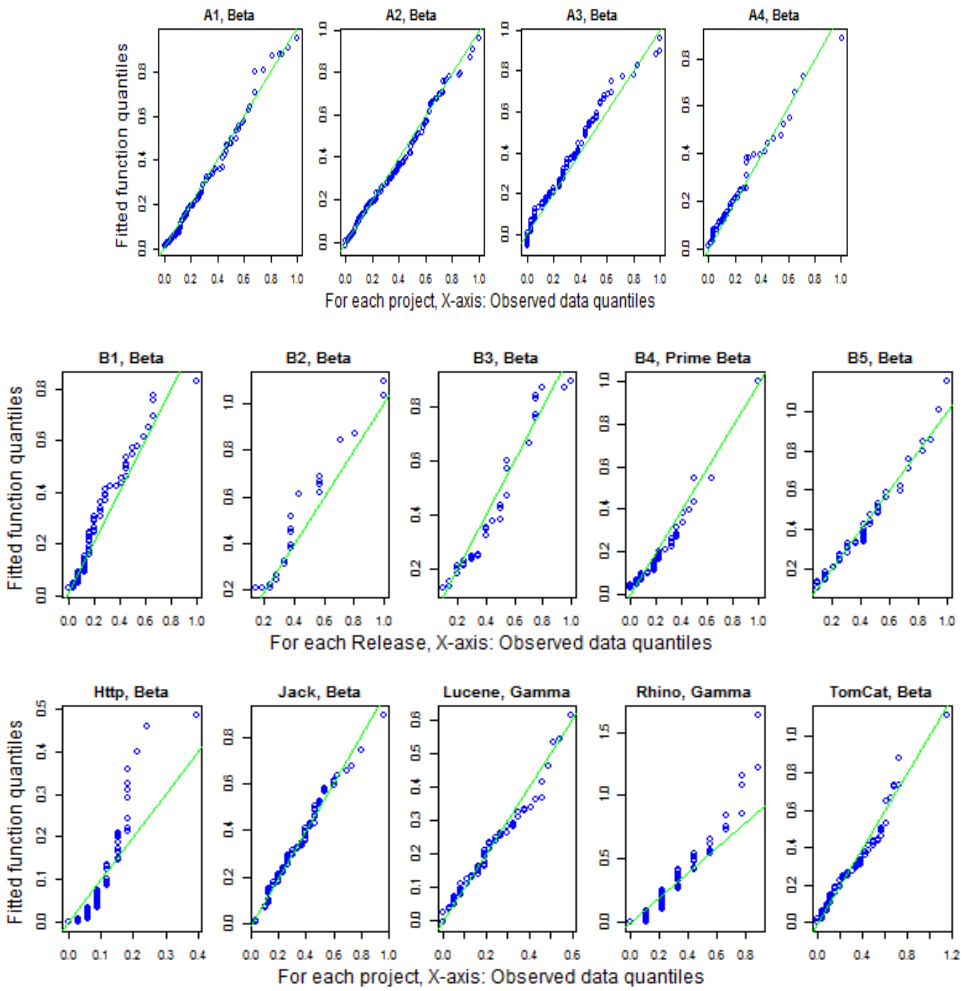

Figure 50: Quantile–Quantile plots (QQ-plots) for all projects/releases and respective best fit distribution

In total 12 out of 14 projects/releases defect inflow data was described best by beta distribution from the selected six distribution families, only for two open source software projects gamma distribution provided the best fit to bug inflow data.

Knowledge of underlying distribution of observed defect inflow data from historical projects is helpful for:

a. In choosing the correct statistical methods for data analysis. If the defect inflow data is not normally distributed, non-parametric methods must be preferred for statistical analysis.
b. Visualization, simulations and scenario analysis. For example a new project is launched where based on project attributes like size, time span etc. the total number of defects is estimated to be n, knowing that majority of historical projects defect inflow followed beta distribution, one could simulate the expected defect inflow intensity over the project development/testing time span. This information can be used for planning the test resources effectively.
c. Knowing the distribution also helps with selecting appropriate SRGM for modelling software reliability. For example if it has been established that defect inflow in most historical projects at a given company had Weibull or Beta distribution, SRGMs with mean value function based on these distributions can be used for modelling software reliability with higher confidence than selecting a model ad-hoc.
d. The information on underlying distribution of data is particularly useful for Bayesian analysis. The distribution that fits best to historical projects can be used to code the initial knowledge as prior probability (distribution). The observations from on-going project are then used to obtain the posterior distribution using Bayes statistics.

3.21.4 Threats to validity

We address the threats to validity in manner as described by Wohlin et al. [37]. Threat to internal validity exists for this study regarding what is considered to be a defect. To minimize the threat a common definition of defect was used which was verified for each industrial project before the data was collected. What is marked as defect is usually not strictly controlled in the open source software projects, the threat of misclassification in these projects was minimized by using the manually validated bug reports from earlier work of Kim, Just and Zeller [143] which helped ensure the quality of data used was high for OSS projects as well.

Another threat to internal validity is concerned with the selection criteria for the best distribution. Best fit distribution was selected using criteria recommended in literature, a number of criteria have been proposed and used to select the best fit. In

this paper we evaluated different information criteria to ensure higher validity of selected distribution which minimizes the threat to internal validity.

A threat to conclusion validity is present because of using a limited set of distributions and selection criteria's, the best distribution selected is valid only among the tested distributions using the selection criteria's evaluated in the study. The selected distributions include most commonly used distributions applied in reliability engineering. With regard to selection criteria for the selecting the best fit distribution, as proposed in [147], multiple criteria's were used to arrive on the conclusions which strengthens the validity of conclusions.

External validity concerns with the generalizability of results in settings outside of the particular study. In our earlier work [123], the analysis was limited to software projects form the automotive domain. In the work presented here, we extended that analysis with five more large software releases from another large company engaged in software development in different industrial domain. We also analysed five projects from the open source community to include software developed with different development paradigms. The defect inflow profiles for studied projects/releases varied widely which indicates towards better generalizability of results. We do not claim that the distribution found best in this study will be applicable for all defect inflow data for any software project but provide empirical evidence from sample of software projects form different domains. The results obtained in this study indicate towards possibility of common distribution of defect inflow for software projects, which is useful for the given company or open source community.

3.22 Conclusions

Six standard distribution families were evaluated for their fit to defect inflow of fourteen projects from the industrial and open source domain. We set out to:

- *Explore which statistical distribution fit best to the defect inflow from large software projects, and*
- *Explore how different information criteria differ in selection of best distribution fit.*

For the projects analysed in this study, beta distribution fitted best to the defect inflow data from the industrial software projects. Defect inflow of three of five OSS projects also has beta distribution while remaining two projects defect inflow followed gamma distribution. To select the best fit distribution to defect inflow, out evaluation suggest that it can be done with high confidence, using one of many recommended likelihood based criteria. If the number of observations is large and

comparison is among distributions with about same number of parameters, penalized-likelihood criteria are unlikely to give a different result than choosing model with highest log-likelihood.

Knowing the underlying distribution of defect inflow, helps with understanding and explaining the process of defect discovery in a given software development and testing environment. The information is further useful for selecting right statistical methods and techniques for data analysis and also to choose appropriate models for defect/reliability modelling and predictions.

Knowledge of underlying disturbing of defect inflow also allow for easy tracking, visualization and simulation of data which is useful for scenario based analysis. Underlying distribution information is especially useful to model experience or personal belief as prior probability distribution in Bayesian statistical analysis.

Chapter 4:
Consequence of mispredictions

Included Publication:

VI. M. Staron, R. Rana, W. Meding, and M. Nilsson, ***"Consequences of Mispredictions of Software Reliability: A Model and its Industrial Evaluation"***, In the proceedings of 24nd International Conference on Software Measurement, IWSM-Mensura, Rotterdam, The Netherlands, 2014

4 CONSEQUENCES OF MISPREDICTIONS OF SOFTWARE RELIABILITY: A MODEL AND ITS INDUSTRIAL EVALUATION

Abstract— Predicting reliability of software under development is an important part of estimations in software engineering projects. In many organizations as the goal is that software products are released with no known defects, the process of finding and removing defects correlates with the effort for software projects. Software development projects estimate the resources needed to design, develop, test and release software products, and the number of defects which have to be handled. In this paper we present a model for consequence analysis of inaccurate predictions of quality in software projects. The model is a result of multiple case studies and is evaluated at two companies. The model recognizes the most common mispredictions – e.g. over- and under-prediction, early- and late-predictions – and the combination of theses. The results from the industrial evaluation show that the consequences can be grouped according to under- and over-predictions and that the late- and early-predictions have the same consequences. The results show also that mispredicting the shape of the reliability curve has a significant consequence with regard to assessment of release readiness and resource planning.

Keywords— Software Reliability; SRGMs; Consequence; Mispredictions; Software; Forecasting

4.1 Introduction

Predicting the number of defects in software modules is one of the tasks of quality managers in mature software development organizations. The quality managers predict rate of defect inflow in order to support the organization in directing software testing or optimizing testing efforts [149], [150]. One of the methods for predicting is using software reliability growth modelling [151]–[153] which is based on developing a reliability growth formula[6] and applying it on an on-going project in order to predict the future defect inflow and the total number of defects [154].

The development of the reliability formula requires domain and product knowledge and can be costly in terms of data collection. The formula may also be inaccurate as software development projects are dynamic entities exposed to external factors (e.g. sick leaves, equipment failures, project delays). One of the aspects which make the reliability growth formulas being neglected by software professionals is the lack of a cost-model which could allow the professionals to reason about the potential costs of mispredictions.

Based on our previous work in reliability modelling we observed this as an important problem which we address in this paper [15], [77], [155], [156]. In particular, in this paper we address the following research question:

Given the software quality growth prediction curve, what are the consequences of mispredicting the total number of defects and release readiness?

In the research question we explicitly recognize two common axes of the accuracy of predictions – (i) the prediction of the asymptote or the total number of defects and (ii) the prediction when the total number of defects is discovered or the release readiness [157].

In order to address the research question we conducted an action research project where we develop the model together with the industry professionals from Ericsson and Volvo Car Group. The model shows that the costs of over and under-predictions as well as predictions using an incorrect type of curve can cause significant extra costs for the companies in terms of unnecessary extra testing effort or costly post-release corrective maintenance.

[6] The usual name is *Software Reliability Growth Model*, but we use the term "formula" in order to avoid mixing it with the consequence model presented in this paper.

4.2 Related work

Reliability growth formulas
Common terms related to software reliability are defined in IEEE 1633: Recommended practice on software reliability [78]:

Software Reliability (SR): is *(A) the probability that software will not cause the failure of a system for a specified time under specified conditions. Or (B) the ability of a program to perform a required function under stated conditions for a stated period of time.*

Software Reliability Model (SRM) is *a mathematical expression that specifies the general form of the software failure process as a function of factors such as fault introduction, fault removal, and the operational environment.*

IEEE standard 1633 also provides metrics used in reliability modelling and specifies recommended procedure for software reliability assessment and prediction. SRMs can be classified as white box and black box formulas. White box formulas use source code attributes for making assessment and predicting defect proneness of given software artefact, while black box models uses defect inflow data for modelling reliability. Based on the nature of data used, white box and black box models are also known as static and dynamic formulas. Dynamic/Black box formulas are usually referred to as SRGMs and uses defect data from development and/or testing phases. The failure or reliability process can be modelled using calendar or execution time. Though execution time models have been shown to be more accurate, calendar time models provides more intuitiveness for testers and mangers making them easy to interpret.

SRGMs can be applied primarily for two purposes, the firstly for optimal allocation of test resources [97] and secondly for release readiness assessment [95]. In cases where SRGMs are used for optimal resource allocation such as amount of testing time or test case allocation - SRGMs are applied during the testing process on the partial defect inflow data. Then the predicted/estimated defect inflow information is used to allocate the testing resources optimally such that the product is ready for release by the release date. Release readiness is assessed by applying SRGMs on the defect inflow data post the testing phase, suitable SRGM based on a given testing process is used to model the defect inflow and estimate the total number of defects. If the prediction is close to number of defects already detected and fixed the software is assessed ready for release, while if the SRGM show presence of latent defects higher than the required quality criteria then the software is assessed as not ready for release and send back for further testing.

Given the nature of applications, it is apparent that models that provide superior fit to full defect data are better suited for applying SRGMs for release readiness, while models that have better long-term predictive power are more useful for resource allocation applications. Table 14 presents a set of commonly used software reliability growth models.

Table 29: Software reliability growth formulas used in this study

Formula Name	**Shape**	**Mean Value Function**	**Ref.**
Musa-Okumoto	Concave	$m(t) = a\ln(1 + bt)$	[113]
Goel-Okumoto	Concave	$m(t) = a\,(1 - e^{-bt})$	[79]
Inflection-S model	S-shaped	$m(t) = \frac{a\,(1 - e^{-bt})}{(1 + \beta e^{-bt})}$	[7]
Delayed-S model	S-shaped	$m(t) = a\,(1 - (1 + bt)e^{-bt})$	[80]
Rayleigh model	S-shaped	$m(t) = a\,(1 - e^{-\left(\frac{t}{b}\right)^2})$	[69]
Logistic model	S-shaped	$m(t) = \frac{a}{(1 + e^{-b(t-c)})}$	[158]
Gompertz model	S-shaped	$m(t) = a\,e^{-be^{-ct}}$	[91]
Linear model	Linear	$m(t) = g * t + c$	[112]
Release readiness (RR)	Linear	$RR = \frac{\#defects}{drr-(trr-tpr)}$§	[157]

§ where drr = defect removal rate; trr = test execution rate and tpr = test pass rate

In this study we categorize models according to their shape – Convex, S-shaped, Concave and linear.

4.3 Mispredicion consequence model

The model for assessing the consequences of mispredictions contains the following elements:

- Shape of the prediction formula
- Set of consequences
- Strategies to minimize the risk of mispredictions

The consequences of mispredictions are usually negative for the project, the company or the product under development. In the model we recognize three situations:

- Mispredictions of the asymptote – mispredictions of the total number of defects

- Mispredictions of the release readiness – mispredictions of when all predicted defects are found
- Mispredictions of the shape of the prediction formula

As we show in the model below there are numerous similarities in the consequences for these three types of situations.

4.3.1 Mispredicting the asymptote

Predicting the asymptote of the number of defects shows the total number of defects which the product might have. The asymptote is usually the same as the coefficient *a* in the formulas in Table 14. Figure 51 shows two types of mispredictions –over and under- predictions. The *optimal* prediction is the prediction which is the closest to the reality[7]. The mispredictions of the asymptote are important for the ongoing project, which means that the consequences regard the fact that the predictions are over- or under- the optimal curve during development.

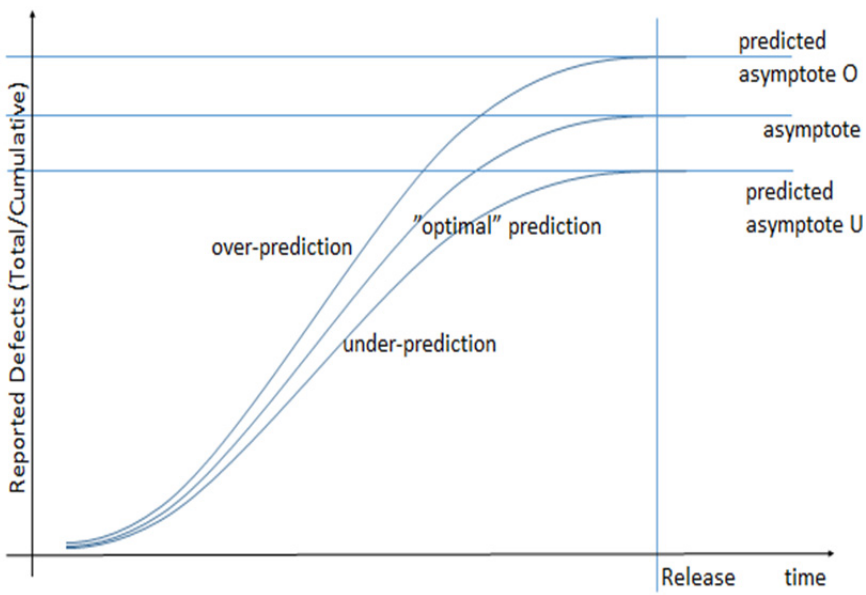

Figure 51: Mispredictions of asymptote

For the *over-predictions* the general consequence is that the project expects to find more defects than they do. This means that during the duration of the project the

[7] In theory this is the optimal prediction is equal to the actual defect inflow in the project, but in practice there can be deviations.

project management can perceive their testing process as inadequate or ineffective thus putting more effort into finding defects which do not exist. More detailed examples are:

- Too high expectations on the defect inflow – more pressure on the testing team to find more defects (where there might be none).
- Assumption that testing is ineffective as during the project (before the release) the number of predicted defects is higher than the number of discovered defects.
- Additional cost of test analyses in search for new test areas (unnecessary)
- Risk for postponing release
- Risk for lost time to market
- Risk for wasted costs for testing
- Risk for unnecessary RCAs to find area which are not tested enough

Over-predictions occur when testing is done early in the project and when the majority of the defects are found early (and expected late). This situation is common when the wrong shape of the curve is using for predictions – e.g. S-shaped instead of Convex.

For the *under-predictions* the general repercussion is that the project expects too few defects and thus risks releasing the software to the customers with defects. Consequences of under-prediction:

- Releasing the product with defects (since effective testing is not really effective).
- Additional costs for post-release defect removal activities and patches.
- Defects which are manifested as integration problems requiring quick fixes.
- De-prioritizing testing effort at early stages and thus finding large number of late (and thus costly) defects during system testing or acceptance testing.

Under-predictions occur when testing is done late in the project or when the product has the functionality that requires full integration (e.g. complex functionality or large embedded products). The situation is common when Concave or S-shaped predictions are used instead of Convex-predictions.

4.3.2 Mispredicting release readiness

Another dimension of the model is the timelineness of predicting [159] when the reliability growth curve reaches the asymptote. There are two cases of mispredictions – the early-prediction and late-prediction. Figure 52 illustrates these two cases compared to the optimal/true prediction.

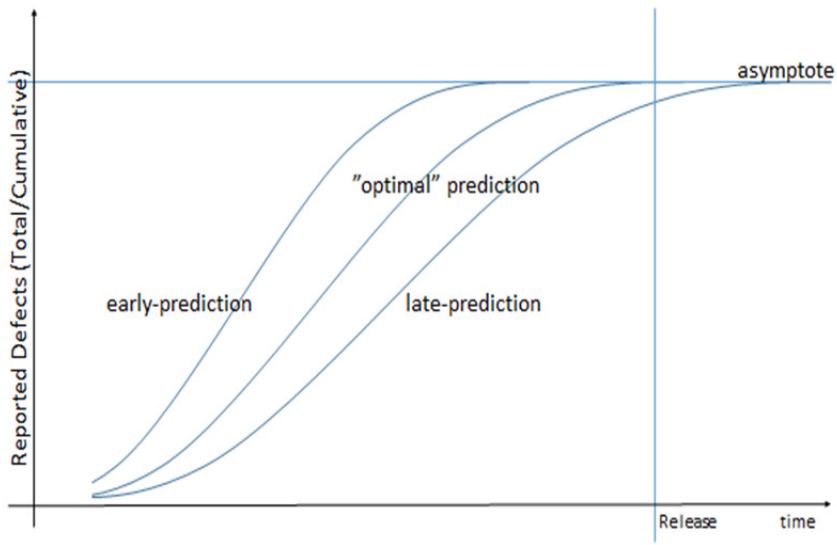

Figure 52: Mispredictions of release readiness

For the *early-prediction* the general repercussion is that the project management is informed about being ready earlier than in reality. This means that the consequences can be:

- Releasing the software with defects
- Higher cost of corrective maintenance of the product
- Postponing the release (if the mispredictions are discovered before the release)

For the *late-prediction* the general repercussion is that the project management received information about being late when in fact being on track. This means that the consequences can be:

- Unnecessary additional testing resources to get back on track
- Postponing the release in expectation of more defects to come and in order to avoid costly corrective maintenance

- Additional costs of test analysis to increase the speed and effectiveness of testing

4.3.3 Mispredicting the asymptote and the release readiness

The superposition of the mispredictions amplifies the situation and increases the risks for negative impact on the project. However, it does not introduce new risks. Figure 53 presents the superposition.

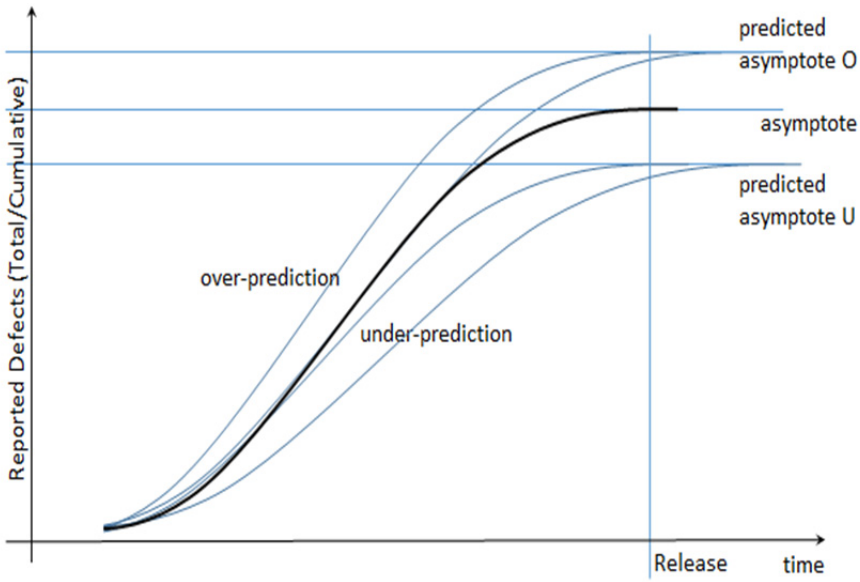

Figure 53: Mispredictions of release readiness and asymptote

As the figure suggests the superposition of the mispredictions is similar to over- and under-predictions. The difference, however, is that in the beginning of the project the under-predictions of the asymptote may be perceived as over-predictions. This situation depends on how early the under-predictions predict the asymptote. The more inaccurate mispredictions are, the larger the chance of misperceptions.

4.3.4 Misprediction of the shape of the curve

One of the main issues in using the reliability growth models is the choice of the reliability growth formula, which determines the type of the curve as shown in Figure 54.

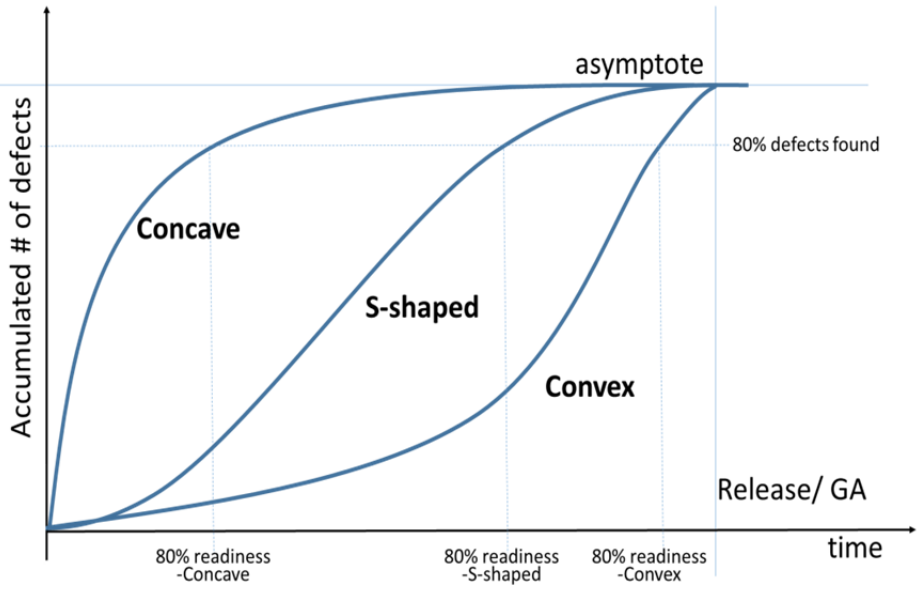

Figure 54: Mispredictions of the shape

Mispredicting the shape with the consequences is described in the following table.

Table 30: Consequences of mispredicting the shape

Actual shape	**Expected shape**		
	Convex	**S-shaped**	**Concave**
Convex		Over-prediction of the total number of defects	Over-prediction of the total number of defects
S-shaped	Release readiness is predicted too early X% of found defects is predicted earlier than expected		Over-prediction of the total number of defects
Concave	Release readiness is predicted too early X% of found defects is predicted earlier than expected	Too much resources for late testing	

The consequences of the mispredictions of the shape are visible in the course of the project as the decisions of project management are based on false trends of defect inflows.

The mispredictions can have significance impact on decisions in the following situations:

- Predicting when the project discovers X% of expected defects. As shown in Figure 54 this mispredictions may be significant if the Concave model is used instead of the other two.
- Predicting the status of the project w r t target. If a project is done in a continuous way (e.g. Lean/Agile [16], [157]) then the shapes like Convex or Concave can lead to wrong status reporting (too many or too few defects than expected)

The situations and consequences described in this section constitute the most common problems with mispredictions with the most serious consequences.

4.4 Industrial evaluation

4.4.1 Case study design

Following the taxonomy and guidelines for conducting and reporting case studies in software engineering by Runeson and Höst [52], we conducted an exploratory case study using flexible design principle. We studied two large companies from widely different industrial domains (Automotive and Telecom) with significant focus on development of embedded software. Given the differences in domain, the study is designed as an embedded case study with two units of analysis (each company); Figure 55 and Table 31 present an overview of the case study design and summary of case units.

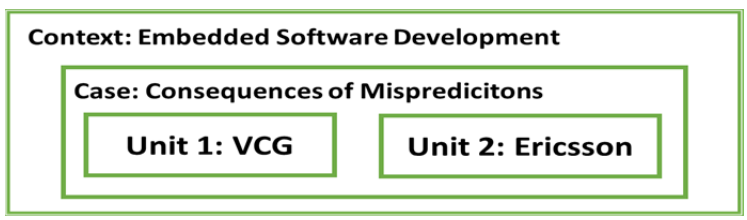

Figure 55: case study design

Table 31: Overview of case units.

Application domain	**Software development process**	**Current methods for software defect prediction**
Automotive	V-shaped software development mostly using sub-suppliers for implementation	Focus on status visualization and analogy based prediction
Telecom	Agile development, mostly in-house	Various modes of presenting current status and predictions methods

To provide the context for this study we provide details about each case unit's domain, important characteristics of their software development and which specific

part of organization we interacted with. The included information provides the context that is needed for meeting the objectives of this research.

Volvo Car Group (VCG): A company from the automotive domain

The team we interacted with in this case study from VCG is responsible for integrating software for electrical systems at complete vehicle level. While some of the software is developed in-house using agile process, majority of the embedded software development in the company is developed following V-model through external suppliers who design, implement and test the functionality based on specifications provided by the VCG. The company on the other hand is responsible for high level functional development which is done in domain specific modelling language such as Matlab/Simulink[8].

Ericsson: A company from the telecom domain

Ericsson develops large software products for the mobile telecommunication networks. Projects are carried out according to the principles of agile and lean software development. In this environment, various teams are responsible for larger parts of the process compared to traditional processes: design teams (cross-functional teams responsible for complete analysis, design, implementation, and testing of particular features of the product), network verification and integration testing, etc. The whole process is dominated with continuous development and testing as expected in highly iterative agile software development process.

4.4.2 Data collection and analysis methods

The main source of data for the case study is obtained through empirical observations and semi-structured interviews, data collected through interviews is a form of first degree methods [52], that are although expensive to collect but offer larger control. Since the objective for this research was to explore, identify and validate consequences of mispredictions of defect inflow on software quality, direct method in form of interviews was assessed as appropriate.

Stronger conclusions can be drawn by using triangulation i.e. using data from several sources [52], therefore we complement the information obtained through interviews with document analysis from these companies. The archival documents analysed related to the information needs within the organization with respect to software defects and information demanded by various stakeholders within the organization. Semi-structured interviews were conducted with managers responsible

[8] Simulink® is a block diagram environment for multidomain simulation and Model-Based Design. Matlab and Simulink are products and registered trademark of The MathWorks, Inc.

for providing software defects related information to different stakeholders within the organizations and quality manger. The interviewees were:

- Manager at Volvo Cars Group within the department responsible for integrating software sourced from different teams and suppliers, the manager have more than 20 years of experience working with software development and testing. Ensuring safety and quality is a major responsibility within this role.
- Team leader of metrics team at Ericsson; the studied department at Ericsson provides the measurement systems to different stakeholders within the organization. The team leader interviewed also have more than 20 years of experience working with software development and testing.

4.5 Results

4.5.1 Summary of results form case unit A: VCG

- A number of different metrics are collected and monitored continuously for tracking and assessing quality of software under development.
- Forecasts are used to track if the software will be ready for release (with respect to quality) by the release date.
- If forecasts show an area with possible problems, then root cause analysis is done to discover the main causes for such deviations.
- The focus after the root cause analysis is on what can be done now to get on track? As it is highly important to meet the release dates, more resources are mobilized and allocated where needed.
- Consequence of under-predictions: Would need task force (resource mobilization) late in the project. While this is not seen as major problem if under-prediction is limited to a few ECUs, but could be a potential problem if under-prediction is widespread across platform (large project).
- Consequence of over-predictions: Over-prediction is not seen as a critical problem in this case unit. If it is recognized late in the project that forecasts have over-predicted, the human test resources simply shift their energies and focus on other on-going projects.
- Consequence of early-predictions: No impact if the project is small as risk can be easily managed at any stage of project. For larger platform projects, in case of early-predictions, the forecasts will be re-checked consecutively for a period of time and cross-validated

by different expert opinions before resources are planned according to forecasts.
- Consequence of late-predictions: In this case, the strategy adopted within this case unit is to find areas affected by late-predictions. The test resource would be balanced in light of new information and with the aim to meet quality requirements by the release date.

4.5.2 Summary of results form case unit B: Ericsson

The impact of mispredictions have two dimensions – (i) metric team which delivers the predictions and (ii) project where the predictions are used.

For the metrics team:

- All mispredictions make the team lose trust from the organization. Once the organization acts upon wrong predictions the team loses the ability to influence – the next time the organization will need a second opinion before acting. This increases the cost of predictions in the long run.
- For the projects:
- Over-predictions:
- Strengthening and reallocation of resources – if this is done during a long period of time then this impacts the release date negatively
- Under-predictions:
- Negative impact on the release date
- Ordered overtime/extra resources – when the organization finds that the reliability was under-predicted
- Reallocation of resources – when the organization finds that the reliability was under-predicted.

4.6 Interpretation and recommendations

Strategies to avoid mispredictions: In order to avoid the costly mistakes in predictions we have identified a number of strategies:

- **Predict often** – update the predictions every 4th data points (e.g. every fourth week for weekly predictions). Updating too often can cause instability of predictions and the loss of trustworthiness [154], [160] and predicting too seldom causes risks of unnecessary costs during longer periods of time.
- **Experiment with three types of curves** – until the prediction model stabilizes (i.e. the curve can be fit with R2 over 85%)

experiment with concave, s-shaped and convex curves. If the prediction model does not stabilize use the logistic model as it gives the most accurate results in the majority of cases [154].

- **Predict the shape of defect inflow using available data** – in our earlier work [161], we showed that by analysing trend of defect inflow it is possible to predict the shape of defect inflow as early as half-way through the project timeline. Such prediction can be useful to select the right SRGM.

4.7 Conclusions

Quality is an important criterion for software products. To ensure that software developed in a project meets its quality requirements by the release date calls for monitoring and forecasting metrics related to software quality. Predicting expected defect inflow, total expected defects and latent defects offer one way of monitoring and forecasting software quality. Such predictions are also important to plan and balance test resources which form major part of software costs. While such predictions help monitor and plan for test resources, their use is associated with risks of mispredictions.

Mispredictions if not handled carefully can have major impact on project timeline and costs. In this paper we provided a consequence model for most common mispredictions. The model helps evaluate what could go wrong and the consequences of the same. The model is validated at two companies which provide insights into which mispredictions are critical in industry and how they are currently managed.

Chapter 5:
Correlation based software defect prediction technique in automotive domain - evaluation

Included Publication:

VII. R. Rana, M. Staron, J. Hansson, M. Nilsson, and F. Törner, ***"Predicting Pre-Release Defects and Monitoring Quality in Large Software Development: A Case Study from the Automotive Domain"***, Submitted to a Journal

5 PREDICTING PRE-RELEASE DEFECTS AND MONITORING QUALITY IN LARGE SOFTWARE DEVELOPMENT: A CASE STUDY FROM THE AUTOMOTIVE DOMAIN

Abstract— Early estimation of software defects in sub-systems and features in an on-going project can be used for effective allocation of effort and resources by the development teams. It also provides support for planning and decisions with regard to software release. Current methods of defect predictions based on code and change metrics requires access to source code and software evolution information which may not always be available or easy to obtain. This paper evaluates if number of defects found in an earlier integration point can be a good predictor of number of defects to be found in next integration point. Using data from four large software projects from the automotive domain, we evaluate the correlation between defects found across integration points and final pre-release defect count. We do the analysis at the granularity of sub-systems (Electronic Control Units) and features with a sample size of 140 sub-systems and 178 features over 9 integration points. Our results show that defects found at integration point 4 and 5 could be used as a good predictor for forecasting total expected defect count at final release. It is also demonstrated how correlation between defects found across integration points can be used to identify risky modules early in the development life cycle which can trigger corrective actions.

Keywords— Software defect management; Defect-prone modules; Predictive models; Software evolution; Automotive domain

5.1 Introduction

Defects in software are real and observable indicators that can be used to track the quality of given product during its development and testing. It has been shown that in large software development projects, majority of software defects are often found in few of the sub-systems and features [162] [163]. Identifying which sub-systems[9] and features[10] are defect-prone and early estimation of number of defects expected to be found in a given project can be effectively used to increase testing efficiency. Such predictions help teams to focus quality assurance activities and resources to areas where they are needed most. Such efforts help in improving the overall quality of software system under development.

Defect prediction models based on code and change metrics requires access to source code which may be a problem when software is developed using sub-suppliers or code is auto-generated from models in model based development. Another shortcoming when using these methods is uncertainty of how to handle cases where software under study contains reused code [28], which is a frequently the case in many industrial domains engaged in embedded software development for example automotive domain.

Simple prediction models that only use defect data from earlier integration points of software project lifecycle to predict defect count in later integration points can elevate many of aforementioned shortcomings. Such models can prove to be simple and cost-effective way of estimating defects at the appropriate granularity level (sub-system or features) and time (early), where such predictions allow quality and project managers to take corrective actions.

In this paper we investigate the relationship between defects discovered during an earlier integration point to that of later ones over a software project lifecycle (*at sub-systems and features level*). The research questions (*RQ1-RQ4*) that we address in this paper are:

- ***RQ1:*** *does small number of modules contain most of the defects found in large automotive software projects?*

[9] We use sub-systems in the given (automotive software) context to refer to software developed for specific Electronic Control Unit (ECU).

[10] Feature refers to a software module that provide a specific functionality for example Anti-lock control module, software module for central locking etc. Features are also commonly referred to as functions in the automotive domain.
There exist many to many relationship between sub-systems and features, while an ECU usually carry more than one feature – a single feature can also be distributed over many ECUs.

- ***RQ2:*** *Do defects found in current integration point strongly correlates to defects found in next integration point?*
- ***RQ3:*** *How can we use defect inflow data for continuous quality monitoring (i.e. early risk identification)?*
- ***RQ4:*** *When in project timeline can we make useful pre-release defect count predictions?*

To answer the research questions posed, we use correlation and simple regression analysis on defect data from four large industrial software projects from the automotive domain consisting of in total 140 sub-systems and 178 features over 9 integration points.

We find evidence which support earlier observations [1] that small number of software modules accounts for majority of defect counts, specifically our results support the 20-60 rule observed by Fenton and Ohlsson [2]. The results from regression analysis show that number of defect found at fourth and fifth integration point can be used as an early indicator for predicting total pre-release defects. We further show that correlation between defects found across integration points can be used to identify sub-systems and features that may need more attention thus helping early interventions to improve their quality.

The remaining of the paper is structured as follows: section 5.2 presents the background and related work for our research with brief overview of different methods of software defect predictions. In section 5.3 we describe in detail the research methodology and data used for this study, while Section 5.4 presents the findings from the study. Section 5.5 provides recommendations for industrial practitioners on how to apply the proposed prediction model and discuss the threats to validity. Finally section 5.6 presents our conclusions and outlines future research directions.

5.2 Background and Related Work

5.2.1 Software Defect Prediction

A software defect can be defined as an issue or deficiency raised due to use of software product which causes it to perform unexpectedly [8]. Software Defect Prediction (SDP) methods are used either to classify which modules are defect-prone or to predict the number of defects expected to be found in a software module/project. Usual techniques used for classification/prediction are using:

- expert opinions for prediction,
- software reliability growth models for prediction,
- regression based methods for classification and prediction, and/or

- machine learning based methods for classification and prediction.

Expert opinion is one of the easiest (if experts are available), but subjective method of predicting software defects in an on-going project. This method can be useful in cases where defects predictions are to be done at project level or large components level and where experts are readily available. But when defect predictions are to be made at lower granularity levels (sub-systems, features, files etc.), this method do not scale down.

Software Reliability Growth Models (SRGMs) use mathematical equations to model the growth of software/system reliability using defect inflow data from the development/testing phases. Appropriate model is selected based on software development/testing process or using empirical evaluations of number of models on the testing data, which is then used to select appropriate SRGM to make defect forecasts. These models are easy to apply in practice, but reliable predictions can only be made when enough testing data is available for model fitting which may be late in the testing process. These models can be used to model reliability growth over testing period or over the software lifecycle using models such as Rayleigh model.

Regression based methods on the other hand uses statistical regression for making defect predictions using a set of software metrics or code change attributes as predictor variables. Regression methods such as logistic regression can be used to classify software modules as defect-prone or not, while multiple linear regression can be used to estimate the number of expected defects in a given software project or modules (sub-systems/features etc.). A range of software process and product metrics have been used as the independent variables in the regression based models; most common among them are the code complexity metrics and source code evolution (change) metrics.

Methods based on machine learning use algorithms based on statistical methods and data mining techniques that can be used for defect classification/predictions. These methods are similar to regression based methods and use similar input data (independent variables). The key difference being that machine learning based methods are dynamic learning algorithms that tend to improve their performance as more data is made available.

5.2.2 Automotive Domain and Embedded Software

Automotive software is essentially embedded software which is defined as the software that resides permanently within a device (hence embedded) and contributes to the device control and functionality. Today's modern cars carry large amount of software; some estimates suggest that premium segment cars can carry up to 100 million lines of code, much more than that of F-35 Joint Strike Fighter (5.7 million)

and even Boeing's 787 jetliner (6.5 million) [54]. A typical modern car can carry about 800 features [4] realized by software which is distributed over 70 to 100 microprocessor-based Electronic Control Units (ECUs) [54].

Automotive software is diverse and complex, the major reasons for complexity can be attributed to factors such as [1]:

- *Interaction between software and hardware with number of sensors and actuators;*
- *Expected real time behaviour based on states and events;*
- *Systems with long life time where embedded software is expected to continue working often without updates; and*
- *Demands for high reliability and dependability especially for applications which are safety critical.*

Software development in automotive domain mainly follows V-model where left branch (early phase) is dominated by software design and implementation, while verification and validation is prominent on the right branch. Figure 2 shows the mapping of different stages/phases in automotive software and electronic hardware (ECU) development at the case company (Volvo Car Group, VCG). Requirements at the vehicle level are grouped based on features (or features), each feature has an assigned owner responsible for overlooking the design-to-acceptance of that feature in the final product. System designers design the system based on all the features that are carried over and to be introduced (new). The system is designed such that each ECU is assigned number of logical components which implements the required functionality. Thus there is one to many relationship between feature and logical components for example to provide an Anti-lock feature/feature, central electronic module (ECU) may have a logical component named Anti-lock control component, while ECU controlling the wheel braking may have another logical component that implements the braking action under anti-lock conditions, which together fulfil the full functionality of Anti-lock braking feature.

It is common in automotive domain that Original equipment manufacturers (OEMs) such as VCG take responsibility of design and acceptance testing of software and hardware at vehicle level, while electronic hardware (ECUs) and base software for the ECUs is developed by their suppliers. While OEMs do implement some of the application level software in-house (generally features/features that are new and innovative which provide market differentiation to their products), but much of the application level software is also sourced through tier-1 and tier-2 suppliers customized to the need of individual OEMs. Under these conditions access to change metrics is not readily available as the software is developed/customized by supplier and not developed in-house.

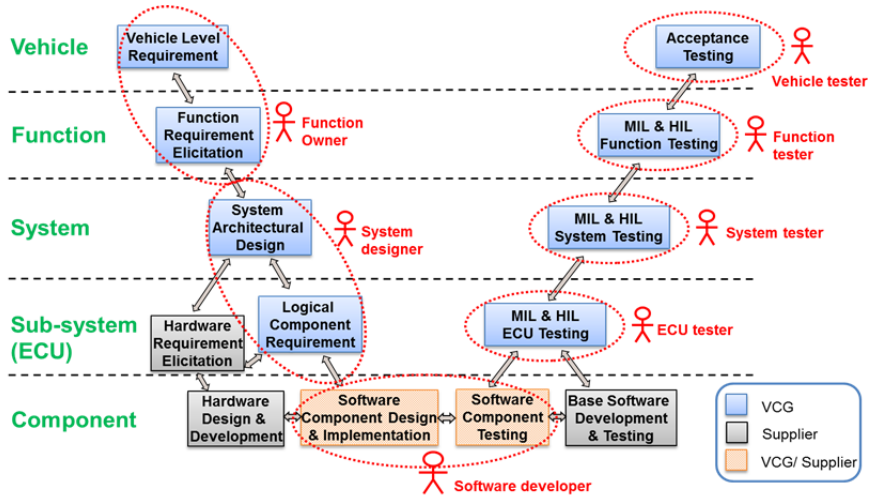

Figure 56: Overview of software development process at VCG

Further software development in the automotive domain often uses combinations of different programming languages and techniques. Use of domain specific languages (DSLs) such as Matlab/Simulink is common among major companies in this sector (both OEMs and their suppliers) and also among other embedded software domains (e.g. aerospace). The production code that runs on a typical ECU today may have mix of code elements that are auto-generated from behavioural models, behavioural model that includes legacy code and hand written code. Figure 3 shows the possible mix of software elements that can be part of production code providing the intended functionality. In such features and systems obtaining precise and accurate complexity metrics possess challenges for e.g.

- Should we use complexity metrics from behavioural models or from code generated from these models?
- Can we reliably use the complexity metrics for code that is auto-generated and optimized using different Model-to-Code generation tools?
- Can we compare or combine complexity metrics from auto-generated and hand-written code?

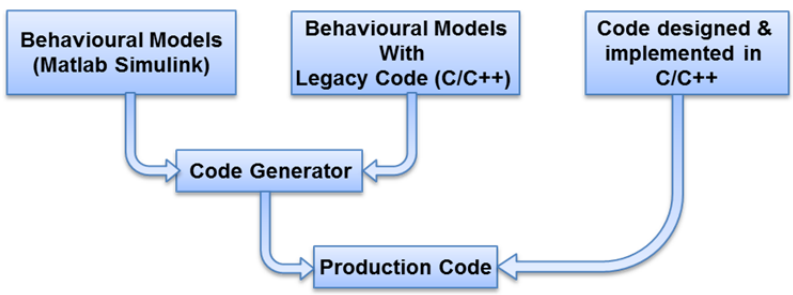

Figure 57: Possible mix of software elements in automotive production software

Thus software defect prediction techniques based on change and complexity metrics may not always be feasible or easy to apply in certain cases within automotive and other embedded software domains due to:-

- Difficulty to calculate the size of software module with good accuracy, thus difficulty calculating and working with defect densities.
- Source code metrics e.g. complexity, size, couplings are readily defined for hand written code, but corresponding metrics for behavioural models are often not validated and in some cases not yet defined.
- Sourcing software from suppliers may also pose difficulty in obtaining reliable software evolution information (i.e. change metrics) during its implementation.

With these issues in perspective, for large iterative software development projects we evaluate the efficacy of defect count at a given integration point as an early predictor for defect count in next integration point and for predicting total pre-release defects. This paper makes following key contributions:

- New defect count prediction technique that does not require code or change metrics.
- The evaluation of usefulness of defect count in given integration point as early predictor for defect count in following integration point and total expected pre-release defect counts.
- Approach for early identification of features and sub-systems that may need more attention.

5.2.3 Related Work

A number of earlier studies have provided empirical support for the Pareto principle of defect distribution over software modules. Fenton and Ohlsson [2] observed 20-60 rule i.e. approximately 20% of software modules accounting for more than 60% of defects discovered during pre-release testing. The observation was also confirmed in the replication study by Andersson and Runeson [164], the principle have also been documented for software systems by other researchers [165] [166]. In this study we provide complementary evidence to support the Pareto principle of defect distribution for four large software projects from the automotive domain and hence strengthen the evidence in this regard.

When it comes to predicting expected number of defects at project or lower levels of granularity – a number of approaches/techniques have been used. Earlier studies have evaluated expert opinions and different SRGMs for predicting software defects. Wood [13] applied eight SRGMs on industrial defect inflow data and found significant correlation between pre-release defects and post-release defects. Staron

and Meding [15] studied defect data from the telecom domain and concluded that models based on moving average provided good predictability for weekly defect predictions; the model was also found to be better than the predictions made using expert opinions [16]. While SRGMs are useful for modelling reliability at the system/project level, applying them at lower levels (feature/sub-system) is difficult. The defect count at that level is small which poses a challenge for fitting the models to the observed partial defect inflow making it difficult to make reliable forecasts.

Regression based SDP approaches have been quite successful. Logistic regression have been used by Khoshgoftaar and Allen [29] for classifying modules as fault-prone or not. Logistic regression was also used to classify file/packages in Eclipse project as defect prone in the study done by Zimmermann, Premraj and Zeller [30]. Multiple linear regression has been used to model software changes where a set of software complexity metrics were used as independent variables [23]. Khoshgoftaar et al. [24] used linear regression for predicting program faults, their model also relied on set of code complexity metrics and number of changes to a given module to predict the dependent variable (program faults).

Bell et al. [167] used negative binomial regression models for predicting software faults in an industrial voice response system. The study confirmed Pareto distribution of faults and used set of simple, readily accessible predictor variables that are independent of programming language. Nonetheless the predictor variables used require access to source code and evolution metrics that we do not use in our prediction models. Further Weyuker et al. [168] also compared between four modelling methods that included negative binomial regression, random forests, recursive partitioning and Bayesian additive regression trees for such prediction models and found the first two outperforming the latters.

Our work complement these earlier works based on regression techniques, we also use a linear regression model with one main difference. Instead of set of code and change metrics as independent variables we only use defects found in/until a given integration point to make future defect predictions. Thus the model is simple and can be applied with little effort - it does not require access to source code, static analysis nor the information on software evolution/change during its development. Table 32 provides a brief overview of some of the related work in the area of software defect prediction.

Table 32: Overview of related work

Predictor variable	Summary of work
Static code analysis	Zheng et al. [169] provide empirical evidence on the usefulness of static analysis for fault detection in software. Their analysis of projects from Nortel Networks showed that static analysis provides cost effective means for detecting software faults, it is particularly useful for identifying assignment and checking faults.
	Nagappan and Ball [170] showed strong positive correlation between defects found by static analysis and the actual pre-release defect density. In the case study of Windows Server 2003, they showed that defects found by static analysis can be used to classify components into high quality and low quality with overall accuracy of 82.91%.
Complexity metrics	Subramanyam and Krishnan [171] provides empirical support for link between Object-Oriented design complexity metrics and software defects. They found significant association between OO metrics and defect counts even after controlling for size.
	Nagappan, Ball and Zeller [172] empirically tested correlation between software complexity metrics to post-release defects. Authors were able to find set of complexity measure for each of five major projects (from Microsoft) that correlated with the post-release defects. It was also shown that no single set of complexity metrics provided the best defect predictor for all projects, highlighting the need for validation at the individual project level.
Software evolution or change metrics	Kim et al. [173] developed algorithm which caches 10% of source files based on change history of software project which is then used for predicting faults at file and feature/method level. They evaluated their approach on seven open source projects displaying good predictive ability.
	Nagappan and Ball [174] used relative code churn measures to predict the system defect densities. They also showed that the relative code churn measure can be used to discriminate between fault prone and not fault prone binaries.
	Snipes, Robinson and Murphy-Hill [175] presented a tool to mine change records from configuration management system (CMS) to highlight defect prone areas in the source code. In the study, defect risk for each file is predicted using software evolution (code change) metrics - number of unique developers and count of changes.
Complexity and change metrics	Nagappan and Ball [176] evaluated ability of software dependencies and churn measures as predictors for post-release failures using source code complexity metrics (dependencies) and change metrics (code churn measures).
	Kim et al. [177] used complexity and change metrics to classify changes introduced to software as clean or buggy. The proposed change classification approach was applied on 12 open source projects giving an average accuracy of 78% with 60% recall.
Other measure	Zimmermann and Nagappan [178] used network analysis on dependencies graphs to identify program units that are more likely to contain defects. The approach was shown to provide 10% point higher recall than using only complexity metrics in their case study of Windows Server 2003.
	D'Ambros, Lanza and Robbes [179] provide an extensive comparison of well-known bug prediction approaches on publically available data set. They found churn and entropy of source code as best classes of metrics overall.
	Fenton and Neil [20] provide a critical review of software defect prediction methods based on size and complexity metrics. They recommend holistic models using Bayesian Belief Networks which are capable of modelling the causal relationships between the variables.

As explained earlier our thesis is that in cases where such information (complexity and change metrics) may be difficult to obtain - simple prediction models which do not require access to source code or information related to software evolution may provide useful alternative for making quick defect count predictions and help software practitioner with information that they need to make early intervention decisions.

Predictions using only defect inflow data

The work presented here closely follows the study by Yu, Shen, and Dunsmore [28] who evaluated the relationship between defects in earlier and later phases using linear regression model. The authors tested the model on two large software projects and found strong linear relationship between defects discovered in earlier phase and those discovered later. While there are some differences to our study, in general our results support the earlier findings. Compared to earlier study where the focus was on relationship between different phases, we investigate the strength of relationship between software integration points in large software projects following iterative development process. We also evaluate the correlation between defects found until a given integration point and total pre-release defects. Predicting pre-release defects is of great interest for industry practitioners and our evaluations suggest that simple regression model may be used effectively to make these predictions and thus satisfy the information need of software engineers, quality, and project managers.

5.3 Research Methodology and Data

5.3.1 Case Study Design

The study presented here is a case study according to Robson's [106] classification, the main goal being to evaluate the relationship between defects found across different integration points over a large software project. The context is that of embedded software development projects where we use the data from the automotive domain. Following the taxonomy and guidelines for conducting and reporting case studies in software engineering by Runeson and Höst [52], the case study presented in this paper is an interpretive study using a fixed design principle. The research is organized as an embedded case study with unit of analysis as sub-systems and software features. While we evaluate the defect data from four large software projects, the main interest is on evaluating if correlation exists at the level of sub-systems or features, thus the case study is evaluated at that unit level.

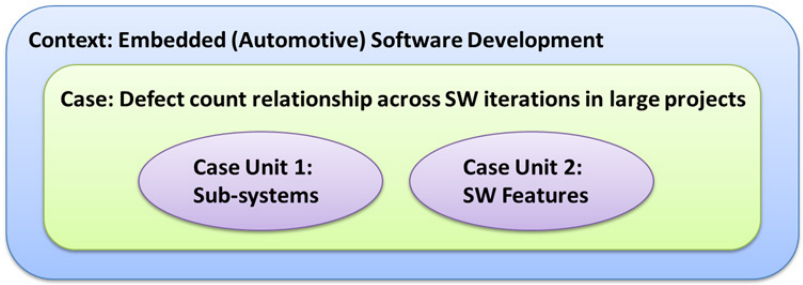

Figure 58: Overview of case study design

The reason to choose sub-systems and features as case units is mainly driven due to practical relevance. Defect prediction at project level is helpful for project and quality managers to monitor the progress of project and evaluate quality and reliability characteristics of the software system under development, but such predictions do not provide predictions at the level where software engineers and component (sub-system/feature) owners could take corrective actions. Predictions at sub-system and feature level provide information to these stakeholders who can use these predictions to make changes to design and put more focus on implementation of software features or sub-systems (in next integration points) which are predicted to be more prone to defects.

5.3.1.1 Company Profile: Volvo Car Group

Volvo Car Group (VCG) is a Swedish car Original Equipment Manufacturer (OEM) based in Gothenburg. VCG develops software and hardware in a distributed software development environment using a number of sub-suppliers. For a minority of Electronic Control Units (ECUs) software is developed in-house. The development is done by the software development teams who usually also have responsibility for integrating the software with the hardware developed by the suppliers. The majority of the embedded software deployed in the car, is however developed by external suppliers who design, implement, and test the functionality based on specifications from VCG [108], [109]. The size of the entire automotive project in terms of resources is substantially large due to the fact that both OEM and suppliers (first and second tier) are involved and car development projects are usually conducted using the product line approach with reference architecture [110].

5.3.1.2 Software Development Process

The software (SW) development process at VCG predominantly follows V-model (Figure 2), the projects studied here are platform projects which span for a long period of time and are thus divided into number of integration points (marked as Integration points 1, 2, to 9). Every integration point is effectively a new sprint within a larger project where new functionality is designed, developed, tested,

verified, and released into the latest system builds. The iterative software development model used is shown in Figure 59.

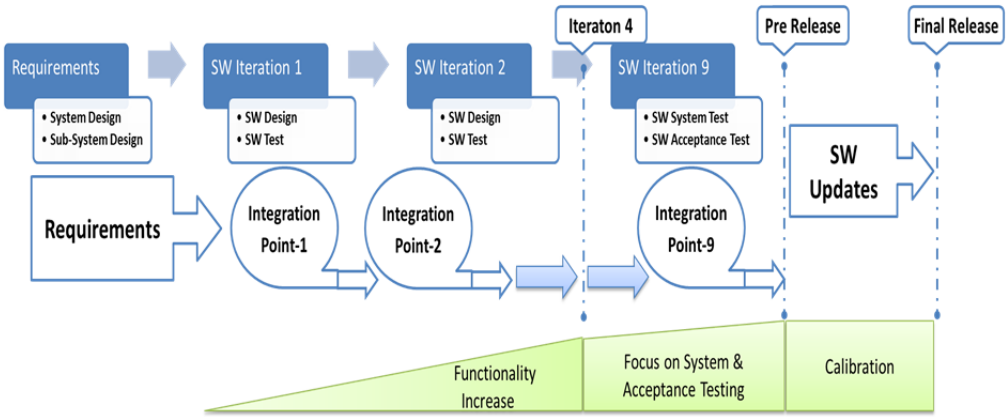

Figure 59: Representation of iterative software development process at the case company

The project starts with setting up the requirements following which each integration point consist of design, implementation (in-house or using suppliers) and the last part of each integration point is concentrated on testing the newly developed software. The emphasis until integration point 4-5 is on adding new functionality, while after integration point 4-5 the focus is shifted towards system and acceptance testing although more functionality may also be added. The integration points are followed by phase where calibration and optimization is the main activity. Defects found while testing the software are removed as they are detected or patches are provided in the software updates.

Within each integration points, development of software is a continuous process (which approximately follows the V-model, Figure 2) where requirements are finalized, functionality is designed and implemented in form of implementation models or source code, after implementation various levels of testing (unit, function and integration) is done to ensure the final product is what was originally intended.

5.3.2 Data Collection and Analysis Methods

5.3.2.1 The Basic Data

In this study the defect data is collected from four large software development projects (Proj-1 to Proj-4) from the automotive domain. The projects come from the E/E (Electrical and Electronics) integration department within the VCG which deals with the integration of various software functionalities and responsible for the final

assessment of full EE hardware and software systems. All the projects consist of different modules developed by different teams and tested within the development team (unit testing), while further integration and acceptance testing is done by dedicated teams in the integration department. All defects detected during integration, system and acceptance testing phases are reported in one of the central defect database administered by the integration department - which was also the source of data compilation for this study. Data was collected in close cooperation with the industrial partners; projects used in this study have been completed during the last decade, thus we had full data at hand.

In terms of project timeline, on average each project spans between two to three years[11]. In the first stage that spans about four to five integration points the focus in on development of functionality, while in later stages (beyond integration point 5) – the focus is on system and acceptance testing. On average by the time integration point four and five are reached, the project is about 60% and 75% complete respectively with respect to planned timeline. Although time period between integration point four-five and pre-release is approx. 40%-25% (refer Figure 60), it still represents close to one year in calendar time and major part of test resources utilization (since the focus in this phase is on system and acceptance testing). Thus predicting pre-release defect count by integration point four-five have high practical importance, as it allows quality and project managers to effectively allocate and mobilise test resources as per the expected demand of project under development.

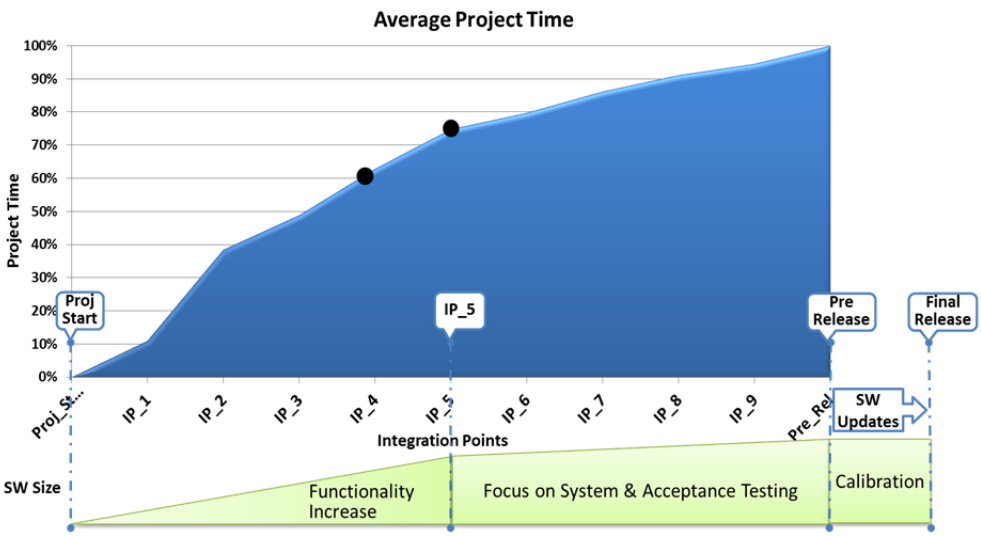

Figure 60: Average project timeline over integration points

[11] The exact time period and much of detailed data about individual projects is confidential

5.3.2.2 The Analysis Methods

Next we provide a brief overview of the analysis method used for answering the research questions ***(RQ1-4)*** that we address in this paper,

RQ1: *does small number of modules contain most of the defects found in large automotive software projects?*

To answer the first research question, we evaluated what percentage of defects is accounted by top 20 percent of sub-systems and features. The analysis is done at the module level for all data and also at individual project level to give a detailed picture.

RQ2: *Do defects found in current integration point strongly correlates to defects found in next integration point?*

For research question RQ2, where the objective is to measure the strength of relationship between defect counts across different integration points, we evaluate the correlation coefficient (or Pearson product moment correlation coefficient, r) between the two variables. Since there is no universally agreed value to determine if the correlation is strong or weak as it depends on the context of problem, for our analysis we peg the levels as: the correlation is considered weak if $r < 0.6$, moderate if $0.6 \leq r < 0.8$, strong for $0.8 \leq r < 0.9$ and very strong for $r \geq 0.9$.

Simple linear regression model is also constructed with one dependent variable (Y) and one independent variable (X). The model is represented by equation (1):

$$\text{Equation 1: } Y = \beta_0 + \beta_1 X + \eta$$

where β_0 is the intercept, β_1 the regression coefficient and η is the error term. The square of correlation coefficient, also called coefficient of determination(R^2), is also provided for each regression, which is a measure of how well the regression line represents the data or what proportion of the variance of one variable is predicable from the other variable. The dependent and independent variables for research question two are:

X=Number of defects reported in given (current) integration point,
Y=Number of defects reported in next (current+1) integration point.

RQ3: *How can we use defect inflow data for continuous quality monitoring (i.e. early risk identification)?*

Identification of sub-systems and features for further review

The correlation between defects found between consecutive integration points, can also be used for identification of set of modules (sub-systems and features) that may need more attention.

Using simple linear regression model (as represented in equation 1) between defects found in consecutive integration points. The 95% confidence bound can be calculated as,

Lower bound: $\mathrm{LB} = (\beta_0 - 2\sigma_0) + (\beta_1 - 2\sigma_1) * \mathrm{X}$,

Upper bound: $\mathrm{UB} = (\beta_0 + 2\sigma_0) + (\beta_1 + 2\sigma_1) * \mathrm{X}$

where σ is the standard deviation for the given coefficient value and X is the defect count in given (current) integration point.

If the observed defect count in next integration point falls outside of the upper or lower bound, could be an indicator of given modules higher than expected defect-proneness or potential lack of test coverage. Thus such modules may be selected for further review by the quality and project managers.

RQ4*: When in project timeline can we make useful pre-release defect count predictions?*

The primary objective of this research question is to identify at what stage of project development, total defects expected to be found by the pre-release can be predicted. We first use correlation between number of defects found at a given integration point with total pre-release defects found at the level of sub-systems and features to find the earliest time in project timeline when such prediction can be made.

Secondly to check the predictive ability of such prediction model, we build models based on simple linear regression as represented in equation (1) above with variables:

X=Number of defects reported by pre-release,

Y=Number of defects found at (or until) integration point four/five.

The prediction model is validated using 10-fold cross-validation and using relative absolute error for assessing the predictive accuracy. Relative Absolute Error is given by equation (2) as:

$$\text{Equation 2: } \frac{|p_1 - a_1| + \cdots + |p_n - a_n|}{|\bar{a} - a_1| + \cdots + |\bar{a} - a_n|}$$

where p_n is predicted value for nth observation, a_n is the actual value for that observation and $\bar{a}$ the mean of all actual values. Thus relative absolute error measures predictive performance of given prediction model in comparison of a simple predictor that predicts average values. Thus a value of less than 100%

indicates better performance than simple predictor while a value relative absolute error of over 100% indicate performance worse than using mean value predictor.

Note on outliers: The defect data we evaluated (Proj-1 to Proj-4) constituted a total of 140 sub-systems and 178 features. Two out of 140 sub-systems and one out of 178 features were excluded from the analysis due to practical reasons. The two sub-systems and one feature excluded all came from a single project (Proj-2) and were considerably different from the typical sub-system and features developed under software projects at the studied company. Thus to ensure that our analysis is practically relevant and is not influenced by such one-time events, these modules were excluded from the analysis. The decision to exclude these from analysis was done after close consultation and approval from the industrial partner who pointed out the unique characteristics of these modules. Thus the final sample size was 138 sub-systems and 177 features and data was available for 9 integration points before the pre-release. Thus for *RQ2-4*, doing the analysis for all sub-systems and features gave a sample size of 1104 and 1416 observations respectively at sub-system and feature level.

Again for practical reasons the analysis is not only done at all sub-system and feature levels, but also for a smaller sub-set consisting of Top-15, Top-10 and Top-5 sub-systems and Top-15 and Top-10 features (Top-X here refer to X sub-systems/features with most defect count at pre-release). The main reason for evaluating the model at smaller sub-set is to check if the relationship which we observe and prediction models thus build using all data also holds for these sub-sets of data. The sub-sets evaluated in the study have high practical importance: during a large project, developers, system owners and managers are more concerned with monitoring and predicting defect counts for sub-systems and features that are most defect-prone where it would be most helpful to optimize test resource allocation and early interventions in design and implementation will provide best returns in terms of higher quality.

5.4 Results

In this section we present and discuss the results from the evaluation of strength of relationship between defect counts across integration points in large software projects and evaluate if it can be used to build simple regression models that can be used for defect predictions. We first present the defect count distribution over the nine integration points in the four projects and then the results are organised according to five hypotheses we test in this study.

5.4.1 Does small number of modules contain most of the defects found in large automotive software projects?

Large-scale software development projects may include hundreds of designers and testers working at different sub-systems and features level. Due to various factors such as size, complexity etc., not all sub-systems or features report defects on same magnitude and severity. This presents an opportunity where software quality assurance activities can be made more efficient by effectively allocating more proportionally to the defect proneness of these modules.

Some earlier studies have shown that majority of defects and failures are found in few of the sub-systems and features [162] [163], we test this observation for the projects in this study under our first research question ***(RQ1)***. The results are provided in Table 33 and defect distribution by sub-systems for all projects is shown Figure 61.

Table 33: Summary of defect distribution and percentage of modules with 80% of reported defects

Module	*Project*	*N, total number of modules*	*%age of defects in top 20% modules*
Sub-systems	*Proj-1*	50	76.0%
	Proj-2	33	67.7%
	Proj-3	28	85.4%
	Proj-4	27	89.6%
Features	*Proj-1*	43	85.3%
	Proj-2	47	80.2%
	Proj-3	48	57.1%
	Proj-4	39	66.1%
Total sub-systems		138	81.0%
Total Features		177	83.0%

For full dataset which constitutes 138 sub-systems and 177 features the, 20% modules accounted for majority (more than 80%) of reported defects. At individual project level percentage of reported defects in top 20% sub-systems range from 67% to 89%. For features at project level, again except for one project (Proj-3), top 20% of features accounted for more than 60% of reported defects.

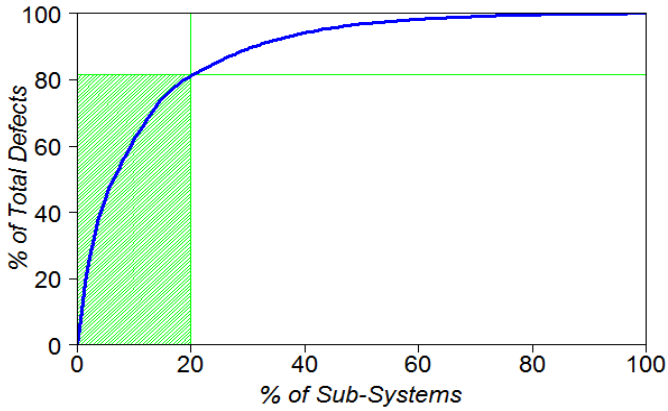

Figure 61: defect distribution at sub-systems and features level

Thus with an exception of Proj-3, we confirm the earlier observation that 20% modules account for more than 60% reported defects [163]. The results that 20-60 rule holds for large iterative automotive software projects has practical importance. From given projects, we infer that a small set of modules (sub-system and features) account for major proportion of defects. The observation adds further evidence that Pareto principle of defect distribution also holds for large automotive software projects.

5.4.2 Do defects found in current integration point strongly correlates to defects found in next integration point?

For second research question, we test if we can use the defect count in a given integration point (current) to make a good prediction of expected number of defects to be found in next (current+1) integration point. Such predictions can help designers and managers to identify the most defect-prone modules in the upcoming integration points and may trigger some corrective actions early for example by making modifications on design or allocation of more resources to ensure high quality. Figure 62 shows the scatter plot for all sub-systems and features showing the relationship between the defect count in current and next integration points.

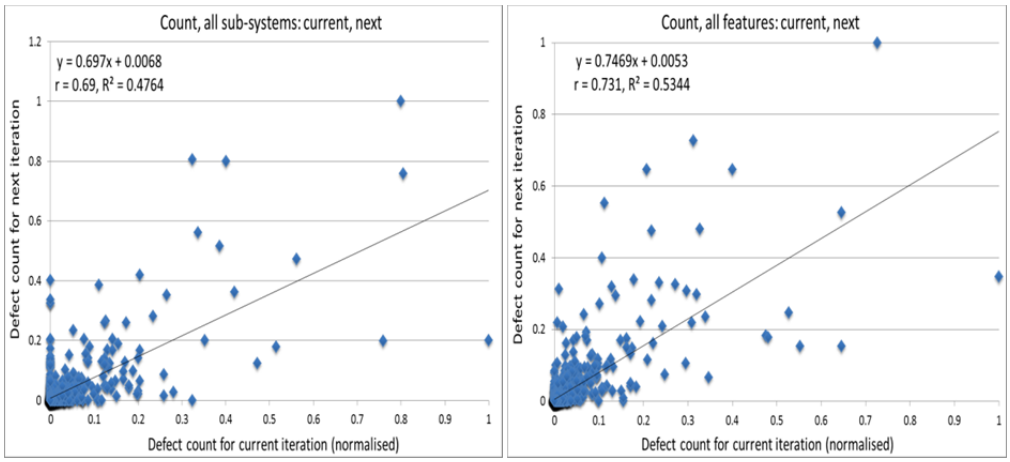

Figure 62: Scatter plot for defect count in current and next (current+1) integration point for all sub-system and features

We observe that while there exists a positive correlation between the defect count in current and next integration points, it is not very strong. The linear regression line also help get a subjective overview with number of points away from the line show correlation between the two variables is not strong. The correlation coefficient (r) and the results of regression model are presented in Table 34 for sub-sets with different modules and scope. We note that correlation coefficient for all cases is moderate ($0.6 < r < 0.8$) but not strong.

Table 34: Regression results for the prediction model for defect count in next (current+1) integration point using defect count in current integration point

Module	Scope	Sample Size	Coefficient ($\beta 1$)	Standard Error	P-value	r, Correlation Coefficient	R^2, Coefficient of Determination
Sub-systems	Top 15	480	0.676	0.034	0.000	0.670	0.449
	Top 10	320	0.665	0.042	0.000	0.660	0.436
	Top 5	160	0.647	0.061	0.000	0.645	0.415
Features	Top 15	480	0.711	0.033	0.000	0.700	0.490
	Top 10	320	0.695	0.041	0.000	0.687	0.472
All sub-systems		1104	0.697	0.022	0.000	0.690	0.476
All Features		1416	0.747	0.019	0.000	0.731	0.534

This (weak correlation between consecutive integration points when measured for all integration points combined) is not un-expected given that defects found per unit time tend to increase as project progresses until a certain time, stabilizes and then begin to fall off as the software matures – giving a characteristic S-shape to defect inflow profile [180]. What is interesting however is - if there exists any strong

correlations between the consecutive integration points. To test this, we measure the correlation coefficient across integration points one to nine and results are presented in Table 35. It is observed that correlation is weak across IP_1 to IP_3 particularly for sub-systems. Overall for all sub-systems correlation across consecutive integration points is moderate (between IP_5 to IP_7), strong (between IP_3-IP_4 and IP_7-IP_8) and very strong between IP_4-IP_5 and IP_8-IP_9. For all features the correlation between defects found across integration points range from moderate to very strong.

Table 35: Correlation coefficient between defects found across integration points one to nine

Module	Scope	Sample Size	IP_1 - IP_2	IP_2 - IP_3	IP_3 - IP_4	IP_4 - IP_5	IP_5 - IP_6	IP_6 - IP_7	IP_7 - IP_8	IP_8 - IP_9
Sub-systems	Top 15	60	-0.09	-0.11	0.85	0.96	0.67	0.57	0.79	0.95
	Top 10	40	-0.10	-0.14	0.85	0.96	0.64	0.52	0.78	0.95
Features	Top 15	60	0.85	0.68	0.88	0.71	0.77	0.89	0.93	0.88
	Top 10	40	0.85	0.66	0.88	0.69	0.76	0.88	0.93	0.91
All sub-systems		138	-0.05	-0.07	0.85	0.96	0.72	0.63	0.81	0.95
All Features		177	0.85	0.70	0.90	0.75	0.82	0.90	0.94	0.89

Thus for integration point three onwards, we find correlation between defects found across consecutive integration points to be moderate-to-very strong. This information can be used in subsequent projects to check if a given module follows general trend of defects found at a given integration point which can help identify sub-systems/features for further review - described in following section.

5.4.3 Identification of sub-systems and features for further review

As explained in section 3.2.2, using 95% confidence intervals of regression coefficients, we can find an upper and lower bound of forecasted defect count for next integration point; an example is shown in Figure 63.

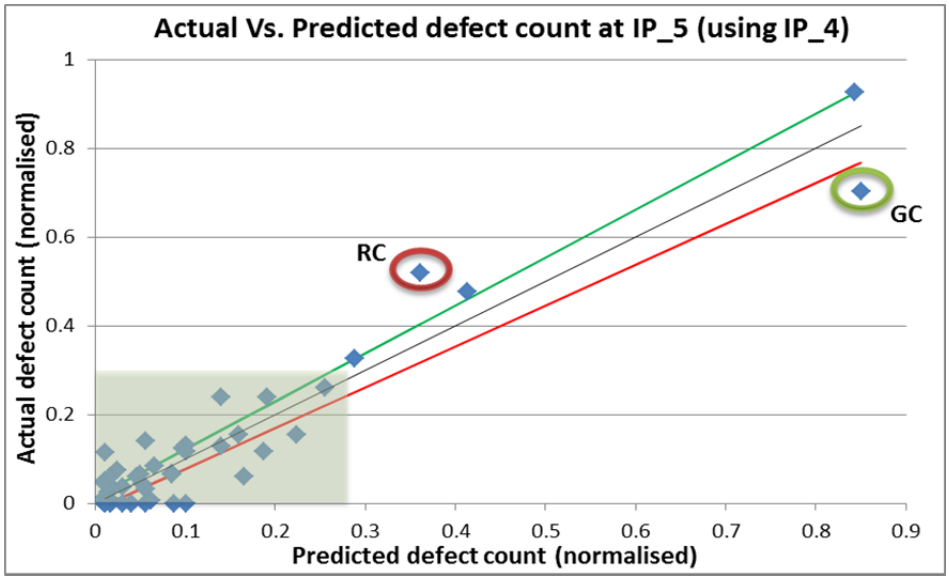

Figure 63: Scatter plot showing potential identification of sub-systems or features for further review. The graph shows actual Vs. predicted defect count for IP_5 using defect count at IP_4 for all features

In the review phase after a given integration point and before a subsequent integration point begins, mangers can use the forecasted values of defect counts and actual defect counts for identification of sub-systems or features that need extra attention. For example in Figure 63, these are marked by red, and green circles.

For the software modules (sub-systems/features) that report defect counts much higher than the forecasted upper bound are marked by red circles, managers may choose to take actions that may include:

- root cause analysis,
- design inspection/review,
- dependencies check,
- further reliability analysis, etc.

The modules that report less than the forecasted lower bound (marked by green circle) may also need more attention; the actions taken by mangers for these modules among others may include:

- check the adequacy of test cases,
- initiate more testing,
- inspection or manual walkthrough,
- root cause analysis, etc.

Not all modules outside the lower and upper bound would need close inspections or further actions, there may be practical reasons for the observed deviations (for example less functionality addition in given integration point compared to previous or reusability of code etc.). Also for cases where the defects count is low, small deviations may not be as important as modules with high defect count and large deviations. Small deviations at low defect count may just be due to low defect count at previous integration point or start of new functionality addition to a given module. Thus given on available information these modules may be taken for further review or not.

It is thus noted that simple regression models are not only useful to make a forecast of defects expected to be found in software modules in next integration point which can be used to allocated necessary resources, but also can help identify smaller subset that may need more attention.

5.4.4 When in project timeline can we make useful pre-release defect count predictions?

When it comes to defect prediction and management, one information that most stakeholders are interested in is to predict the number of defects expected to be found by pre-release. Important questions in this respect are if we can predict (with good accuracy) number of pre-release defects when the project is on-going and when these predictions can be made with respect to project timeline. Pre-release defect count predictions at project and components level help project managers and quality assurance mangers to ensure that they plan for expected resource demands. It also helps these mangers to monitor the progress of project with respect to its quality and reliability characteristics. At individual team level, the sub-system responsible and software engineering use pre-release defect counts at sub-system and feature level to identify which features (area of code) need more attention. If the predictions highlight specific features to be defect prone, early interventions in terms of design change or refactoring may be undertaken to improvise the quality and thus avoid defects late in the development process.

To answer these questions we analyse the correlation between defect count reported at each integration point with the number of pre-release defects. Table 36 shows correlation matrix for defect count at each integration point with the total defect count by the pre-release at sub-system and feature level. The analysis is also done at sub-sets that are of high practical value to company such as Top10 sub-systems or Top10 features. It is observed that correlations between defect count at individual integration point and pre-release are mostly weak for integration points 1-2 and for integration points 7-9. The correlation is moderate/strong between pre-release defect count and integration point 3 and 6. For integration points 4 and 5, defect count

show the highest correlation with the pre-release defect count where correlation is either strong or very strong.

Table 36: Correlation matrix for defect count in given integration point with pre-release defect count

Correlation Coefficient	Sub-systems				Features		
	All	Top 15	Top 10	Top 5	All	Top 15	Top 10
IP_1	0.424	0.374	0.336	0.527	0.278	0.231	0.205
IP_2	0.262	0.221	0.194	0.112	0.609	0.566	0.542
IP_3	0.748	0.735	0.722	0.786	0.823	0.799	0.791
IP_4	0.911	0.906	0.904	0.907	0.863	0.841	0.837
IP_5	0.943	0.938	0.934	0.937	0.930	0.913	0.912
IP_6	0.824	0.784	0.756	0.664	0.721	0.651	0.618
IP_7	0.556	0.474	0.400	0.169	0.720	0.651	0.623
IP_8	0.437	0.371	0.321	0.176	0.655	0.580	0.565
IP_9	0.353	0.295	0.246	0.120	0.659	0.606	0.614
Pre_Rel	1	1	1	1	1	1	1

Cumulative defect count (total defect count until a given time/integration point) tend to be a more stable measure as it averages out fluctuations of defect count over individual iterations, it also carries more information as it is an aggregate measure of defects reported until a given date. Correlation between cumulative defect count at end of each integration point to pre-release defect count is summarized in Table 37.

Table 37: Correlation matrix for cumulative defect count in given iteration with pre-release defect count

Correlation Coefficient	Sub-systems				Features		
	All	Top 15	Top 10	Top 5	All	Top 15	Top 10
IP_1	0.424	0.374	0.336	0.527	0.278	0.231	0.205
IP_2	0.396	0.343	0.309	0.210	0.578	0.531	0.492
IP_3	0.797	0.775	0.756	0.796	0.784	0.753	0.734
IP_4	0.909	0.901	0.894	0.900	0.865	0.844	0.837
IP_5	0.936	0.930	0.925	0.927	0.930	0.917	0.916
IP_6	0.963	0.957	0.953	0.948	0.961	0.952	0.952
IP_7	0.979	0.975	0.972	0.967	0.986	0.983	0.983
IP_8	0.994	0.993	0.992	0.991	0.995	0.994	0.994
IP_9	0.999	0.999	0.999	0.999	0.999	0.999	0.999
Pre_Rel	1	1	1	1	1	1	1

The correlation between cumulative defect count and pre-release defects is either strong or very strong starting from iteration 4. The correlation between the number of defects found at and by a given integration point to total defects found by pre-release is also presented using line chart in Figure 64 and Figure 65 for sub-systems and features respectively.

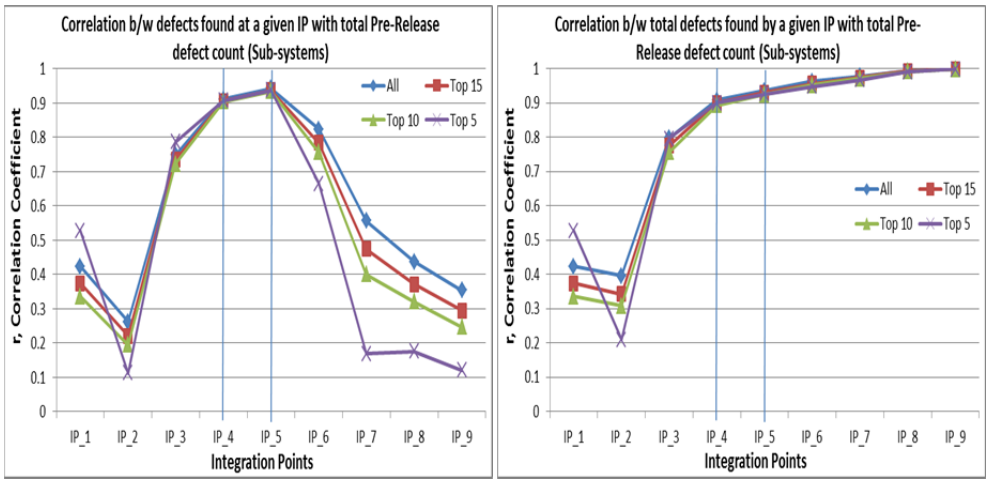

Figure 64: Line plot showing correlation between defects found at and by (cumulative defects) a given integration point with total defects found by pre-release for sub-systems

The figures (Figure 64 and Figure 65) show clearly that the correlation between defects found at a given integration point with total defects found by pre-release is strongest at integration point 5. Also correlation between defects found at integration point 4 and pre-release defect count is very strong for sub-systems (>0.9) and strong for features (>0.8). Further as expected, correlation between cumulative defect count at given integration points and pre-release defect count in general follows a concave profile due to smoothening of variations over defect count at individual integration points.

From Tables (Table 36 and Table 37) and their representation in Figure 64 and Figure 65, it is also observed that until integration point 5, correlation with pre-release defect count is slightly higher with defects found at a given integration point than with cumulative defect count at the same integration point.

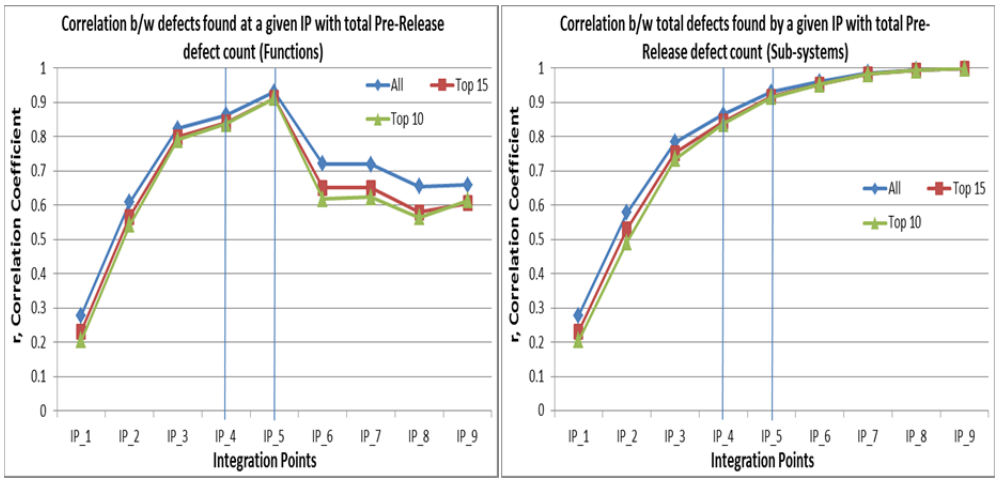

Figure 65: Line plot showing correlation between defects found at and by (cumulative defects) a given integration point with total defects found by pre-release for features

It is noteworthy here that Integration point 4 is reached when project is about 60% through its planned timeline (refer Figure 60) and thus if predictions for pre-release defect count could be made at this time, it can prove to be of high practical importance with respect to assistance in test resource planning and allocations to projects.

Predictions using linear regression model

To build the prediction models we use the simple regression model described earlier, the model is represented by equation 3,

$$\text{Equation 3: } Y = \beta_0 + \beta_i X_i + \eta$$

We build two types of model and check their predictive accuracy using relative absolute error (as described in section 3.2) and using 10-fold cross-validation, these models are:-

1. **Simple linear regression models:** in these models the dependent variable (Y) is pre-release defect count which is predicted using only one independent variable (X). We use defects found at integration point 4 or 5 as the predictor variable, the results are summarised in Table 38.
2. **Multiple linear regression models:** in these models for same dependent variable (Y, pre-release defect count) is predicted using defect counts at integration points until integration point 4 or 5. Since this model takes into account not only defects found at integration point 4 or 5 but also the evolution of defects found until that point, it is expected to provide higher

predictive accuracy. The results for multiple linear regression models are summed up in Table 39.

Table 38: Results of simple linear regression models for predicting pre-release defect count using defect count at integration point 4 or 5

Module	Scope	Sample Size	Linear Regression Model	Relative absolute error	Root relative squared error	r, Correlation Coefficient	R^2
Sub-systems	Using IP_4	138	Pre-Rel = 3.13*IP_4 + 15.9	40.7%	43.3%	0.901	0.812
	Using IP_5	138	Pre-Rel = 2.72*IP_5 + 13.84	31.1%	34.0%	0.939	0.883
Features	Using IP_4	177	Pre-Rel = 2.97*IP_4 + 24.4	47.5%	53.7%	0.846	0.716
	Using IP_5	177	Pre-Rel = 5.45*IP_5 + 7.41	34.7%	41.1%	0.911	0.830

Table 39: Results of multiple linear regression models for predicting pre-release defect count using defect counts until integration point 4 or 5

Module	Scope	Sample Size	Linear Regression Model	Relative absolute error	Root relative squared error	r, Correlation Coefficient	R^2
Sub-systems	Until IP_4	138	Pre-Rel = 4.38*IP_1 + 1.65*IP_2 - 1.02*IP_3 + 3.30*IP_4 + 11.58	38.5%	41.7%	0.909	0.826
	Until IP_5	138	Pre-Rel = 3.44*IP_1 + 0.81*IP_2 - 0.94*IP_3 + 0.90*IP_4 + 2.13*IP_5 + 11.41	31.6%	35.6%	0.934	0.872
Features	Until IP_4	177	Pre-Rel = 1.37*IP_2 + 2.60*IP_4 + 22.62	49.3%	59.8%	0.818	0.670
	Until IP_5	177	Pre-Rel = 2.75*IP_1 + 0.56*IP_2 + 1.25*IP_3 + 4.13*IP_5 + 2.72	19.0%	19.3%	0.981	0.962

The results from simple and multiple linear regression models (from Table 38 and Table 39) are as follows:

a. Only using defects found at integration point 4 or 5, pre-release defect count can be predicted with good accuracy (compared to mean value predictor),
b. Using defects found at integration point 5 provided better predictive accuracy than using defect count at integration point 4,

c. The best predictions for sub-systems (among tested models) was obtained using simple linear regression model using defects found at integration point 5 as predictor variable, and
d. Best prediction model for features was obtained using multiple linear regression model using defect counts until integration point 5 as the independent variables for predicting pre-release defects.

5.5 Recommendations and Threats to Validity

5.5.1 How to apply the prediction models:

Recommendation to software quality and project managers for using correlation based simple defect prediction models

Quality assurance and project managers may:

- Identify high risk modules using predictions confidence intervals (as explained in section 4.3) and use the information to ensure that necessary support in terms of required resources is available to teams working on such modules. Also the information will facilitate managers to allocate their attention effectively to areas that need it most.
- Use simple prediction models after integration point 4 (as explained in section 4.5) to forecast total number of expected defects at pre-release for long-term planning and monitoring the progress of system verification and validation activities at complete system/project level and also at individual teams level.

Software development and testing teams can benefit from simple correlation based defect prediction modelling in following ways:

- They can use the simple rule based predictions (e.g. for a given feature by end of integration point X+1 we usually expect to find Z times the number of defects we found by end of integration point X) as a quick check to see if any particular modules stand out and if so investigate the reasons (root cause) or take appropriate actions as demonstrated in section 4.3.1.
- By continuously monitoring their current scorecard (with respect to defect counts) and comparing it against the forecasts, the team leaders and members at each team can monitor their own progress. Check if they will be able to deliver their part of software by expected release date and if needed when it is the time to call for more support such that corrective actions are taken on right time rather than late in the development process.

5.5.2 Threats to validity

We address the threats to validity in manner as described by Wohlin et al. [37]. Threat to conclusion validity is minimized by studying the correlation between the dependent and independent variable. In this study the dependent and independent variable are essentially the same (defect count or its cumulative value) only separated temporally, thus it is reasonable to assume that they are affected by similar underlying factors and thus we expect to see a relationship between the two. For a given sub-system or feature new functionality is added and tested in each integration point, and although it is possible to have major differences between the functionality added across integration points – in general the characteristics is expected to follow linear trend. A feature that is expected to be a large or complex feature at the end of process is also more likely to have large or complex functionality addition during the different integration points compared to a feature with is not expected to have such characteristics.

A threat to internal validity is minimized in this study as the independent variable always precedes the dependent variable (temporal precedence). We analyse the covariance (correlation) and build and recommend regression models only when the correlation is found to be strong or very strong. There is however threat to internal validity as the defect count is in a given sub-system or feature over different integration points could be due to other underlying factors such as size, complexity etc. But since the intention of building the regression model is intended to be only used to make defect count predictions in next integration points and we do not claim that there is causal relationship between the independent and dependent variable, this treat is not a major threat to the validity of our conclusions.

External validity concerns with the generalizability of results in settings outside of the particular study. In the work presented here, we evaluated strength of relationship between the defect counts across different integration points in large software development projects. We studied four projects which had different project characteristics (size, scope, complexity etc.), but it is also noteworthy that all projects came from a single domain (automotive) and a single company (VCG). Given that we studied four large projects which are finished over last decade, at level of sub-systems and features over nine integration points, provide us with large sample size and thus conclusions are drawn from large and heterogeneous projects. While we do not claim that results can be generalized to all large software projects, but given that most of such projects are broken down in an iterative manner, the results are expected to be more generally applicable than the scope of the given study.

5.6 Conclusions

Testing, verification and validation is a major part of software development process and an important activity to ensure desired quality and reliability is achieved. Software defects proved a real and observable indicator to monitor the reliability growth of software under development. Early estimation of how many defects are expected to be found at the system/project, sub-systems and features level can help with:

- Project managers to manage release cycle decisions and monitor progress,
- Quality assurance mangers to plan and allocate human and test resources optimally, and
- Help designers and developers to make early intervention in design and implementation to take corrective actions when needed

In this paper using defect data from four large software projects from the automotive domain, consisting of 139 sub-systems and 177 software features over nine integration points – we evaluated five hypothesis. The results from our study show that:

RQ1: *does small number of modules contain most of the defects found in large automotive software projects?*

We found evidence to support earlier observations, for all sub-systems and features – 20% modules account for more than 80% of all defect reported. While at individual projects level the 20-60 rule as observed by Fenton and Ohlsson [2] was supported.

RQ2: *Do defects found in current integration point strongly correlates to defects found in next integration point?*

The correlation between defects count in consecutive integration point when analysed for all integration points together was not found to be strong. Nonetheless, for most consecutive integration steps, the correlation between defects found across integration points is found to be moderate-to-very strong for all sub-systems, features and their subsets.

RQ3: *How can we use defect inflow data for continuous quality monitoring (i.e. early risk identification)?*

Following RQ2, for integration point 3 and beyond the correlation between defects found in consecutive integration points is found to be moderate-to-very strong. This

correlation can be used to build simple prediction rules for predicting expected defect count in next integration point using defect count in current integration point.

The simple prediction rule can also be used to identify sub-systems and features that report large deviations from the expected increase to investigate deeply the underlying reasons for such deviations. Thus the prediction model can be useful in practice to also help identify small subset of sub-systems and features where root cause analysis will yield useful insights and help identify any trends/patterns that if left unchecked could cause problems late in the development process.

Chapter 6:
Evaluating reliability characteristics of executable models

Included Publication:

VIII. R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, and F. Törner, ***"Early Verification and Validation According to ISO 26262 by Combining Fault Injection and Mutation Testing,"*** Published in Software Technologies, Springer, 2014, pp. 164–179.

6 EARLY VERIFICATION AND VALIDATION ACCORDING TO ISO 26262 BY COMBINING FAULT INJECTION AND MUTATION TESTING

Abstract— Today software is core part of modern automobiles. The amount, complexity and importance of software components within Electrical/Electronics (E/E) systems of modern cars is only increasing with time. Several automotive functions carrying software provide or interact with safety critical systems such as systems steering and braking and thus assuring functional safety for such systems is of high importance. Requirements for the safety assurance are specified partially by such functional safety standards as ISO 26262. The standard provides the framework and guidelines for the development of hardware and software for components deemed to be safety critical. In this chapter we argue that traditional approaches for safety assurance such as fault injection and mutation testing can be adapted and applied to functional models to enable early verification and validation according to the requirements of ISO 26262. We show how to use fault injection in combination with mutation based testing to identify defects early in the development process – both theoretically and on a case of self-driving miniature vehicles. The argument is grounded upon the current best practices within the industry, a study of ISO 26262 standard, and academic and industrial case studies using fault injection and mutation based testing applied to the functional model level. In this paper we also provide the initial validation of this approach using software of a self-driving miniature vehicle.

Keywords— Fault injection, Mutation testing, ISO 26262, Simulink, Model based development, Automotive domain, Safety critical software

6.1 Introduction

Nowadays, a typical premium car has up to 70 ECUs which are connected by several system buses to realize over 2000 functions [3]. As around 90% of all innovations today are driven by electronics and software, the complexity of car's embedded software is already high and expected to grow further. The growth is fuelled by cars beginning to act more proactively and provide more assistance to its drivers, which requires software to interact with hardware more efficiently and making more decisions automatically (e.g. collision avoidance by braking, brake-by-wire or similar functions). In total with about 100 million lines of code (SLOC), premium segment vehicles carry more software code than in modern fighter jets and airliners [181].

Software for custom functionality in modern cars is usually developed by multiple suppliers although it is largely designed by a single Original Equipment Manufacturer (OEM) like Volvo Cars. The distributed development and use of standards like AUTOSAR aims to facilitate reuse of software and hardware components between different vehicle platforms, OEMs and suppliers [182]. However, testing of such systems is more complex and even today testing of software generally accounts for almost 50% of overall development costs [183].

ISO-26262 in automotive domain poses stringent requirements for development of safety critical applications and in particular on the testing processes for this software. These requirements are intended to increase the safety of modern cars, although they also increase the cost of modern cars.

The position for which we argue in this paper is that *efficient verification and validation of safety functions requires combining Model Based Development (MBD) with fault injection into models with mutation testing*. This position is based on the studies of the ISO 26262 standard (mainly chapter 6 that describes requirements on software development but also chapter 4, which poses requirements on product development [35]). It is also based on previous case studies of the impact of late defects on the software development practices in the automotive domain (e.g. [184])

The requirements from the ISO 26262 standard on using fault injection techniques is challenging since it relates to the development of complete functions rather than components or sub-components of software. The current situation in the automotive sector is that fault injection is used, but it is used at the level of one electronic component (ECU) or one software system and rarely at the function level [185] [186].

The current state of art testing is not enough for detecting safety defects early in the automotive software development process since fault injection is done late in the development (when ECUs are being developed), which usually makes the detection of specification-related defects difficult and costly [184]. As much possible this detection should be done at the model level when the ECUs' functionality is still under design and thus, it is relatively cheap to redesign/reconfigure. The evidence from literature on successful use of fault injection shows that the techniques are indeed efficient in finding dependability problems of hardware and software systems when applied to compute [187]. Finally, to be able to increase the effectiveness of the fault injection strategies and identify whether the faults should be injected at the model, software or ECU level - Mutation testing should be applied to verify the adequacy of test cases and finally how the combination of these approaches when applied at the model level will enhance the detection of safety defects right at the design stage.

In this paper, we provide a roadmap, which shows how to introduce fault injection and mutation testing to modelling of automotive software in order to avoid costly late defects and increase the safety of modern and future cars. This paper is the extended version of our previous work [188] where we presented the theoretical approach. In this paper we include a validation of this framework on a set of software components of self-driving miniature vehicles. The system used for initial validation is developed using a code-centric approach which makes the framework more generic as the initial evaluation in [188] was conducted on model-based development.

The remaining of the paper is structured as follows: In the next section 6.2 we provide an overview of software development in automotive domain and associated concepts. This is followed by brief discussion on related work in section 6.3 and our position is presented and discussed in section 6.4. Section 6.5 presents the initial validation case for the framework and section 6.6 provides conclusions.

6.2 Background

In this section we take a brief overview on the current state of automotive software development process and environment, how safety is important in safety critical applications and overview of theoretical background on fault injection techniques and mutation testing.

6.2.1 Automotive Software Development & ISO 26262

Various software functions/applications developed within the automotive industry today are classed as safety critical, for example Volvo's City Safety feature consists of components that are safety critical.

Figure 66: Volvo Cars city safety function, image provided by Volvo Car Group

[3] gives examples of functions/areas within automotive domain with recent development which includes crash prevention, crash safety, advanced energy management, adaptable man-machine interface, advanced driver assistance, programmable car, car networking etc., much of these also fall within the safety critical functionality and thus demands high quality and reliability. Also a number of on-going projects are directed towards the goal of self-driving cars.

Software development in automotive sector in general follows the 'V' process, where OEMs take the responsibility of requirement specification, system design, and integration/acceptance testing. This is followed by the supplier, which develops the actual code that runs on ECUs. Although the code is tested at the supplier level (mainly unit testing), the OEMs are responsible for the final integration, system and acceptance testing to ensure that the given implementation of a software (SW) meets its intended functional and safety goals/demands.

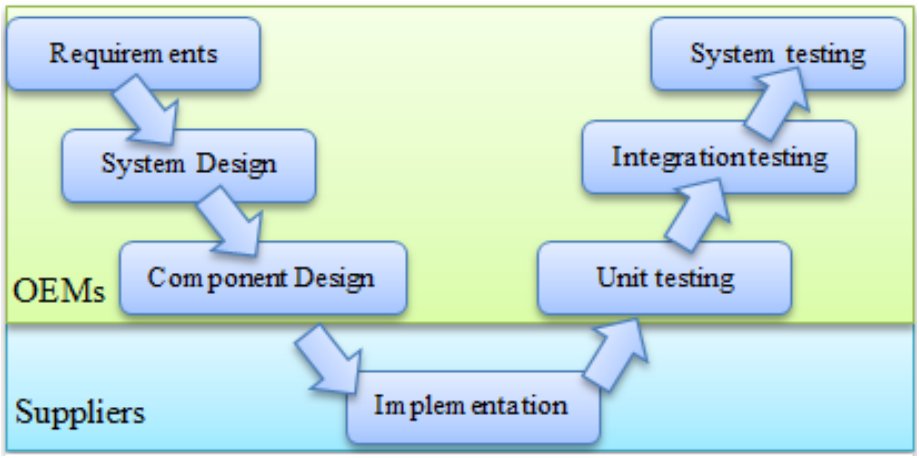

Figure 67: The V-model in the automotive industry with distinction between the OEM and supplier contributions

In this model of software/product development (see Figure 67) testing is usually concentrated in the late stages of development, which also implies that most of the defects are discovered late in the development process. In a recent study using real defect data from an automotive software project from the industry [189] showed that late detection of defects is still a relevant problem and challenge yet to overcome. The defect inflow profile presented in this study is reproduced in Figure 68 for reference, which exhibits a clear peak in number of open defects in the late stages of function development/testing.

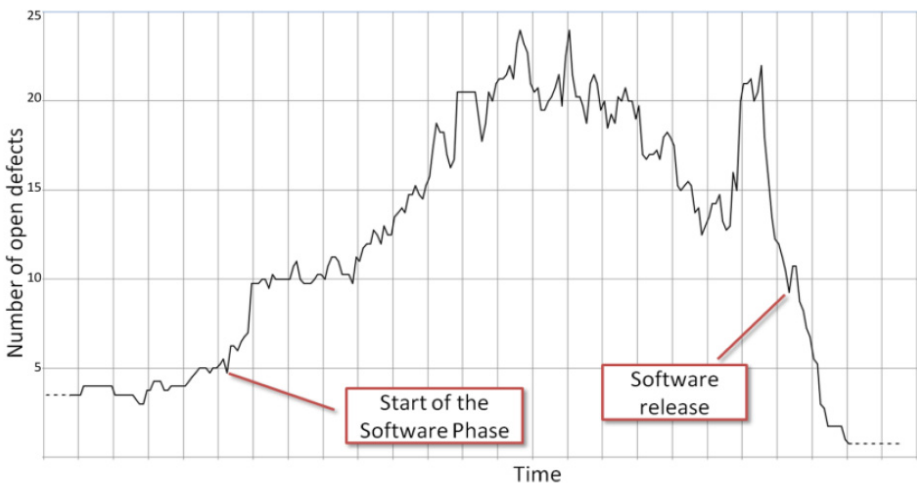

Figure 68: Defect inflow profile for automotive software project, as given in (Mellegård et al., 2012)

Testing the software is an important tool of ensuring correct functionality and reliability of systems but it is also a very resource intensive activity accounting for up to 50% of total software development costs [2] and even more for safety/mission critical software systems. Thus having a good testing strategy is critical for any industry with high software development costs. It has also been shown that most of the defects detected during testing do not depend on actual implementation of code, about 50% of defects detected during testing in the study by [190], were found during the test preparation, an activity independent of the executable code. And since automotive sector has already widely adopted MBD for the software development of embedded systems, a high potential exists for using the behavioural modes developed at the early stages of software development for performing some of the V&V (Verification & Validation). Early V&V by helping to detect defects early will potentially save significant amount of cost for the projects and reduce the cycle time.

6.2.2 ISO 26262

ISO/IEC 26262 is a standard describing safety requirements. It is applied to safety-related systems that include one or more electrical and/or electronic (E/E) systems. The overview of safety case and argumentation is represented in Figure 69, based on [35].

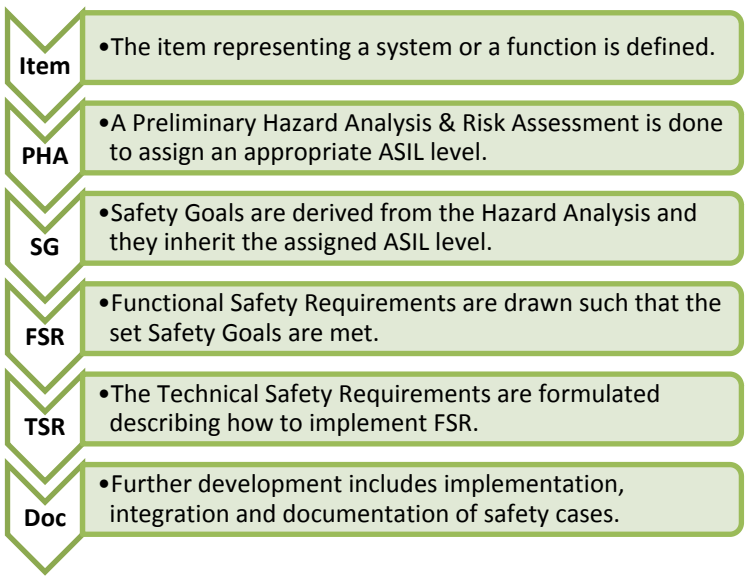

Figure 69: Overview of ISO-26262 safety case & argumentation process

Written specifically for automotive domain/sector, the ISO-26262 standard is adapted for the V-model of product development corresponding to the current practice in the industry. The guidelines are laid out for system design, hardware and software design & development and integration of components to realize the full product. ISO-26262 includes specifications for MBD and provides recommendations for using fault injection techniques for hardware integration and testing, software unit testing, software integration testing, hardware-software integration testing, system integration testing and vehicle integration testing, for overview on fault injection recommendations in ISO-26262 see [191]. Although the functional safety standard specifies clearly the recommendations for using fault injection during various stages of testing but it does not recommend anything with respect to using mutation testing. This also reflects the current standard practice within the automotive industry where mutation testing is not widely adopted yet.

6.2.3 Fault Injection

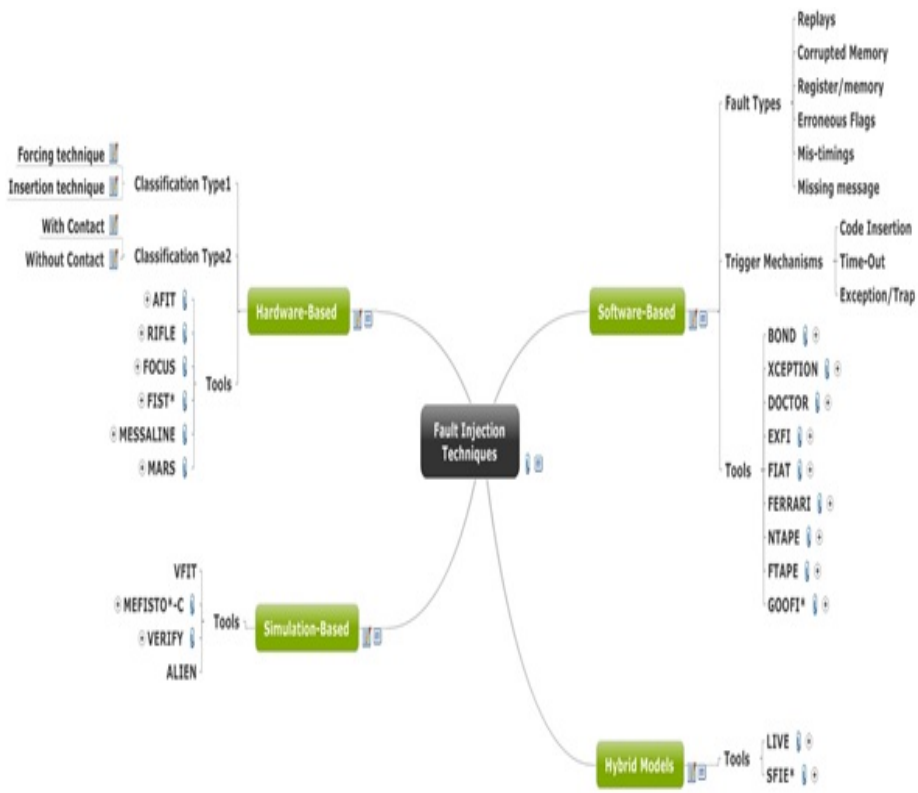

Figure 70: Common classification of fault injection techniques and implementation tools, description available in (Ziade et al., 2004, Hsueh et al., 1997)

Fault injection techniques are widely used for experimental dependability evaluation. Although these techniques have been used more widely for assessing the hardware/prototypes, the techniques are now about to be applied at behavioural models of software systems [192] - thus enabling early verification of intended functionality as well as enhancing communication between different stakeholders. Fault injection techniques applied at models level offer distinct advantages especially in an industry using MBD for its software development, but use of these techniques at model level in automotive industry is currently at its infancy. Figure 70 shows a mind map of classification of fault injection techniques based on how the technique is implemented; some of the tools which are developed based on given approach are also listed for reference. For a good overview of fault injection techniques readers are referred to [187], [193].

6.2.4 Mutation Testing

Mutation testing is technique for assessing the adequacy of given test suite. Mutation testing includes injection of systematic, repeatable seeding of faults in large number thus generating number of copies of original software artefacts with artificial fault infestation (called a mutant). Percentage of mutations detected by the given test cases/suite is a metrics (called "mutation adequacy score" [194]) used for measuring effectiveness of the given test suite. The variants of code (faults) can be introduced by hand or auto-generated using tools like Insure++, Plextest, Certitude, ESPT for C/C++ codes. It has been shown that the use of mutants yields trustworthy results [195], i.e. mutants do reflect characteristics of real faults.

Mutation theory is based on two fundamental hypotheses namely Competent Programmer Hypothesis and the Coupling Effect, both introduced by [196]. The Competent Programmer hypothesis reflects the assumption that programmers are competent in their job and thus would develop programme close to correct version (although making a number of mistakes) while the Coupling Effect hypothesis means that complex mutants are coupled to simple mutants in such a way that a test data that detects large percent of simple faults is also effective in detecting high percentage of the complex defects" [197].

6.3 Related Work

A number of European Union sponsored projects, within the area of embedded software development and safety critical systems have looked at and developed techniques to effectively use fault injection for safe and reliable software development. The examples include the ESACS (Enhanced Safety Assessment for Complex Systems) [198] and the ISAAC [199](Improvement of Safety Activities on Aeronautical Complex systems). These projects have used the SCADE (Safety-

Critical Application Development Environment) modelling environment to simulate hardware failure scenarios to identify fault combinations that lead to safety case violations.

A model-implemented fault injection plug-in to SCADE called FISCADE is introduced in [200]. The plug-in tool utilizes approach similar to mutation based testing, where it replaces the original model operators by their equivalent fault injection nodes. The derived models are then used to inject the fault during execution and log the results which are analysed later. Dependability evaluation of automotive functions using model based software implemented fault injection techniques have also been studied in [201].

A generic tool capable of injecting various types of faults on the behavioural/functional Simulink models is also developed and introduced in [192]. The tool called MODIFI (or MODel-Implemented Fault Injection tool) can be used to inject single or multiple point faults on behavioural models, which can be used to study the effectiveness/properties of fault tolerant system and identify the faults leading to failure by studying the fault propagation properties of the models.

Another work [202] with its root in the European CESAR (Cost-efficient methods and processes for safety relevant embedded systems) project provides a good theoretical overview of how fault and mutation based test coverage can be used for automated test case generation for Simulink models. We provide a practical framework on how fault injection combined with mutation testing within an MDB environment can be used in the industry. And how will this practice enhance the verification and validation of software under development, its functional validation that would generates statistics for the effective argumentation of ISO 26262 compliance.

6.4 Framework for Early Verification and Validation According to ISO 26262

We contend that fault injection can be effectively used at the model level to verify and validate the attainment or violation of safety goals. We also propose that it should be complemented with mutation testing approach at the model level to provide enough statistical evidence for argumenting the fulfilment of safety goals as per the ISO-26262 safety standard requirements.

A major challenge in successful argumentation of ISO-26262 compliance is to provide statistical evidence that safety goals (SGs) would not be violated during

operation and collecting the evidence for this argumentation within reasonable testing efforts.

If we are able to differentiate early between defects that can cause the violation of SGs and those that cannot cause the violation, the amount of testing required will be manageable. With MBD the functional testing could be done using fault injection techniques and this can be complemented with later system testing of the actual code using the mutation testing approach.

The framework on how this could be achieved in practice is as follows:

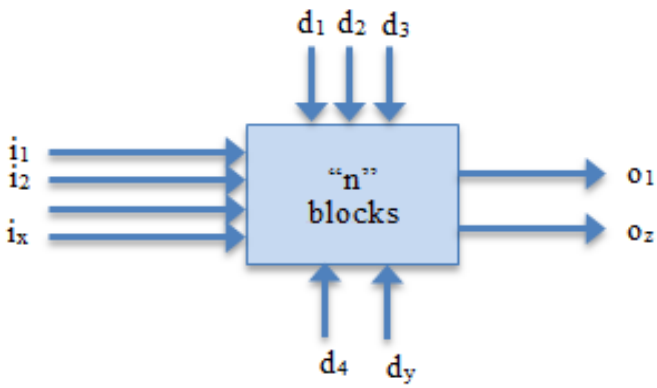

Figure 71: MBD based representation of a general system with inputs, outputs and dependencies

As illustrated in Figure 71, a given system/function generally have following common features (in context of model based development): firstly it will have x inputs ($i_{1,2...x}$); it would have dependencies to other y components/ functions ($d_{1,2...y}$); it will have z outputs ($o_{1,2...z}$); and it will have a number of sub-units/modules within it that implement the intended functionality, let us assume that this part contains n basic blocks in the modelling environment corresponding to n statements for a hand written code. To verify and validate the correct functionality and ISO-26262 compliance of this generic function using fault and mutation testing approach we can follow the steps as:

- Assign or define the technical safety requirements (TSRs) corresponding to the functional safety requirements (FSRs) for the given system/function to its z outputs.
- Use fault injection techniques to inject faults which are similar to commonly occurring defects and other possible fault conditions at the x inputs of the function.

- Fault scenarios that leads to violation of TSRs/FSRs are identified, statistics are built on what percentage of total faults lead to such failures and fault propagation properties of such cases are studied to build the fault tolerance within the system for given fault conditions.
- Repeat steps (b) & (c) to test, correct and validate the given system/function for its dependencies on other functions/components.
- Cause mutations to the n basic blocks of given functional model and asses the detection effectiveness of test suite/cases for possible implementation bugs.
- Examine the mutants which are not killed by given set of test cases/suits for their effect on FSRs. If a given mutation violates the FSRs then a suitable test case is created to detect/kill such mutants, i.e. detect such bugs in actual code.

By following the above mentioned steps we not only ensure that the given function holds the FSRs and TSRs under faulty inputs, but we can also prevent potential implementation defects and ensure that we have test cases ready to catch such faults that can potentially violate the FSRs/TSRs already at the design (model) level.

It is also worthwhile to note here that steps (a) to (e) can be easily automated using the currently available testing methodologies, which makes the usability and industrial viability much higher that testing frameworks requiring high manual interventions.

Further to make this framework/approach more effective in industrial practice we identify a number of best practices that will have positive impact on detecting defects early in the development process and thus have effective V&V of ISO-26262:

- The best practice is to build and maintain models corresponding to each abstraction layer of software architecture.
- The next best practice is to specify and test these models for FSRs and TSR at the appropriate abstraction level.
- Also identification of different types of defects/faults and at what stage they could be modelled/injected in the behavioural models would ensure that models are tested for these faults at the earliest - leading to models being build that are robust right from the start instead of adding fault tolerance properties in the later stages of development.

6.5 Case Study: Validation

In this section we present the validation of proposed framework on a set of components for self-driving miniature vehicles. The software for the miniature vehicles is build using similar methods and tools as professional software in the automotive industry, although on a smaller scale. In the validation we use the self-parking function of a self-driving miniature vehicle [203]. The architecture of the software is described in detail in [204] and one of our miniature vehicles using the self-driving vehicle software and a scenario for a sideways parking realized in our simulation environment are illustrated in Figure 72 & Figure 73. The miniature vehicles are in the scale 1:10 compared to the normal cars.

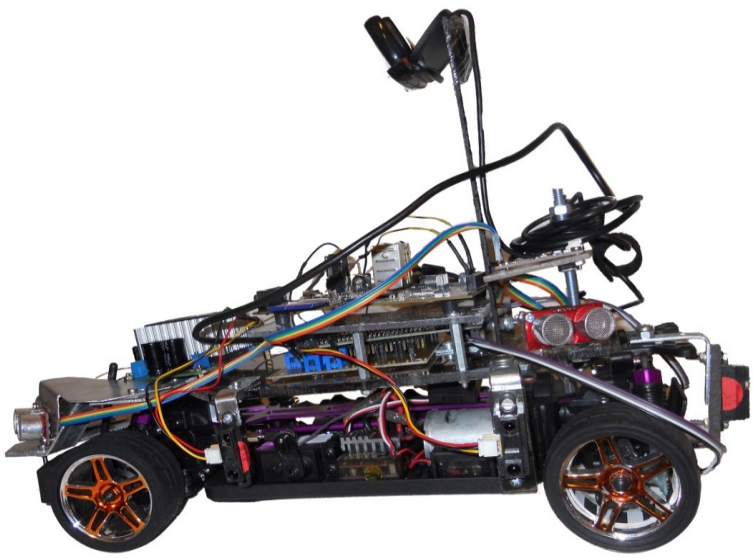

Figure 72: Self-driving miniature vehicle [204]

For understanding the initial validation of this framework it is sufficient to note that the functionality we are dealing with is self-parking for on a sideways parking strip. The self-parking algorithm expects a gap size of at least 7m to park in one turn without using an additional correction trajectory. This scenario is presented in Figure 8.

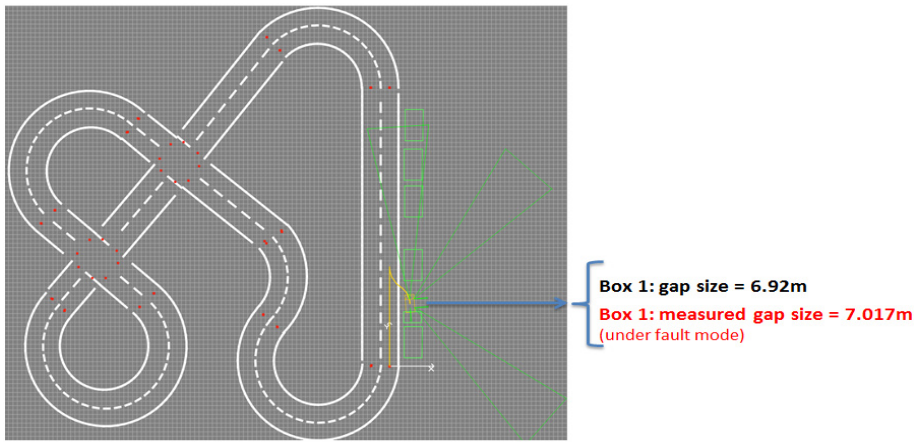

Figure 73: Test track for the experiment with parking gap from our simulation environment

We applied the framework for early verification and validation following the steps given in section 4 as follows:

- Assign FSR/TSR: An example of obvious functional safety requirement (FSR) for self-parking functionality is parking without hitting any other object. The corresponding technical safety requirement (TSR) can thus be parking only when gap size exceeds 7m (minimum gap size requirement).
- Using fault injection to simulate common fault scenario: A fault scenario is created by injecting a fault in the returned value for the travelled path by adding an error value of maximum 3.4% for the relatively travelled path increment. Thus, the size for measured gaps (due to faulty sensor input) increases for example by ~9.7cm to 7.01678m.
- Identify fault scenarios leading to FSR/TSR violations: Since in the experiment with fault injection, the parking algorithm depends on the travelled path; thus the algorithm parks the car in the lower gap which leads to a safety case violation because the cars collides with the obstacle at the rear side.
- Repeat steps (b) & (c) for all inputs: For this experiment, we focused on the fault injection for a single signal.
- Cause mutations: Single point mutations are caused by changing the logical operators in the self-parking function code, the standard test protocol to test the expected functionality was then applied to evaluate the generated mutants.

- Examine mutants & create new test cases: The mutants and the results whether they were successfully detected are provided in Lessons learned
- The initial validation experiment presented in this section for the proposed framework is the first step towards a complete validation of this framework in an industrial setting. Although the framework is focused on using fault injection and mutation testing at functional model level in model-based development to shift some of the verification and validation efforts to early stages of development, the example here demonstrated its applicability of given framework in a code-centric development environment as well.

The experiments using the software of a miniature vehicle provided a proof-of-concept for the framework and provide a frame of reference with respect to possible effectiveness. While in full scale safety evaluations following the ISO 26262, a given function depending on its functionality may be subjected to tens of safety goals and even a larger number of corresponding FSR/TSRs, we only evaluated one such scenario. Still with only a single fault scenario, we were able to identify faults leading to safety case violation. Also the mutation approach applied to this exemplary scenario by using 24 mutations, two out of these 24 mutants produced unexpected results and exposed the deficiency of the current test protocol, which was considered as adequate for the given functionality.

Therefore while these are encouraging results pointing towards applicability and effectiveness of the proposed framework, we also learned that we need further validation on industrial scale projects to increase the external validity of these results. Further for this framework to be successful in any organization much of the steps of described framework will have to be automated and supported by appropriate tools. As explained in section 2 & 3, a number of tools for fault injection and mutation testing based approaches are available for code-centric development making this framework practical for implementation on large scale with high automation. But corresponding tools to support fault injection and mutation based testing at functional model level in model-based development are not widely available and the few tools currently available are in their early stages of development where reliability of such tools will be an issue at least for some time in near future.

- Table 40. In this simple case itself with only 24 mutations, to our surprise two mutations produced unexpected results and violated the assigned FSR. While previously the test protocol has been deemed being sufficient for this function, the experiment clearly demonstrated the need for adding further test cases to reliably spot

these failures and to detect possible faults leading to FSR violations.

6.5.1 Lessons learned

The initial validation experiment presented in this section for the proposed framework is the first step towards a complete validation of this framework in an industrial setting. Although the framework is focused on using fault injection and mutation testing at functional model level in model-based development to shift some of the verification and validation efforts to early stages of development, the example here demonstrated its applicability of given framework in a code-centric development environment as well.

The experiments using the software of a miniature vehicle provided a proof-of-concept for the framework and provide a frame of reference with respect to possible effectiveness. While in full scale safety evaluations following the ISO 26262, a given function depending on its functionality may be subjected to tens of safety goals and even a larger number of corresponding FSR/TSRs, we only evaluated one such scenario. Still with only a single fault scenario, we were able to identify faults leading to safety case violation. Also the mutation approach applied to this exemplary scenario by using 24 mutations, two out of these 24 mutants produced unexpected results and exposed the deficiency of the current test protocol, which was considered as adequate for the given functionality.

Therefore while these are encouraging results pointing towards applicability and effectiveness of the proposed framework, we also learned that we need further validation on industrial scale projects to increase the external validity of these results. Further for this framework to be successful in any organization much of the steps of described framework will have to be automated and supported by appropriate tools. As explained in section 2 & 3, a number of tools for fault injection and mutation testing based approaches are available for code-centric development making this framework practical for implementation on large scale with high automation. But corresponding tools to support fault injection and mutation based testing at functional model level in model-based development are not widely available and the few tools currently available are in their early stages of development where reliability of such tools will be an issue at least for some time in near future.

Table 40: Mutation testing output, case with and without fault mode scenario

ID	Mutant	Change description	Test case result using regular vehicle simulation	Test case result using vehicle simulation with fault injection
1	Unmodified	Original self-parking algorithm	Passed as expected	Failed as not expected, vehicle took first gap, collided (expected: robust algorithm dealing with varying travelled distance data)
2	68	Changed == to >	Failed as expected, vehicle did not start	Failed as expected, vehicle did not start
3	73	Changed first > to ==	Failed as expected, vehicle started hardly noticeable (v < 0.009m/s)	Failed as expected, vehicle started hardly noticeable (v < 0.009m/s)
4	73	Changed && to ||	Failed as expected, vehicle moved forwards slightly and pull back while	Failed as expected, vehicle moved forwards slightly and pull back while turning to left
5	73	Changed second < to >	Failed as expected, vehicle moved to the end of the second parking spot but did	Failed as expected, vehicle moved to the end of the first parking spot but did not start
6	79	Changed first >= to <=	Failed as expected, vehicle did not start	Failed as expected, vehicle did not start
7	79	Changed && to ||	Failed as expected, vehicle moved backwards while turning to left	Failed as expected, vehicle moved backwards while turning to left
8	73	Changed second < to >	Failed as expected, vehicle moved to the end of the second parking spot but did	Failed as expected, vehicle moved to the end of the first parking spot but did not start
9	85	Changed first >= to <=	Failed as expected, vehicle moved backwards while turning to right	Failed as expected, vehicle moved backwards while turning to right
10	85	Changed && to ||	Failed as expected, vehicle moved backwards while turning first to right and then to left (S-shaped)	Failed as expected, vehicle moved backwards while turning first to right and then to left (S-shaped)
11	85	Changed second < to >	Failed as expected, vehicle moved to the end of the second parking spot but did	Failed as expected, vehicle moved to the end of the first parking spot but did not start
12	91	Changed first >= to <=	Failed as expected, vehicle moved backwards while turning to left	Failed as expected, vehicle moved backwards while turning to left
13	91	Changed && to ||	Failed as expected, vehicle moved backwards while turning to left	Failed as expected, vehicle moved backwards while turning to left
14	91	Changed second < to >	Failed as expected, vehicle moved to the end of the second parking spot, started parking, but stopped after the first right	Failed as expected, vehicle moved to the end of the first parking spot, started parking, but stopped after the first right turn
15	97	Changed >= to <=	Failed as expected, vehicle did not start	Failed as expected, vehicle did not start
16	115	Changed first > to <	Failed as expected, vehicle did not find the parking stop and continues driving	Passed as not expected, vehicle parked in the second parking spot because the noise added to the travelled distance resulted in a valid parking gap size
17	115	Changed && to ||	Failed as expected, stopped before the first parking gap, collided with parked	Failed as expected, stopped before the first parking gap, collided with parked car
18	115	Changed second > to <	Failed as expected, stopped before the first parking gap, collided with parked	Failed as expected, stopped before the first parking gap, collided with parked car
19	126	Changed first > to <	Failed as expected, vehicle took first gap, collided	Failed as expected, vehicle took first gap, collided
20	126	Changed && to ||	Failed as expected, vehicle did not find the parking stop and continues driving	Failed as expected, vehicle did not find the parking stop and continues driving
21	126	Changed second > to <	Failed as expected, vehicle did not find the parking stop and continues driving	Failed as expected, vehicle did not find the parking stop and continues driving
22	135	Changed first > to <	Failed as expected, vehicle did not find the parking stop and continues driving	Failed as expected, vehicle did not find the parking stop and continues driving
23	135	Changed && to ||	Failed as expected, stopped before the first parking gap, collided with parked	Failed as expected, stopped before the first parking gap, collided with parked car
24	135	Changed second > to <	Failed as expected, stopped before the first parking gap, collided with parked	Failed as expected, stopped before the first parking gap, collided with parked car

6.6 Conclusions

The development of software in the automotive domain has widely adopted the paradigm of model based development to allow for easier integration of functionality usually developed by multiple suppliers. By the nature of the domain much of the functionality developed and implemented in cars is safety critical; the criticality that requires observation of stringent quality assessment and adherence to functional safety standards such as ISO 26262.

Development of behavioural models in MBD offers significant opportunity to do functional testing early in the development process. Fault injection and mutation testing approach in combination can be used to effectively verify and validate the functional properties of a software system/function. The approach also provides required statistics for the argumentation of safety standards compliance. In this paper the need for such validation and a framework on how this could be achieved in practice is discussed. The results are a roadmap for further research and tool support to bring this approach into wider industrial adoption.

Initial validation of our proposed framework provided a proof-of-concept and produced encouraging results indicating its usefulness and effectiveness in practice. It is also noted that the framework will become much more effective and easy to use for model-based development as tools related to fault injection and mutation testing at model level matures over time. In the meantime, validation on industrial scale functions will provide further evidence to evaluate the applicability and effectiveness of the proposed framework in practice.

By detecting defects early and being able to do much of verification and validation of intended functionality, robustness and compliance to safety standards on the models – the quality and reliability of software in automotive domain can be significantly enhanced. Effective approaches and tools support reduce the V&V costs and lead to shorter development times.

Chapter 7:
Machine learning techniques for software defect prediction in industry - adoption

Included Publications:

IX. R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, and W. Meding, ***"A framework for adoption of machine learning in industry for software defect prediction"***, In the proceedings of 9th International Joint Conference on Software Technologies, ICSOFT-EA, Vienna, Austria, 2014

X. R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, and W. Meding, ***"The adoption of machine learning techniques for software defect prediction: An initial industrial validation"***, In the proceedings of 11th Joint Conference On Knowledge-Based Software Engineering, JCKBSE, Volgograd, Russia, 2014

7 A FRAMEWORK FOR ADOPTION OF MACHINE LEARNING IN INDUSTRY FOR SOFTWARE DEFECT PREDICTION

Abstract— Machine learning algorithms are increasingly being used in a variety of application domains including software engineering. While their practical value have been outlined, demonstrated and highlighted in number of existing studies, their adoption in industry is still not widespread. The evaluations of machine learning algorithms in literature seem to focus on few attributes and mainly on predictive accuracy. On the other hand the decision space for adoption or acceptance of machine learning algorithms in industry encompasses much more factors. Companies looking to adopt such techniques want to know where such algorithms are most useful, if the new methods are reliable and cost effective. Further questions such as how much would it cost to setup, run and maintain systems based on such techniques are currently not fully investigated in the industry or in academia leading to difficulties in assessing the business case for adoption of these techniques in industry. In this paper we argue for the need of framework for adoption of machine learning in industry. We develop a framework for factors and attributes that contribute towards the decision of adoption of machine learning techniques in industry for the purpose of software defect predictions. The framework is developed in close collaboration within industry and thus provides useful insight for industry itself, academia and suppliers of tools and services.

Keywords— Machine Learning, software defect prediction, technology acceptance, adoption, software quality

Acronyms Used— ML: Machine Learning; SDP: Software Defect Prediction; TAM: Technology Acceptance Model

7.1 Introduction

Testing is an essential activity in software engineering [205], but also one of the most expensive phase within software development life cycle with some estimates approximating it to consume about 50% of time and resources [206]. Software Defect Prediction (SDP) offers one possible way to make software testing more effective by making it possible to optimize test resource allocation, i.e. distributing more effort to parts (files/modules) that are predicted to be more prone to defects. The importance of such predictions is further substantiated by previous research suggesting applicability of 80:20 rule to software defects (that is approximately 20% of software files are responsible for 80% of errors and cost of rework) [207] [162].

Different methods for defect prediction have been evaluated and used; these can broadly be classified as traditional (using expert opinions and regression based approaches) and those based on machine learning techniques. Methods based on machine learning offer addition advantage with their ability to improve their performance through experience (as more data is made available over time). Despite the importance of predicting defects in a software project and demonstrations that SDP using ML techniques is not too difficult to apply in practice [208], their adoption and application by practitioners in industry has been limited which is apparent from the lack of published experience reports. Adoption of any complex method/technology is dependent on several dimensions [209], but most of the earlier studies in SDP have focused mainly on the aspect of predictive accuracy. In this paper we argue that our lack of understanding of other factors relevant to industrial practitioners is a major reason for low adoption of ML techniques for SDP in industry.

Based on the technology acceptance model (TAM) and technology adoption frameworks we develop a framework for explaining the adoption of ML for SDP in industry. TAM intends to explain why users' belief and their attitudes towards a technology affect their acceptance or rejection of the information-communication technology. While TAM is parsimonious and theoretically justified model to explain information technology adoption [210], to use this model for a specific technology requires identification of detailed attributes specific to the given technology and context which collectively explain the belief and attitude of uses towards the given technology. The research question we address in this paper is:

"How can we use the technology acceptance and adoption models for developing framework for ML adoption in industry and how to adapt it for software defect prediction?"

7.2 Background and Related work

7.2.1 Software defect prediction using tradition approaches

Traditional methods used for software defect prediction and risk assessment can be broadly categorized under:

- Expert Opinions
- Analogy Based Predictions
- Regression Based Approaches

Statistical approaches based on regression have also been used for the task of defect prediction. The dependent (or outcome) variable could be binary (defective or not defective) as in logistic regression or the model could be built to predict the number of expected defects as in case of multiple linear regression. Logistic regression has been applied in Khoshgoftaar and Allen [29] for classifying modules as fault-prone or not. Zimmermann, Premraj and Zeller [30] also applied Logistic regression to classify file/packages in Eclipse project as defect prone (has defect Vs. not has defect) . Multiple linear regression is used to model software changes [23] as a function of a set of software complexity metrics. Linear regression was also used by Khoshgoftaar et al. [24] for predicting program faults in two subsystems of a general-purpose operating system, where they also evaluated different fitting criteria's (namely Least Squares, Least Absolute Value, Relative Least Squares and Minimum Relative Error).

7.2.2 Software defect prediction using ML techniques

Broad types of Machine Learning (ML) techniques used for software defect prediction:

- Decision Trees (DTs)
- Support Vector Machines (SVMs)
- Artificial Neural Networks (ANNs)
- Bayesian Belief Networks (BNNs)

Machine learning algorithms can also be used to model the software defect prediction as a classification problem as in case of DTs and SVMs where the class variable can take two values (defective or not defective). Or the problem can be modelled to predict expected number of defects in a software module/system using different code and change metrics. ML techniques for pattern recognition for e.g. ANNs and BNNs can be used to accomplish such tasks.

Number of various classification models including DTs and SVMs have been evaluated and compared in [12]. Iker Gondra [32] applied machine learning algorithms to predict the fault proneness and compared between the ANNs and SVMs and found that if fault proneness is modelled as classification task, SVMs performs better than the ANNs.

Table 41 provides an overview of some of the important ML techniques that can be applied for SDP and lists their main advantages and limitations. For details on ML techniques applicable in software engineering domain; readers are referred to work by Zhang and Tsai [211].

Table 41: Overview of ML techniques used for software defect prediction

Algorithm Type	**DTs**
Domain Knowledge	Not Required
Training Data	Adequate data needed to avoid over-fitting.
Advantages	Robust to noisy data; Missing values tolerated; Capable of learning disjunctive expressions.
Disadvantages	Prone to over-fitting.
Algorithm Type	**SVMs**
Domain knowledge	Not Required
Training Data	Adequate data needed for training.
Advantages	Effective for high dimensional spaces, is memory efficient and is versatile as it can take different kernel functions as decision function
Disadvantages	SVMs are likely to give low performance if number of features is much higher than the number of samples
Algorithm Type	**ANNs**
Domain knowledge	Not Required
Training Data	Adequate data needed for training.
Advantages	Able to learn non-linear and complex functions; Robust to errors in training data.
Disadvantages	Slow training and convergent process; Prone to over-fitting; Results difficult to interpret.
Algorithm Type	**BNNs**
Domain Knowledge	Not Required
Training Data	Required for estimate the prior probabilities.
Advantages	Able to give probabilistic predictions; Useful for knowledge discovery; Can be used very early in the development lifecycle
Disadvantages	Requires estimation of many prior probabilities that can be very large for big models; computationally expensive; requires domain expertise for building the network.

7.2.3 Technology Adoption Framework

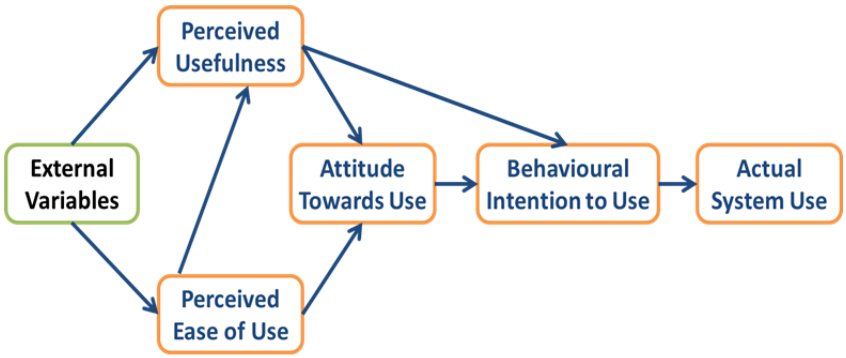

Figure 74: Overview of Original Technology Acceptance Model [209]

According to Attewell [212] adoption of complex technology is not an event, but resembles knowledge acquisition over time, the perspective is applicable where new innovation/technique is [212]:

- Abstract and have demanding scientific base,
- Fragile in sense of consistency, i.e. do not always perform as expected,
- Difficult to try in a meaningful way, and
- Unpackaged, i.e. adopters cannot pick a tool out of shelve and use it as a black box model, but instead need to acquire broad tacit knowledge and procedural know-how.

Characteristics of ML based techniques fits well to most above point and thus can be classed as complex technology/techniques. Further according to the Theory of Reasoned Action (TRA) [213], the intention of adoption of behaviour or technology is based on the beliefs about the consequences of adoption. The theory have been used to build Technology Acceptance Model (TAM) by Davis [41], an overview of model is presented in Figure 74. TAM postulates that a users' adoption intention and the actual usage of information technology is determined by two critical factors, the perceived usefulness and perceived ease of use. Perceived usefulness is defined as the degree to which a user believes that using a particular system would enhance his/her job performance, while perceived ease of use is the degree to which the user believes that using the system would be effort free [210].

In this study we are focused on technology adoption decisions, thus the model we use for our framework is based on the revised version of original TAM model [214], the postulation of revised model is that potential users of a technology actively

evaluate the usefulness and ease of use of given technology in their decision making process [215]. Our position in this paper is similar:

We contend that applying technology adoption framework to ML techniques use in SDP is needed to better understand the needs of industry - which will help accelerate the technology transfer and adoption process of these techniques.

Technology adoption framework by Tornatzky et al. [42] also provide a model of adoption that has been applied widely. According to the framework, there are three elements which influence the innovation adoption process:

1. The external environmental context,
2. The technological context, and
3. The organizational context.

Chau and Tam [216] used the framework to model the factors affecting adoption of open systems in the Information Science (IS). We adapt their framework in conjunction with the Technology Acceptance Model (TAM) to model the factors affecting adoption of ML in industry.

7.3 Study Design

The research process for development and quantitative validation of adoption framework for ML techniques in industry is shown in Figure 75. The focus of this paper is Stage-1, where the centre of attention has been to develop the general adoption framework for machine learning techniques and demonstrate how the model can be adapted for the specific case of software defect prediction (SDP).

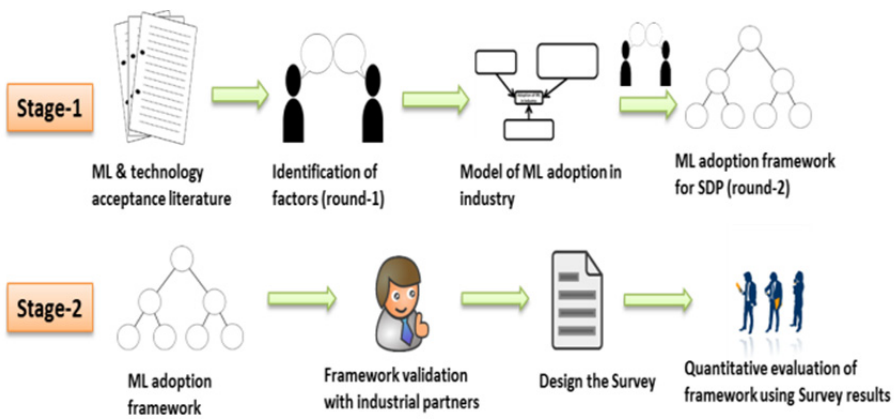

Figure 75: Research process overview

Literature Review: To capture the factors that affect the adoption of ML techniques in industry we searched for likely factors mentioned in software engineering, machine learning and technology adoption literature. A list of factors deemed potentially relevant for industry was compiled which was used for discussions with the industrial practitioners. The application area we concentrated on is defect prediction in software system/projects.

Interviews: Semi-structured interviews were conducted with industrial practitioners to first evaluate which factors are relevant for ML adoption in industry. In the next round the same interviewees helped adapt this general model for the case of software defect prediction.

In total four managers from two large companies with significant focus on software development were interviewed consequently in two rounds. The companies included in the study are:

- **Volvo Car Group (VCG):** A company from the automotive domain, and
- **Ericsson:** A company from the telecom domain

The divisions we interacted with have one thing in common, they have not yet adopted machine learning as their main method/technique for predicting software defects, but they are evaluating it as a possible technique to compliment the current software defect measurement/prediction systems in place. The interviewees included,

- Manager at Volvo Cars Group within the department responsible for integrating software sourced from different teams and suppliers, the manager has more than 20 years of experience working with software development and testing. Ensuring safety and quality of software developed is a major responsibility in this job role.
- Team leader at Volvo Car Group responsible for collection, analysis and reporting of project status with regard to software defects and their predictions, the team leader has more than three decades of experience in various roles at the company.
- A senior quality manager at Ericsson whose experience with software (mainly within quality assurance) spans more than three decades, and
- Team leader of metrics team at Ericsson; metrics team is a unit at Ericsson that provides the measurement systems for various purposes including software defect measurement, monitoring and prediction systems within the organization.

The main focus in the first round of interviews is to identify the factors relevant with regard to technology adoption/acceptance decisions (to build a general framework of ML adoption in industry). While the second round of interviews were focused on identification of relevant attributes for each factor in the specific context of software defect prediction.

7.4 Framework for adoption of ML techniques in industry

It is important to note that for any organization at any given point in time, the trade-off analysis is not between adopting or not adopting a new technology/process (as in case of ML techniques); the trade-off is between adopting it now or deferring that decision until a later date. This distinction is important as the factors that affect the adoption are not only specifically related to direct advantages and limitation of given technology/process, but also organizational and environmental at a given point in time. In this context, nine important factors that affect the adoption of ML techniques were identified; these can be grouped into three categories according to the framework by Tornatzky [42]. The framework for adoption of ML in industry is presented in Figure 76.

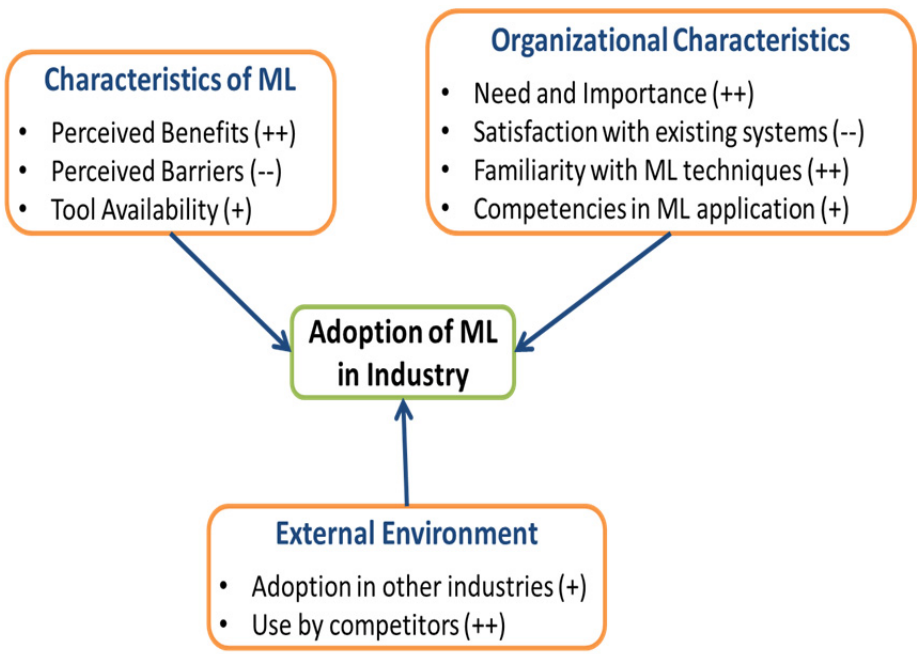

Figure 76: A Model for ML adoption in Industry

In Fig 3 (+) and (-) signs denote the possibility of positive/negative relationship with medium strength between a given factor and probability of adoption of ML. A double (++/--) indicate a strong relationship; the strength of relationship can be tested by setting a stricter significance level during quantitative evaluation (for e.g. alpha value of 0.1 for +/- and 0.05 for ++/--). Accordingly hypotheses for each factor can be formulated which can be tested quantitatively from a survey. We provide a couple of examples of null hypothesis that can be quantitatively tested:

H1: *Higher levels of perceived benefits of adopting ML techniques will strongly (and positively) affect the likelihood of their adoption.*

H2: *Higher levels of perceived barriers of adopting ML techniques will strongly (and negatively) affect the likelihood of their adoption.*

7.5 Adaptation of ML adoption framework for SDP

We adapt the general framework for ML adoption in industry (Fig 3) to the specific problem of software defect prediction.

7.5.1 Characteristics of machine learning

Adoption of any new technology or process change is heavily dependent upon the characteristics of technology/innovation. Factors affecting cost-benefit trade-off of adoption are some of the critical factors in decisions of adoption. The relevant attributes that affect the acceptance of ML for software defect predictions are presented in Figure 77.

Perceived benefits: one of the most critical factors in adopting ML techniques in industry are the perceived benefits of these techniques for a given organizations specific context. The keywords here are perceived and context. While the actual benefits, an organization can achieve by adopting a new innovation/technology is important in long run, at a given point in time what affects an organizations decision to adopt a new specific technology/innovation is its perception.

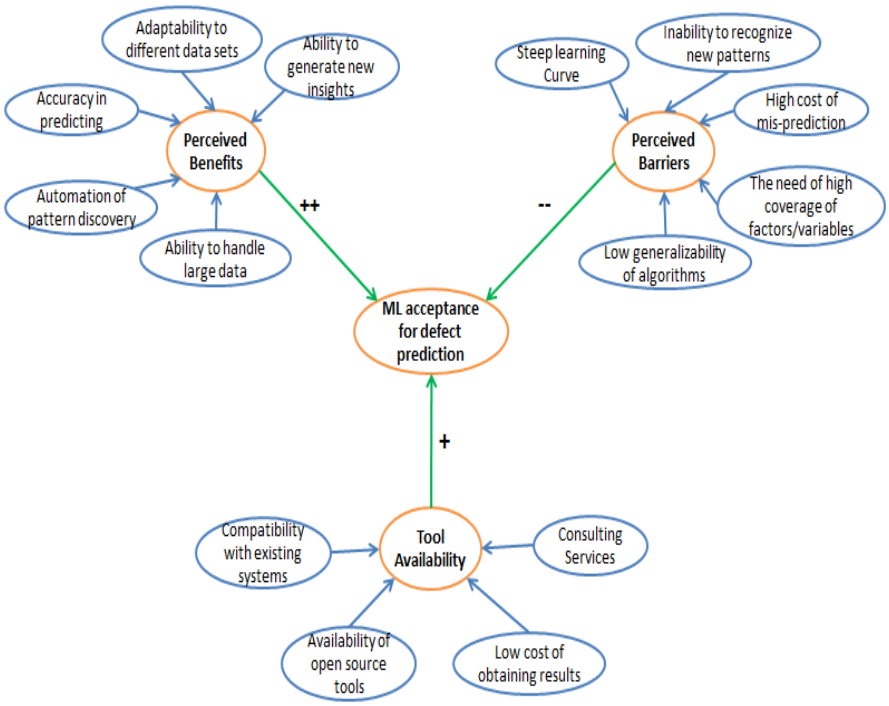

Figure 77: Overview of attributes relevant to ML characteristics that affects its acceptance for SDP

When it comes to SDP, the perceived benefits of using ML approaches as expressed in previous studies evaluating ML techniques for SDP and opinions expressed by the interviewees of this study are ability of ML based algorithms to:

- Provide higher prediction accuracy (high probability of detection and low probability of false alarm) [32].
- Be highly automated, i.e. most aspects of system including data collection to visualization of results can be done using smart algorithms mining and analysing data autonomously from the multiple local databases [217] with minimal human intervention.
- It is perceived that ML techniques can handle large data; in fact ML methods are expected to improve their performance as more data is made available over time [211].
- Another important expectation with techniques applied to predicting software defects is that these techniques are capable of identifying new patterns in data thus providing new insights from the data itself. This offers possibility to use large historical data to discover regularities and use them to improve future decisions [218]. New insights can be generated using large data by employing

specific ML techniques such as causal modelling for example by using Bayesian Networks to model causal networks and deduct probabilistic relationships.
- Given the self-adaptive nature, using ML techniques is also perceived to be low on maintenance activities.

Perceived barriers: On the other hand perceived barriers negatively affect the adoption/acceptance of ML techniques. For software defect predictions, some of the common perceived barriers are:

- Steep learning curve – According to Edmondson et al. [219], users of new innovation/technology need to understand it well before they can put it into productive use. Their study also suggests that when tacit knowledge is needed, new technologies may fail in market even when their advantages have been proven.
- For example in case of SDP, when using classification or pattern recognition, selecting the set of attributes (inputs) that give optimal results is very much based on domain experience and experience of using ML based techniques which is difficult to document/codify explicitly for new users.
- Lack of trust – stakeholders in software projects who are used to traditional approaches of predicting defects (such as expert opinions) do not generally trust the algorithms to outperform expert based predictions.
- For software projects, in general and in particular for safety and business critical software products, the penalty for mis-prediction is an important barrier. The severity of mis-prediction is correlated to importance of information need and actions it can trigger. For example a prediction model that falsely predicts 20% of software modules as defect prone (compared to actual 10%) may lead to review of 10% modules which was unnecessary and results in resource allocation which is not optimal.
- As traditional methods have been used for comparatively longer time, their levels of (un)certainty are known – which is not the case with ML techniques. To overcome this barrier we recommend that in the initial phase of adoption of machine learning techniques, these should be using alongside the traditional methods to validate their usefulness and predictive accuracy in practice. This provides the comparisons industrial practitioners want to see before trust in new techniques begins to build up over such trial periods.
- Given that most practical aspects can be affected by wide range of factors; techniques based on ML approaches usually do not take

into account all of these. Human factors such as differences in productivity, people getting sick or motivation level of employees are hard to measure and account for in algorithmic models for SDP and thus a source of error in such techniques.

- Uncertainty regarding generalizability of ML over projects. The perception is that while ML techniques (used for classification and pattern recognition) work well in recognizing existing patterns in the data, but their performance degrades for patterns that are unseen before.

Availability of tool and support is expected to increase the acceptance of ML in industry [220]. Some of the attributes related to this factor are - if the available tools are open source or proprietary, how much support is available and how much they cost. Others include if the given tool is compatible with existing measurement systems and in-house competences with respect to its usage. Consulting services can also help specific companies to get started with new approaches that they do not have enough experience with - thus helping acceptance of new techniques and tools in industry.

A number of packages implementing ML algorithms are available for e.g. Netlab, Spider and BNT for Matlab; Nodelib, Torch for C++; and CREST for python. Commercial (e.g. Ayasdi, NeuroSolutions etc.) and open source tools (e.g. Weka, KNIME etc.) are also available with GUI. While availability of such tools is likely to increase the adoption of ML in industry, other attributes such as support and consulting services is also important in determining the level and speed with which ML is adopted in the industry.

One possible way of enhancing adoption through tool and support availability is by making available problem specific customized solutions for highly relevant industrial problems such as SDP. Other activities that can potentially accelerate the adoption process is integration of ML based algorithms in existing software packages widely used within industry, for e.g. Microsoft Neural Network algorithm available for SQL Server 2012.

7.5.2 Organizational characteristics

Need and importance: The higher the need and importance of given information is in an organization, the higher is the likelihood for adopting new techniques to satisfy this information need.

To improve on the accuracy and reliability for such measures, new approaches that offer higher accuracy and reliability are more likely to be adopted. Zhang and Tsai

[211] provides a good overview of applications of ML in software engineering domain which outlines different information needs within this domain. Examples of information need specific to software defect predictions are:

- Predicting software quality (identification of high-risk, or fault-prone components)
- Predicting software reliability
- Predicating expected number of defects
- Predicting maintenance task effort
- Predicting software release timings

Factors such as how satisfied a company is with its existing defect prediction systems, their familiarity with machine learning techniques and in-house competences are also important for explaining acceptance and adoption of ML for SDP within a company. A model of attributes that contribute to these factors is presented in Figure 78.

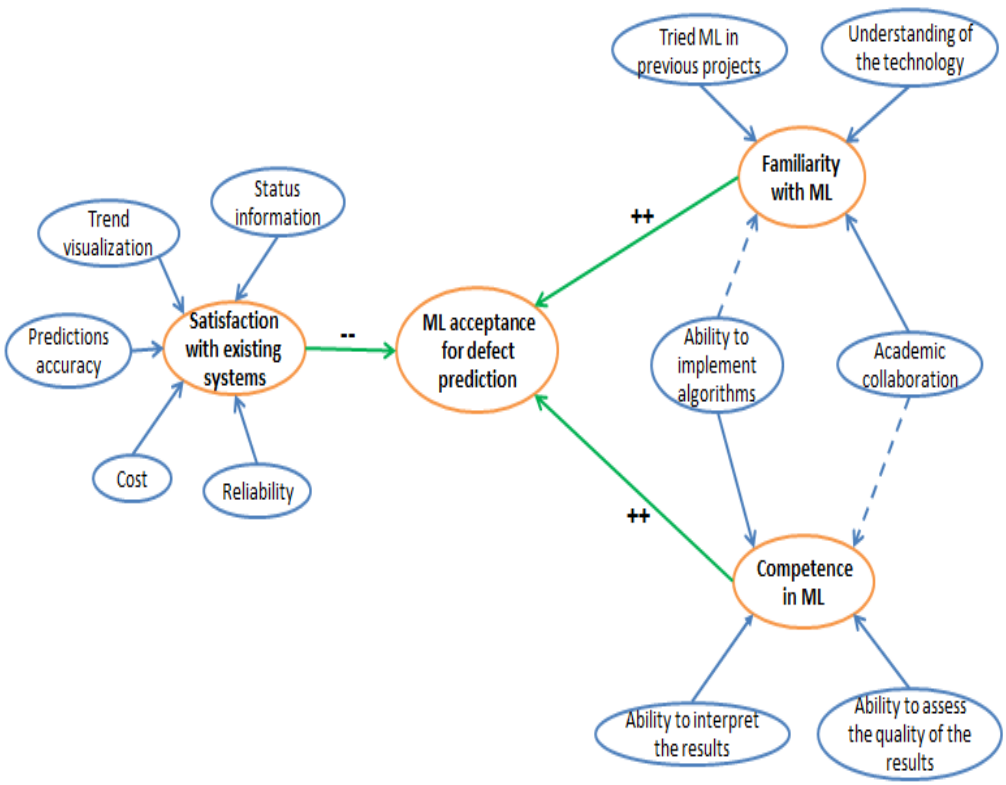

Figure 78: Overview of attributes relevant to organizational characteristics that affects its acceptance for SDP

Satisfaction with existing systems: the motivation for change (adoption of new approaches) is strongly connected to given organizations satisfaction with its current measurement/analysis systems. If a company is well satisfied with accuracy and efficiency of existing methods it is unlikely to invest significant amount of cost, resources and learning on new approaches. In case of software defect prediction, attributes relevant to satisfaction with existing systems are:

- If or not the existing system satisfies the information need of stakeholders involved in the project.
- Does existing system allow stakeholders to effectively and efficiently visualize the trend over time and let them compare current projects with similar historical projects data.

The reliability and cost also plays important role in determining the level of satisfaction with existing defect management and prediction systems within software development organizations.

Familiarity and competence with ML techniques: organizations familiar with approaches of machine learning though their workforce or collaborations with academia will have better understanding of advantages and limitations of such approaches. These organizations will also be more informed about practical applicability of these techniques and thus in a position where they can identify and assess areas where the benefits of using ML techniques outweigh the barriers – therefore organizations that are familiar with such methods are strongly likely to adopt these methods.

Attewell [212] proposes that "*firms delay in-house adoption of complex technology until they obtain sufficient technical know-how to implement and operate is successfully*"

Almost all mature organizations engaged in developing software generally collect, store and analyse their product and process related data. Given that such data is available in large quantities (within the organizations), an organization with good competences/skills in machine learning are more likely to try ML techniques on their data and eventually adopt it on larger scales.

The main challenge in this context is unavailability of structured data. Much of the data generated within an organization is in form of unstructured text (e.g. software requirements, defect reports, customer feedback written in textual form). On the other hand most ML algorithms require inputs in numeric or categorical form which presents challenge in using such data in practice. Developments in field of Natural

Language Processing (NLP) are already addressing these challenges and advances in such areas are likely to increase the adoption of ML based techniques for SDP.

7.5.3 External environment

ML techniques, if adopted in different industries signals their applicability in practice, although this is not expected to be a strong factor deriving adoption in other industries – it is likely to affect positively the probability of adoption.

A similar but stronger factor for adoption of new technology/approaches such as ML in a given company is likely to be the information whether or not any of the competing companies are using such techniques. The motivation behind this factor is simple - every organization in a given domain intends to be at the forefront of technology or process knowledge. The adoption of a particular technique/process by a competitor is a strong signal that given technique could have potential benefits; this can potentially motivate the need for evaluation of such methods within the given organization.

7.6 How to use the framework

Over the years companies have begun capturing huge volumes of data about their products, consumers and operations [218]. ML offers new tools that can use this data to recognize patterns and provide useful insights hidden within these huge volumes of data.

7.6.1 Setting the research direction

The research in software defect predictions has been mainly focused on evaluating and highlighting the predictive accuracy of ML techniques and in some cases comparing it to traditional methods. On the other hand the adoption framework indicates that not only predictive accuracy, but attributes such as cost, reliability and generalizability are also important for adoption decisions.

Therefore the technology adoption framework, such as one proposed here, can be useful to guide future research directions by helping to identify which factors are relevant for industrial adoption, but currently unaddressed in terms of their scientific evaluation.

7.6.2 Evaluating specific ML techniques by a given company

Technology acceptance/adoption frameworks enhance our understanding of which factors affect the end users decision to adopt a given technology/innovation.

Although these factors do play a role to varying degree when companies evaluate their decision to adopt or delay the adoption of such techniques, the lack of a framework can lead to sub-optimal decisions. Without a guiding framework there is high probability that effect of some detailed attributes that affect the overall usefulness is missed. The severity of problem is greater when comparisons are made between two or more techniques or tools where it is likely that evaluation would focus only on small set of attributes which does not provide the full picture.

Table 42: Example of how adoption framework can be used to compare between two new tools/services

Attribute	**Tool A**	**Tool B**
Predictive Accuracy	85%	82%
Auto data acquisition	Yes	Yes
Report generation	Yes, web based	Yes, multiple format
Can handle multiple projects	Yes	Yes
Generate causal maps	Yes, Non-Interactive	Yes, Interactive
Running time (typical project)	30min	40min
Cost of license (tool)	$ 20000/ license	$ 35000/ license
Maintenance cost (estimate)	$ 7000 pa	$ 9000 pa
…		

Table 43: Example of how comparative checklist can be used to evaluate new technique for SDP

Attribute	**Existing Method**	**New ML based technique**
Predictive Accuracy	Good	Very Good
Auto data acquisition	Yes	Yes
Report generation	Yes, word document	Yes, web based
Can handle multiple projects	No	Yes
Generate causal maps	No	Yes
Running time (typical project)	15min	30min
Cost of license (tool)	None	$ 20000/ license
Maintenance cost (estimate)	$ 2000 pa	$ 7000 pa
…		

In such cases, the adoption framework can be used as a guide so that all important factors and associated attributes are covered when considering adoption of new techniques or tools or even as a checklist to make such assessment and comparison between two or more techniques/tools using Likert-type scale for evaluation. To provide an example, Table 42 shows a checklist to compare a ML based technique against existing system for SDP and Table 43 show potential use of similar checklist for comparison of two competing tools. Industrial practitioners can use such

checklists to make informed decision with regard to adoption of these techniques and for effective comparison between tools.

The technology adoption framework also help companies to reflect upon their strengths with respect to given technology and areas of potential improvement. Such analysis is useful to identify areas where training and competence build-up would be advantageous. For example in SDP, if a company identifies that the in-house competence for implementing and maintaining ML based system would benefit a specific business unit within the organization, necessary training and or recruitment targeting those specific skills could be quickly arranged, thus improvising the long term competitiveness of the company.

7.6.3 Improvising the tool and services by vendors

Technology adoption framework is also useful for tool vendors who can use the information in multiple ways, to:

- Prioritize feature introduction, and
- Effective marketing of their tools and services

Tools based on emerging technologies/techniques usually provide new functionality not available in old well established tools, but at the same time they are not mature and need to constantly evolve to engage and acquire new customers. Understanding clearly which attributes are key for adoption decision help these tool vendors to prioritize the features they implement and deliver to their customers. For example, a vendor with Tool X for SDP which at a given time do not outperform existing tools on predictive accuracy; finds out that running and maintenance costs are important attributes in adoption decisions - may use this information to strategically decide to develop a light version of tool which demands low running and maintenance costs.

Understanding of which attributes play a key role in adoption decisions also help tool and service vendors to make their marketing more effective. Vendors may choose to highlight how they provide value to their customers on the key attributes industry is looking for when considering adopting a new technology based product or services. This accelerates the adoption and acceptance of new techniques within the industry.

7.7 Conclusions and future work

Large and constantly growing amount of data is now available within organizations that can be used for gaining useful insights to improvise process, products and services. Machine learning techniques have high potential to aid companies in this purpose. Despite demonstration of usefulness of such techniques in academia and

availability of tools, the adoption of these techniques in industry currently is far from optimal. Our position in this paper has been that for accelerating the adoption of ML based techniques in industry, we need to enhance our understanding of information needs of industry in this respect. Technology acceptance model offer cost effective approach to meet this purpose.

In this paper we developed a framework for the adoption of ML techniques in industry. The framework is developed with its basis on previous research on technology adoption and technology acceptance models. We also adapted the framework to the specific problem of software defect predictions and highlighted that while adoption decisions are multi-dimensional, current research studies have mainly focused on few of these attributes. We contend that elevating our understanding of factors and attributes relevant for industrial practitioners will help companies, researchers and tool vendors to meet the specific information needs.

In future work we plan to quantitatively evaluate the effect size of important attributes towards ML adoption decision using large scale survey of companies that have already adopted ML techniques and ones that are yet to embrace them. Research with regard to which factors are important for industry and evaluative studies of ML based techniques/tools on these factors can complement the existing and on-going work on establishing the characteristics of ML techniques and thus contribute toward their adoption in industry and society.

THE ADOPTION OF MACHINE LEARNING TECHNIQUES FOR SOFTWARE DEFECT PREDICTION: AN INITIAL INDUSTRIAL VALIDATION

Abstract— Existing methods for predicting reliability of software are static and need manual maintenance to adjust to the evolving data sets in software organizations. Machine learning has a potential to address the problem of manual maintenance but can also require changes in how companies works with defect prediction. In this paper we address the problem of identifying what the benefits of machine learning are compared to existing methods and which barriers exist for adopting them in practice.

Our methods consist of literature studies and a case study at two companies – Ericsson and Volvo Car Group. By studying literature we develop a framework for adopting machine learning and using case studies we evaluate this framework through a series of four interviews with experts working with predictions at both companies - line manager, quality manager and measurement team leader.

The findings of our research show that the most important perceived benefits of adopting machine learning algorithms for defect prediction are accuracy of predictions and ability to generate new insights from data. The two most important perceived barriers in this context are inability to recognize new patterns and low generalizability of the machine learning algorithms.

We conclude that in order to support companies in making an informed decision to adopt machine learning techniques for software defect predictions we need to go beyond accuracy and also evaluate factors such as costs, generalizability and competence.

Keywords— Machine Learning, software defect prediction, technology acceptance, adoption, software quality

7.8 Introduction

Modelling software reliability and predicting defect prone files/modules have been a practical challenge for software project and quality managers [20]. A number of methods are available to address the challenge of Software Defect Predictions (SDP) – ranging from mathematical reliability growth modelling [65], regression based models [172], analogy based predictions [16] and expert opinions [221]. The main limitation of these methods is the fact that they are based on existing patterns (or trends) in defect inflows or software metrics and thus are not robust to changes in these patterns. Recently data mining and Machine Learning (ML) techniques have been applied in this domain with acclaimed success [21], which can address the robustness limitations. Given easy access to growing amount of data and nature of software engineering problems, the use of ML in this area is expected to grow too [211].

While a number of companies have tested or started using these methods/tools [222], the methods are still used in a limited manner - which indicates that there are number of barriers preventing companies from adopting them in practice. A number of studies have evaluated different machine learning techniques for the purpose of software defect predictions [21] [32] [31], but they focus mainly on predictive accuracy of these methods while disregarding the ease of introduction or ability to evolve together with the data sets. On the other hand, when companies consider adopting new methods/techniques, they are also concerned with a range of other factors that are currently not adequately addressed. In this paper we investigate which of these factors are important for companies when they consider using machine learning for software defect predications. The research question we address is:

What are the factors that are important for companies to make informed decision to adopt (or not adopt) ML algorithms for the purpose of software defect predictions (SDP)?

Based on review of technology adoption/acceptance and machine learning literature, we developed a framework and outlined factors that potentially affect the adoption of ML in industry in our earlier work [223]. In this paper we present the initial validation of same from the perspective of its users i.e. the industry. The main objective is to provide insights of which factors companies regard as being important to them when they consider adoption of ML techniques in this context and what their main concerns are. These insights are useful for multitude of players in this domain from researchers to tool venders and the companies themselves who can use this explicit knowledge to make better decisions using a structured framework/approach.

The remainder of this paper is organized as follows. In following section 7.9 we summarize briefly the related work. Section 7.10 introduces the study design, case study context, data and analysis methods. The ML adoption framework with important factors is provided in section 7.11, while section 7.12 provides the results from the case study. The paper ends with conclusions and ideas for future work discussed in section 7.13.

7.9 Related Work

ML has already been applied for predicting defects or defect proneness using code and change metrics and achieved good accuracy. Using code metrics data of projects from NASA IV&V facility Metrics Data Program (MDP), Menzies et al. [31] model based on naïve Bayes predicted with accuracy of 71% (pd, probability of detection) and probability of false alarm (pf) of 25%. Iker Gondra [32] also using NASA project data set (JM1) and obtained correct classifications of 72.6% with ANNs and 87.4% with SVMs. Using data from 31 projects form industry and using BNNs Fenton et al. [21] obtained an R^2 of 0.93 between predicted and actual number of defects. In [208] Menzies et al. tested different feature subset selection and report that software defect detection using machine learning approach is not too difficult in practice. As it can be observed from above cited studies - most compare and report performance with respect to predictive accuracy of different ML based algorithms, but performance evaluation on other dimensions either is limited or simply do not exist.

On the other hand studies within the area of technology adoption/acceptance have shown that adoption of complex technologies depend on multitude of factors [216] [224]. Building on the Theory of Reasoned Action (TRA) [213], Davis [41] developed the Technology Acceptance Model (TAM) to explain user acceptance of computer-based information systems. TAM has been applied and extended in number of previous studies for example to explain the adoption/acceptance of computer based technologies such as object oriented development processes by individual software developers [225], to explain the gender differences in perception of email usage [226] and predicting use of web-based information systems [227]. Wallace and Sheetz [224] used TAM in their attempt to provide a theoretical foundation for explaining and predicting the adoption of software measures. Chau and Tam [216] applied Tornatzky et al. [42] adoption framework, to explain factors affecting adoption of open systems in organizational computing, they found that organizations tend to focus more on their ability of adoption than on the benefits from adoption. Further the authors show that organizations take a reactive approach towards adoption of opens systems rather than a proactive attitude which have strong managerial implications.

We adapt and customize the TAM and Tornatzky et al. adoption framework [42] to explain which factors are relevant for explaining the adoption (or non-adoption) of machine learning techniques for software defect predictions [223]. In this paper we provide the perception of industry to these factors – which factors and their sub-dimensions (or attributes) are deemed important by the industry. The perception of industrial practitioners in this context is important as it provides useful insights on what is desired from these techniques. The framework and understanding of level of importance of attributes also help to set the direction for future research where different ML techniques can be compared on these attributes, which accelerates the technology transfer and its adoption.

7.10 Study Design

The overview of the research process employed in this study to capture the factors important for acceptance/adoption of machine learning in industry is presented in Figure 79.

Figure 79: Research process overview

The main steps in the process were:

- Existing research literature on machine learning and technology acceptance/adoption was explored for list of important benefits and challenges in applying ML in industry.
- The information was used to drive discussions with the industrial practitioners and a framework for ML adoption in industry (for software defect prediction) was developed [223].
- Attributes are mapped for each factor within this framework.
- Industrial practitioners validate the framework and mark the level of importance of each attribute in relation to making adoption decisions.

7.10.1 Case Study Context

Following the taxonomy and guidelines for conducting and reporting case studies in software engineering by Runeson and Höst [52], we conducted an exploratory case study using flexible design principle. We studied two large companies from widely different industrial domains (Automotive and Telecom) with significant focus on development of embedded software. Given the differences in domain, the study is designed as an embedded case study with two units of analysis (each company); Figure 80 and Table 44 present an overview of the case study design and summary of case units.

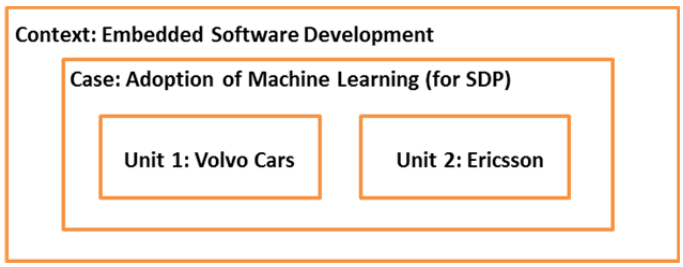

Figure 80: Case study design overview

Two companies were selected that come from two different domains:

- Volvo Car Group (VCG), A company from the automotive domain
- Ericsson, A company from the telecom domain

The divisions we interacted with have one thing in common, they have not yet adopted machine learning as their main method/technique for predicting software defects, but they are considering evaluating it as a possible technique to compliment the current measurement/prediction systems in place. Since the objective of this paper is to present the factors affecting the adoption of machine learning in industry for software defect prediction, the subjects selected are considered appropriate for the purpose.

Table 44: Overview of case units

Unit of analysis (Domain)	**Software development process**	**Current methods for SDP**	**Current state of adoption of ML for SDP**
VCG (Automotive)	V-shaped software development	Focus on status visualization and analogy based prediction	Considering evaluation
Ericsson (Telecom)	Lean and Agile development	Various modes of presenting current status and predictions methods	Considering evaluation

7.10.2 Data collection and analysis methods

The main source of data for the case study is obtained through semi-structured interviews, which is a more open method compared to structured interviews – this allows for adaptation of questions to given context and exploration of new ideas during the interview. Data collected through interviews is a form of first degree methods [52], that are although expensive to collect but offer larger control. Since the objective for this research is to explore, identify and validate factors affecting adoption of ML in industry, direct methods in form of interviews was assessed as appropriate.

Stronger conclusions can be drawn by using triangulation i.e. using data from several sources [52], therefore we complement the information obtained through interviews with document analysis from these companies. The archival documents analysed related to the information needs within the organization with respect to software defects and information demanded by various stakeholders within the organization.

Semi-structured interviews were conducted with managers responsible for providing software defects related information to different stakeholders within the organizations, these interviews were also complemented by interviews with managers responsible for quality. This setting provides us with both - the perspectives of practitioners responsible for delivering the information (roles responsible for applying/implementing ML techniques for software defect predictions) and the end users of this information who use it at various levels for decision support. The interviewees included:

- Manager at Volvo Cars Group within the department responsible for integrating software sourced from different teams and suppliers, the manager has more than 20 years of experience working with software development and testing. As ensuring safety and quality is a major responsibility in this role we refer to this manager by (VCG, QM).
- Team leader responsible for collection, analysis and reporting of project status with regard to software defects and their predictions (VCG, MetricsTL), the team leader has more than three decades of experience in various roles at the company.
- A senior quality manger whose experience with software (mainly within quality assurance) spans more than three decades (Ericsson, QM), and
- Team leader of metrics team at Ericsson; metrics team is the unit at Ericsson that provide the measurement systems within the organization (Ericsson, MetricsTL).

7.11 Factors affecting adoption of ML techniques in industry

The framework for adoption of ML in industry with how each factor is likely to affect the probability of this adoption is represented in Figure 81. In the figure (+/-) indicates the possibility (hypothesis) of existence of positive/negative relationship with medium strength between a given factor and probability of adoption of ML in industry; a double (++/--) indicate a strong relationship.

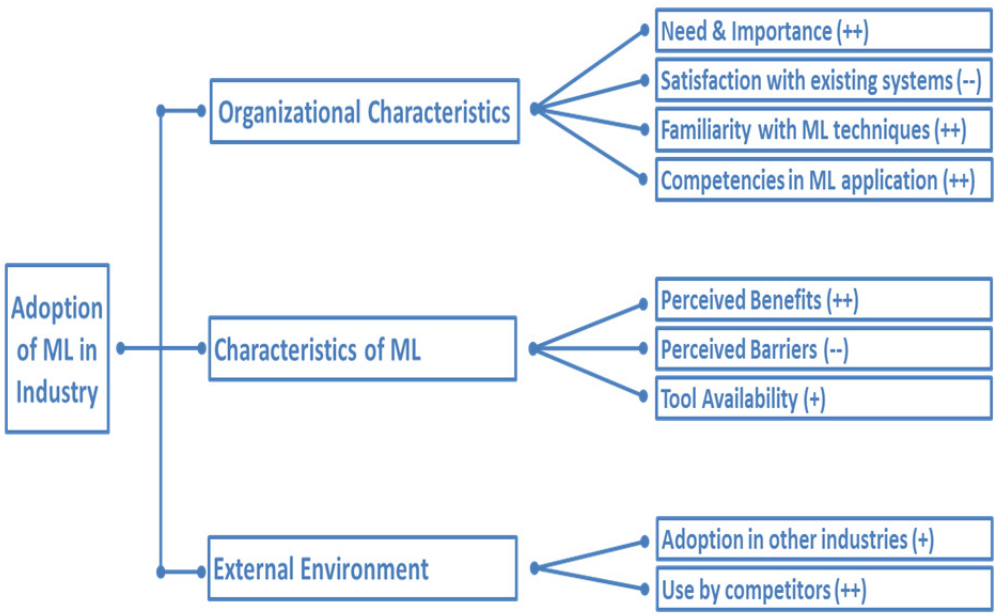

Figure 81: A framework for ML adoption in Industry

7.11.1 Organizational and ML characteristics

The factors of ML adoption framework are further broken down to sub-dimensions (or attributes) which represents the tangible measures the industrial practitioners can use to comment on their level of importance. The attributes for ML and organizational characteristics are shown in Figure 82.

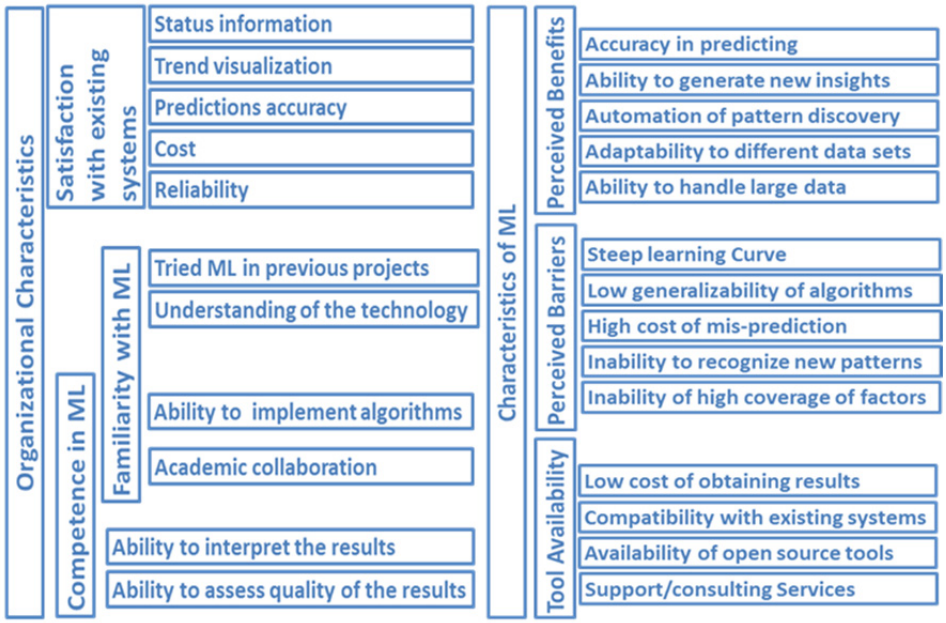

Figure 82: Overview of attributes which relate to acceptance of ML for defect prediction

7.11.2 Operationalization of factors

Factors were operationalized by asking interviewees to give the level of importance to each attributes on a five-point Likert-type scale. The levels that could be selected were:

- Very Low (VL)
- Low (L)
- Medium (M)
- High (H)
- Very High (VH)

The levels of scale reflect the degree of importance that an attribute has for adoption/acceptance of ML techniques for software defect predictions. The levels are used in different contexts; Table 45 summarizes the definitions used for each level.

Table 45: Defining the levels for different contexts

Level	Need and importance (Table 2)	Level of Satisfaction (Table 3)	Level of importance (Table 4)
Very Low (VL)	The information is not needed.	Not satisfactory, improvement is needed.	The attribute is not needed for analysis.
Low (L)	The information is desired, but not considered important.	Not satisfactory, improvement is desired.	The attribute can be considered but not required.
Medium (M)	The information is desired and is considered of value (if available).	Satisfactory, but could be improved.	The attribute is useful for making the analysis.
High (H)	The information is deemed as needed and is considered important.	Satisfaction is high.	The information on given attribute is needed for making the analysis.
Very High (VH)	The information is a must and should be provided with high accuracy.	Satisfaction is very high, with low scope for further improvement.	Cannot make a decision without information about this attribute.

7.12 Findings

7.12.1 Information need and its importance for SDP

When it comes to defect management in software development, mature organizations collect and monitor wide range of defect related metrics. There is also need for various types of predictions to manage defects (and software quality) effectively and efficiently. The interviewees from two case units were asked to indicate the importance of different information needs.

Table 46: Examples of information need and its importance in industry

Prediction Needs w.r.t software defects	VCG (QM)	VCG MetricsTL	Ericsson (QM)	Ericsson MetricsTL
Classification of defect prone files/modules	L	H	VH	VH
Expected number of defects in SW components	H	H	L	VH
Expected defect inflow for a project/release	H	H	L	VH
Release readiness/expected latent defects	H	VH	H	VH
Severity classification of defects	VH	M	H	H

Table 46 shows that different organizations information needs can be different – among others this is dependent on factors such as how the software is developed, tested and verified within an organization. At VCG similar to most OEMs (Original Equipment Manufacturers) in automotive domain, Model Based Development (MBD) is prevalent. Much of the software in this company is developed using

Simulink[12] models where code is generally auto-generated from models or sub-contracted to suppliers. In such environments classification of files/modules prone to defects is not the top priority. While for Ericsson which has more code-centric approach predicting defect prone files/modules holds very high importance.

Assessing release readiness is important (High) for both case units as is the case with severity classification of defects. It is interesting to note differences in their information need due to differences in their software testing and quality assessment approach. While at Ericsson finding smaller set of files/modules more prone to defects helps testing and quality teams to focus the limited resources to achieve high quality, at VCG knowing number of expected defects in a software component or expected defect inflow at a given point in time is more useful to mobilize their testing efforts to meet high quality demands.

7.12.2 Current status of each case unit

The same scale (five-point Likert-type) was used to indicate the level of satisfaction with current defect management/prediction systems, familiarity and in-house competence of ML techniques. The results are summarised in Table 47.

Table 47: Current status of each case unit

Factors	VCG (QM)	VCG MetricsTL	Ericsson (QM)	Ericsson MetricsTL
Satisfaction with existing systems				
Status information	H	H	H	H
Trend visualization	H	M	M	H
Predictions accuracy	M	M	L	H
Cost (current costs are low)	VH	VH	-	VH
Reliability	VH	H	VH	M
Familiarity and competence with ML techniques				
ML tried in previous project	L	L	-	M
Understanding of the technology	L	L	-	M
Ability to implement algorithms in-house	VL	M	-	M
Academic collaboration	M	H	-	M
Ability to interpret the results	H	H	-	M
Ability to assess quality of results	H	M	-	M

**In the fields marked (-), The Quality Manager interviewed at Ericsson was unable to provide assessment with high confidence, thus they are left out from analysis.*

[12] Simulink® is a block diagram environment for multidomain simulation and Model-Based Design. Matlab and Simulink are products and registered trademark of The MathWorks, Inc.

It is observed (from Table 47) that for companies currently not using ML for software defect prediction, satisfaction with existing defect monitoring and prediction systems is high, while the need to enhance the in-house competence in ML techniques is recognised.

Satisfaction with existing systems

- Stakeholders such as quality managers within these companies are satisfied to a high degree with how the defect related information is presented and trend visualized using existing systems.
- The accuracy of predictions is realized to be satisfactory, while it is considered improvements can be made.
- Cost is an important factor when choosing the prediction method - *"Cost of obtaining results is very important factor and the current systems we use are very cheap to run and maintain" – QM at VCG.*

Since the existing systems have been in place for at least two years in each case, the running costs are very low and operational reliability very high.

Familiarity and competence with ML techniques

- It is recognised that there is a need for training before ML techniques can be used for software defect prediction. The improvement potential has also been realised with respect to in-house competence of implementing such algorithms.
- Participating companies in the study show medium to high confidence with their ability to interpret the results from such analysis which is related to the need for training (see point c) above). This is due to fact that experts in these organizations have deep understanding of their process, products and impact of different factors on these gained through experience of multiple projects over long periods of time.

7.12.3 Level of importance of factors

Table 48 presents the level of the importance of different attributes that affect the adoption/acceptance of machine learning algorithms in industry.

Table 48: Level of importance of attributes for the case units

Factors	**Level of importance**			
Attributes	**VCG (QM)**	**VCG MetricsTL**	**Ericsson (QM)**	**Ericsson MetricsTL**
Perceived Benefits				
Accuracy in predicting	H	H	VH	VH
Automation of pattern discovery	M	H	VH	VH
Adaptability to different data sets	M	H	VH	VH
Ability to handle large data	H	H	M	VH
Ability to generate new insights	H	M	H	H
Perceived Barriers				
Steep learning curve	L	VH	VH	VH
Inability to recognize new patterns	VH	M	VH	VH
High cost of mis-predicitons	M	M	VL	L
Low generalizability of algorithms	H	H	H	VH
Need for high coverage of relevant attributes	M	M	L	VH
Tool availability				
Compatibility with existing systems	M	L	H	VH
Availability of open source tools	L	H	M	VH
Low cost of obtaining results	VH	H	H	M
Support/consulting services	H	M	L	VL
External factors				
Adoption by other industries	L	L	L	M
Use by competitors	H	M	L	M

It is observed that while there are some variations depending on the case units, attributes related to perceived benefits and barriers are considered highly important for making adoption decisions. Attributes related to tool availability are deemed important but not critical, while external factors had little influence on adoption decisions of ML techniques in industry for SDP. Specifically,

Perceived Benefits

- Accuracy of predictions, automation, adaptability and ability to handle large are generally regarded as high or very high importance. Some interesting comments highlight these, *"When it comes to the benefits, accuracy and automation are the top priorities for us" – MetricsTL at Ericsson.*
- Using causal models such as Bayesian Networks that can provide range of decision-support and risk assessment capabilities for project managers [21] is perceived as an important benefit of ML techniques application to SDP.

Perceived Barriers

On the other hand uncertainty over if the ML based techniques can be effective for detecting new patterns in the data and concerns over their generalizability are barriers that are considered highly important for making adoption decisions.

- Technology/innovations that need high upfront investment in terms of new knowledge acquisition can slow the adoption process. The respondents in our study also considered this attribute as highly important, except for the QM at VCG, according to him *"Steep learning curve is not a major problem if only few people (experts) need to know it to generate the results as long as they are easy to interpret by rest of the stakeholders" – QM at VCG.*
- Mispredictions can be costly for an organization and usually considered a barrier for prediction systems, but managers at both case units emphasized that this is not a show stopper. *"Since all predictions generally go through number of experts if the predictions are not close to reality they would not be accepted by these experts." – MetricsTL at VCG.*

Tool availability

It was revealed that availability of tools is important for organizations, while if the tools are open source or proprietary does not have same impact on the adoption of new techniques.

- Information that is relevant to companies with respect to tools is the cost of running it (in terms of resources) should be low; i.e. it should be fairly easy to feed in the data and generate the results.
- Availability of support/consulting service for given tool is another factor that depends on given company preferences, companies like VCG in our case study, prefer to use a sub-supplier to provide services which are new to the company and generally incorporate them within in-house systems when confidence in their usability and effectiveness is well established. *"Even if open source tools are available, we typically need a vendor in between to do tool integration, manage upgrades and do maintenance work – we do not have resources for that" – QM at VCG.*

While at Ericsson, the departments which are supported by a specialized in-house team to cater the need of measurement systems, for these departments in order to achieve high transparency and provide greater flexibility prefer to develop in-house measurement systems than relying on external vendors where possible.

External factors

- When it comes to external factors, adoption of a new method/techniques by industries outside of given industrial domain have low impact, while the knowledge that similar techniques are being used within the domain can highly motivate their evaluation within a given company. *"We are not afraid of trying new things and being the first one, but if it is used in automotive sector and we have not tried it surely helps the case" – QM at VCG.*
- The perspective on some attributes within the company is also dependent on the role. This is mainly due to the fact that some roles (as quality manger) are consumers of the information/measurement system, while in others (as a team leader of metrics team) the responsibility is to supply this information (responsible for building and maintaining the measurement/prediction systems).
- The difference can be large for some attributes, while QM is not concerned with maintenance aspects, MetricsTL said with respect to ML techniques for software defect predictions: *"I am not confident that maintenance cost is low with respect to competence and technology we have today" – MetricsTL at Ericsson.*
- Explaining it further MetricsTL highlighted *"first developing a prediction system is time consuming task and further if I have to update it often then costs will be too high. Other thing is that we change our technology (for e.g. tools) from time to time – so what does that mean as a developer of ML based prediction system?"*

7.12.4 Specific challenges in adopting ML techniques in industry for SDP

Apart from common factors identified in previous section, in this section we present the specific challenges that were raised during the interviews towards accepting the ML techniques for software defect prediction.

Lack of information to make a strong business case: Does ML techniques save company time or will they reduce risk? If so how much? These are some of the important questions mangers need - to make a strong business case for motivating the use of new techniques within their team and within the company.

"*Time is a critical factor, especially in automotive domain where a new functionality is promised to the market long before it is completely ready, then the clock is ticking and the product development divisions are expected to deliver on time with superior quality*" *– QM at VCG.*

If expected time savings or reduction in risk could be quantified for a given company, their decision on adoption becomes much easier.

Uncertainty on applicability of ML when access to source code is not available: In cases where software is purchased from suppliers, the access to code and change metrics may not be available. It is unclear if ML based algorithms can still be useful and effective for SDP.

How to adapt ML techniques for model driven development: Model driven development is predominant in many industries such as aerospace and automotive domain model. The question that is yet unanswered for these organizations is if ML based prediction systems can be effectively applied for their specific context.

Some of the important questions are - can we adapt ML based techniques to analyse models (e.g. UML, Simulink etc.) for the purpose of defects or quality predictions? Or can the metrics obtained from code (which is usually auto-generated from these models) be appropriate for SDP using current ML based approaches?

How to effectively use text base artefacts for SDP: While most ML based techniques for SDP use quantitative data, some of the major software artefacts such as requirements and defect reports are largely text based. The ability of ML based methods to reliably handle textual data will boost confidence of industry in these methods, Menzies and Marcus [228] work is a good example of type of work these companies want to see more.

Uncertainty over where ML fits in context of compliance to standards: Industrial domains with safety critical software usually follow stringent safety standards. For example in automotive domain, ISO 26262 is the new functional safety standard which recommends using formal methods for software verification and validation for high safety critical applications. How does ML based software prediction techniques fit in this framework and how can they contribute towards ensuring compliance to such standards is another area currently not well understood but important for organizations in such domain.

7.12.5 Validity

Threats to validity in this study are addressed in manner as described by Wohlin et al. [50]. There exists threat to internal validity to this study with respect to the selection of case units – both case units have not adopted ML widely for software defect predictions. For example it can be expected that there may be a difference between the perceived benefits among companies that have adopted such techniques and those that have not. In this study we only report how important these units feel

these attributes are for taking an adoption decision thus minimizing the mentioned threat. This also aligns well with the objective of this study where our aim is to explore and list the important factors and not what the case units in this study's assessment is about ML techniques and tools.

Threat to construct validity exists with respect to if or not all factors that are important for making adoption decisions of ML in industry are taken into account. We explored the factors and attributes closely with the companies involved in this study. The attributes and model were again validated with the companies involved which limit the possibility of miss-interpretation which minimizes the threat to construct validity.

Incorrect conclusions about relationships can pose threat of conclusion validity. The presented study is designed as an exploratory case study. We present the perception of industry of which factors they deem as important with indications of possible relationship to the adoption framework. The future study planned that quantitatively assesses these relationships will have to seriously evaluate this threat to validity, but for present study it does not pose a major threat.

Threat to external validity is a major threat to this study, since only two units within two large software development organizations are used for validation, but numbers of steps are taken to minimize this threat. Firstly the adoption framework is based on wider technology adoption/acceptance literature, secondly the model is claimed to be only initially validated with these case units, comprehensive validation and quantitative assessment is planned as future work. Further using case units from widely different industrial domain and using different job roles within the units and two stage interviews help minimize the threat to external validity.

7.13 Conclusions and future work

In large software development organizations, a software defect prediction is important for project and quality mangers to realise the goal of zero known defects by the release date. Machine learning techniques offer an alternative to methods based on statistical regression or expert opinions. ML based methods have been compared to traditional methods for aspect such as predictive accuracy, but for companies considering adoption of ML based techniques, a number of other factors are also important.

In this paper we set out to investigate, *What are the factors that are important for companies to make informed decision to adopt (or not adopt) ML algorithms for the purpose of software defect predictions (SDP)?* We identified a total of nine

important factors and 27 related attributes that affect the adoption of ML based techniques for software defect predictions. The framework for adoption of ML for SDP is validated using a series of interviews with experts on quality and team leaders responsible for providing software defects related information at two large software development organizations.

The results suggest that information needs can be different for different companies based on their software development and testing process. The existing systems in place for presenting and visualizing information related to software defects are deemed satisfactory, they offer low running costs and high reliability. The need for training to increase competence in ML techniques is also recognised in these companies. The study further show that for adopting ML techniques, predictive accuracy and ability to generate new insights from large data are most important perceived benefits. At the same time low generalizability and steep learning curve are perceived barriers that need to be overcome to gain higher adoption of ML in industry. Availability of tools and support services can also accelerate the adoption process in this respect.

Impact of understanding such factors is at multiple levels: for companies themselves it explicitly lists the factors that are implicitly deemed important by them when they make adoption decisions on ML based techniques/tools. Listing and visualizing important attributes for such decisions also makes it easier for mangers to see the big picture and objectively evaluate new ML based techniques and tools for their usefulness and applicability for a given problem at a given point in time. The adoption framework is also useful for companies that provide tools and services to larger organizations developing software. With knowledge of important factors they can customize their products (e.g. tools) and services offerings to closely fit the need of these organizations.

In future work we plan to quantitatively evaluate the effect size of important attributes towards ML adoption decision using large scale survey of companies that have already adopted ML techniques and ones that are yet to embrace it. Research with regard to which factors are important for industry and evaluative studies of ML based techniques/tools on these factors can complement the existing and on-going work on establishing the characteristics of ML techniques and thus contribute toward their adoption in industry and society.

Chapter 8:
Summary of research results

8 SUMMARY OF RESEARCH RESULTS

The main results provided in the thesis provide evidence of how software defect prediction techniques can be used in the context of automotive software development. This section presents the summary of results from individual studies included in the thesis, while in next section (8.1) the conclusions drawn from the combined results is provided, finally section 8.2 presents the areas for future research.

Results from the overview study of software defect prediction techniques and the automotive software life cycle

The first study presented in chapter 2 provided the overview of software development life cycle at the level of full EE (Electronics & Electrical System) platform projects. Three distinct phases of life cycle namely concept phase, production software development and in-operations phase were identified and iterative development in the production software discussed. In particular the study resulted in:

a. Life cycle overview of full EE platform projects in automotive domain.
b. Overview of different software defect prediction techniques applicable in automotive domain with their mapping to when they can be applied over the platform project timeline.
c. Classification of defect prediction techniques on basis of what purpose they can be used for and at what granularity level they can be applied.
d. A roadmap for using in-operations data for improving the efficiency of defect prediction techniques.

The mapping to project timeline and classification based on the application purpose and granularity help with selection of appropriate defect prediction technique. The input data needed for each technique and their main advantages and limitations were also highlighted to aid industrial practitioners with the selection. The roadmap included in the study presents possible future scenario where easy retrieval of in-operation data can be used to help calibrate software for optimal performance, develop and adapt software features based on user interaction data.

Results from the evaluation of SRGMs in chapter 3

Chapter 3 was particularly aimed at evaluating SRGMs in the context of automotive domain. A set of commonly used models were evaluated at a sub-system level and also on multiple EE platform projects from the automotive domain. The models

were also evaluated on projects from other industrial domains from the embedded domain that help enhance the external validity of the results. In summary the results were:

a. Two of the commonly used parameter estimation techniques namely maximum likelihood estimation (MLE) and non-linear regression (NLR) were compared and evaluated for their statistical properties and practicality for applying SRGMs to defect count data.
b. A balanced metric for measuring asymptote prediction accuracy, Balanced Predicted Relative Error (BPRE) was defined which is symmetric for over and under predictions unlike more commonly used Predicted Relative Error (PRE).
c. Evaluation of SRGMs on a project from one division at VCG (sub-system level project) showed the ability of SRGMs to fit the defect inflow data.
d. A set of SRGMs were evaluated on eleven large projects from three different industrial domains (including four EE platform projects).
e. Models that performed best within the tested set for defect count prediction and for assessment of release readiness were identified. Overall it was observed that Logistic and Gompertz model performed well compared to other tested models.
f. The results also suggested that given software development process, the asymptote prediction accuracy could be significantly improved by using growth rate from historical projects.
g. It was shown that using simple trend analysis the shape of defect inflow profile for an on-going project could be predicted as early as halfway through the project timeline. Predicting the probable shape of defect inflow profile is useful in selecting the appropriate SRGM for the given on-going project for making defect count predictions.
h. In another study included in this chapter, the defect inflow distribution family was analysed where beta distribution family was shown to fit best to the defect inflow from number of projects data used in the study. Understanding distribution of historical and on-going projects within an organization can help in choosing correct statistical methods, easy visualization and simulation.
i. Knowing distribution of historical projects is also useful in Bayesian analysis were the information is used for describing the prior probabilities.

Overall in chapter 3 we evaluated SRGMs for their applicability for defect count prediction and assessment of release readiness in the automotive domain and found

that they provide a viable option for these analyses. Methods for selection of appropriate SRGMs for an on-going project were also developed and evaluated.

Results from the study of consequence of mispredictions in chapter 4

In chapter 4 we continued with the theme of evaluating SRGMs, but in this chapter we focused on the cost/consequences of mispredictions. Since risk of misprediction is always present when making any forecasts, understanding the possible consequences help practitioners to actively take them into account when they use prediction models in practice. The results from the study suggested:

a. The two main possibilities of misprediction are (i) mispredicting the expected defect count (the asymptote) - over or under predicting it, and (ii) Mispredicting when the total expected defect count would be reached (timing) - early or late predictions.
b. It was noted in the study that while theoretically the consequences would be same for each scenario of misprediction, the cost/consequences to a given organization depend on their domain, process and organizational structure. The case study at two companies (VCG and Ericsson) also summarized the possible response action for such scenarios.

The research question addressed in this chapter provided the balanced view on using defect count predictions and assessment of reliability analysis. While most predictions and data driven analysis help industrial practitioners make informed decisions, but predictions should be made and used with careful consideration of possible risks which were explored in this study for case of using reliability models.

Results from the evaluation of correlation based SDP technique in automotive domain

Having evaluated the applicability and possible risks of using SRGMs, in chapter 5 we evaluated correlation based defect prediction technique for its applicability in the automotive domain. Since EE platform projects tend to be large and span long time period, they are also organized into several iterations over which part of software is designed, implemented and tested. The study results:

a. Supported earlier observations that small amount of software modules (ECUs/features in our case) account for majority of defect counts.
b. The correlation between number of defects found across integration points is found to be moderately-to-very strongly, which may be used for building not very accurate, but quick and easy prediction rules.

c. The pre-release defect count for software modules was strongly correlated to defect count at integration point four/five which is about midway through the project timeline.
d. It was also discussed how correlation based model can be used to identify software modules that may need further review and/or testing.

Correlation based prediction models can use historical projects data to provide simple thumb rule based prediction models that are very easy to apply in practice and are also very intuitive for various stakeholders involved.

Results from the reliability analysis using executable models
In the next study presented in chapter 6, behavioural models that are produced in a Model Based Development (MBD) environment were proposed to be used for early assessment of reliability characteristics. The main results of the study were:

a. A framework using fault injection and mutation based testing was proposed for identification of design defects and defects that can lead to possible safety case violations (according to the definition of ISO 26262 functional safety standard).
b. Since the proposed assessment can be done early in the project life cycle (using executable models), they can provide early feedback to designers and caution testers by assessing the efficacy of test suite intended to catch possible implementation defects.
c. The initial validation of framework on miniature cars provided encouraging results.

Fault injection has been long used successfully for assessment of dependability characteristics in the hardware domain, while mutation testing has been applied with good results for traditional code. With executable models availability in the MBD, the proposed framework can be used for early reliability assessment which can lead to robust design and software with superior reliability properties.

Results from chapter 7
Finally we also looked at machine learning based techniques for software defect prediction. While our research in earlier studies have raised questions on the applicability of these techniques when access to source code and software evolution data is not readily available. Still these techniques are useful within automotive domain where software is developed full or part of software is developed in-house. A number of earlier studies have presented the evaluation of these techniques with promising results, but their adoption in industry is not so widespread. Thus we focused our effort on understanding the factors that influence industrial

practitioners' decision to adopt or defer the decision of adopting such techniques. The study resulted in:

a. A framework for adoption of machine learning based techniques in industry which is based on the technology adoption and acceptance models.
b. The framework was adapted for the specific case of software defect prediction resulting in identification of nine main attributes and number of sub-attributes for each main factor.
c. Guidelines for using framework for evaluating a new potential machine learning based technique for adoption as well as using the framework for comparative purposes between two techniques/tools were provided.
d. The initial validation of framework at two partner companies (VCG and Ericsson) highlighted several important factors that affect the adoption decisions at industry.

Understanding of factors important in adoption decisions serves multiple purposes, firstly it helps in identifying possible areas of research that can address information gaps and thus accelerate the technology transfer process. Secondly it provides tool developers and vendors to prioritize features according to the demands of their intended users and finally it provides industrial practitioners with an objective framework that can help them evaluate potential new techniques and/or alternative tools for adoption.

8.1 Conclusions

We set out this project with the following main goal:

To evaluate how software defect prediction techniques can be effectively applied over the software development life cycle within the automotive domain.

While software developed within automotive domain is highly diverse, we focused mainly at the level of full EE (Electronics & Electrical system) platform projects. In the thesis we started with reviewing the software development process, life cycle of EE platform projects and commonly used software defect prediction techniques. The review of development process and constraints specific to automotive domain limits application of some of these techniques thus we placed special emphasis on techniques that could be applied under given constraints. The series of studies presented in the thesis resulted in evaluation of techniques for defect count prediction and assessment of release readiness.

The thesis shows that testing data driven models such as software reliability growth models and correlation based predictions can be used for defect count predictions, it was specifically shown that:

a. The shape of defect inflow profile can be predicted using simple trend analysis about midway through the project timeline.
b. SRGMs can be used for defect count prediction and release readiness assessment for large software development projects (within embedded and specifically in automotive domain).
c. Using historical data, predicting shape of defect inflow profile and analysis of defect distribution of historical projects can help with selection of appropriate SRGMs.
d. Correlation based models can potentially be used for predicting defect count in upcoming iteration and also predicting pre-release defect count.

The applied aspect of applying these models in the context of automotive domain such as parameter estimation method, selecting best performance model from set of selected models and choosing an appropriate model for predictions were also evaluated as part of the thesis. The evaluation was done to provide objective assessment of these techniques to support decisions on optimal allocation of test resource and with decisions related to release timing.

Moreover a framework combining fault injection with mutation based testing approach applied at the executable behavioural models was proposed and validated. The framework support early identification of design defects and identification of possible implementation defects capable of violating safety goal requirements. Testing behavioural models for possible design defects and testing efficacy of test suite with mutation testing potentially provides a narrow feedback loop for designers to improve and develop robust designs with superior reliability characteristics. By evaluating the efficacy of test suite early, it is possible to avoid potential design and implementation defects from slipping over.

Finally an adoption framework for machine learning based models for software defect predictions was developed and validated. ML based techniques can support parts of software development within automotive domain especially sections of organizations with in-house software development. The adoption framework highlighted important factors such as setup, running and maintenance costs. Understanding of these factors and the adoption framework is useful for various stakeholders including researchers evaluating such techniques, tool vendors and organizations looking to adopt these techniques or tools based on them.

Overall in the thesis we evaluated number of different software defect prediction techniques that can be applied over software development life cycle in the automotive domain. It was shown when they can be applied, for what purpose and at what granularity level they can be applied. A set of these techniques were evaluated for their performance for the objective of supporting optimal resource allocation decisions and release readiness assessment. Using defect prediction techniques and frameworks for early identification of design and possible implementation defects can lead to better allocation of limited test resources and release of mature software with superior reliability characteristics.

8.2 Future research

The research in this thesis opens several directions for further research, particularly in collaboration with industry. The thesis provides a basis for evaluating applicability of software defect predictions in the automotive domain for supporting questions of high practical importance. A set of techniques were evaluated in this thesis and new frameworks proposed and validated, nonetheless a number of open questions and areas for further research are identified, these can be grouped into separate, although not mutually exclusive categories.

8.2.1 A comprehensive comparison of different software defect prediction techniques within embedded domain using data from large set of cross-company projects.

The main research direction stemming from the thesis is the comprehensive comparison of different software defect prediction techniques within the embedded software domain using large number of projects from representative sample of companies from this domain. In particular, the ambitious research goal could be achieved by studies focusing on:

a. Setup of open databases similar to Tukutuku [229] and Promise repository [230] which is open for researchers, but also promote commercial organizations to add anonymised project and defect data, and
b. Using cross-company database for benchmarking and comparative evaluation of different techniques.

8.2.2 Defining and validating product metrics for behavioural models in domain specific languages.

Another research direction that can be pursued is the definition of important product metrics for behavioural models in Domain Specific Languages (DSL) and their validation. A number of industrial domains specifically within the embedded software use various DSLs such as Matlab/Simulink for modelling purposes. Often these models are executable and made at implementation level details (behavioural models) which are used for generation of code that goes into the final systems. While number of metrics have been defined and validated for sequential and object oriented code to measure size, complexity, coupling etc., similar metrics are not yet well defined and validated for behavioural models. Validated metrics for such models will enable enhanced monitoring and control of software evolution properties developed using DSLs. It will also pave the path for application of regression and machine learning based defect prediction and classification techniques that use product metrics and thus cannot be used in such cases.

8.2.3 Industrial validation and further exploration of using fault injections and mutation based approaches on behavioural models for dependability evaluations.

The use of behavioural models is widespread in the automotive and many other industrial domains such as aerospace. Also models are usually developed early in the development lifecycle, thus increasing their use for verification and validation can not only enhance the dependability characteristics of the software under development, but also contribute towards shorter development cycle hence reducing market lead time. Use of approaches such as fault injection and mutation testing and frameworks based on them need to be validated on industrial scale projects and their performance and possible contribution evaluated.

REFERENCES

[1] C. Ebert and C. Jones, “Embedded software: Facts, figures, and future,” *Computer*, vol. 42, no. 4, pp. 42–52, 2009.

[2] E. L. Jones, “Integrating testing into the curriculum—arsenic in small doses,” in *ACM SIGCSE Bulletin*, 2001, vol. 33, pp. 337–341.

[3] M. Broy, “Challenges in automotive software engineering,” in *Proceedings of the 28th international conference on Software engineering*, 2006, pp. 33–42.

[4] S. Fürst, “Challenges in the design of automotive software,” in *Proceedings of the Conference on Design, Automation and Test in Europe*, 2010, pp. 256–258.

[5] U. Eklund and H. akan Gustavsson, “Architecting automotive product lines: Industrial practice,” *Science of Computer Programming*, vol. 78, no. 12, pp. 2347–2359, 2013.

[6] R. B. Grady, “Software failure analysis for high-return process improvement decisions,” *Hewlett Packard Journal*, vol. 47, pp. 15–24, 1996.

[7] C.-Y. Huang, M. R. Lyu, and S.-Y. Kuo, “A unified scheme of some nonhomogenous poisson process models for software reliability estimation,” *IEEE Transactions on Software Engineering*, vol. 29, no. 3, pp. 261–269, 2003.

[8] M. McDonald, R. Musson, and R. Smith, *The practical guide to defect prevention*. Microsoft Press, 2007.

[9] *1044-2009-IEEE Standard Classification for Software Anomalies*. 2010.

[10] N. Melleg\a ard, *Improving Defect Management in Automotive Software Development, LiDeC—A Light-weight Defect Classification Scheme*. Chalmers University of Technology, 2013.

[11] R. Chillarege, I. S. Bhandari, J. K. Chaar, M. J. Halliday, D. S. Moebus, B. K. Ray, and M.-Y. Wong, “Orthogonal defect classification-a concept for in-process measurements,” *Software Engineering, IEEE Transactions on*, vol. 18, no. 11, pp. 943–956, 1992.

[12] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, “Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings,” *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 485–496, 2008.

[13] A. Wood, “Predicting software reliability,” *Computer*, vol. 29, no. 11, pp. 69–77, 1996.

[14] V. Almering, M. van Genuchten, G. Cloudt, and P. J. Sonnemans, “Using software reliability growth models in practice,” *IEEE Software*, vol. 24, no. 6, pp. 82–88, 2007.

[15] M. Staron and W. Meding, “Predicting weekly defect inflow in large software projects based on project planning and test status,” *Information and Software Technology*, vol. 50, no. 7, pp. 782–796, 2008.
[16] M. Staron, W. Meding, and B. Söderqvist, “A method for forecasting defect backlog in large streamline software development projects and its industrial evaluation,” *Information and Software Technology*, vol. 52, no. 10, pp. 1069–1079, 2010.
[17] B. Clark and D. Zubrow, “How good is the software: a review of defect prediction techniques,” *sponsored by the US department of Defense*, 2001.
[18] S. McConnell, “Gauging software readiness with defect tracking,” *Software, IEEE*, vol. 14, no. 3, pp. 136–135, 1997.
[19] L. C. Briand, K. El Emam, B. G. Freimut, and O. Laitenberger, “A comprehensive evaluation of capture-recapture models for estimating software defect content,” *Software Engineering, IEEE Transactions on*, vol. 26, no. 6, pp. 518–540, 2000.
[20] N. E. Fenton and M. Neil, “A critique of software defect prediction models,” *IEEE Transactions on Software Engineering*, vol. 25, no. 5, pp. 675–689, 1999.
[21] N. Fenton, M. Neil, W. Marsh, P. Hearty, Ł. Radliński, and P. Krause, “On the effectiveness of early life cycle defect prediction with Bayesian Nets,” *Empir Software Eng*, vol. 13, no. 5, pp. 499–537, Oct. 2008.
[22] A. Mockus, “Analogy based prediction of work item flow in software projects: a case study,” in *2003 International Symposium on Empirical Software Engineering, 2003. ISESE 2003. Proceedings*, 2003, pp. 110–119.
[23] T. M. Khoshgoftaar, J. C. Munson, and D. L. Lanning, “A comparative study of predictive models for program changes during system testing and maintenance,” in *Software Maintenance, 1993. CSM-93, Proceedings., Conference on*, 1993, pp. 72–79.
[24] T. M. Khoshgoftaar, J. C. Munson, B. B. Bhattacharya, and G. D. Richardson, “Predictive modeling techniques of software quality from software measures,” *Software Engineering, IEEE Transactions on*, vol. 18, no. 11, pp. 979–987, 1992.
[25] S. Chulani and B. Boehm, “Modeling Software Defect Introduction and Removal,” 1999.
[26] B. W. Boehm, *Software engineering economics. 1981*. Prentice-Hall.
[27] C. Jones, “Programming defect removal,” *Proceedings, GUIDE*, vol. 40, 1975.

[28] T.-J. Yu, V. Y. Shen, and H. E. Dunsmore, “An analysis of several software defect models,” *Software Engineering, IEEE Transactions on*, vol. 14, no. 9, pp. 1261–1270, 1988.
[29] T. M. Khoshgoftaar and E. B. Allen, “Logistic regression modeling of software quality,” *International Journal of Reliability, Quality and Safety Engineering*, vol. 6, no. 04, pp. 303–317, 1999.
[30] T. Zimmermann, R. Premraj, and A. Zeller, “Predicting defects for eclipse,” in *Predictor Models in Software Engineering, 2007. PROMISE’07: ICSE Workshops 2007. International Workshop on*, 2007, pp. 9–9.
[31] T. Menzies, J. Greenwald, and A. Frank, “Data mining static code attributes to learn defect predictors,” *IEEE Transactions on Software Engineering*, vol. 33, no. 1, pp. 2–13, 2007.
[32] I. Gondra, “Applying machine learning to software fault-proneness prediction,” *Journal of Systems and Software*, vol. 81, no. 2, pp. 186–195, Feb. 2008.
[33] M. Broy, I. H. Kruger, A. Pretschner, and C. Salzmann, “Engineering automotive software,” *Proceedings of the IEEE*, vol. 95, no. 2, pp. 356–373, 2007.
[34] W. Dieterle, “Mechatronic systems: Automotive applications and modern design methodologies,” *Annual Reviews in Control*, vol. 29, no. 2, pp. 273–277, 2005.
[35] ISO, “International Standard-ISO 26262-Road vehicles-Functional safety.” International Organization for Standardization, 2011.
[36] B. Kitchenham, S. Linkman, and D. Law, “DESMET: a methodology for evaluating software engineering methods and tools,” *Computing & Control Engineering Journal*, vol. 8, no. 3, pp. 120–126, 1997.
[37] B. A. Kitchenham, “Evaluating software engineering methods and tool part 1: The evaluation context and evaluation methods,” *ACM SIGSOFT Software Engineering Notes*, vol. 21, no. 1, pp. 11–14, 1996.
[38] D. Goldsman, B. L. Nelson, and B. Schmeiser, “Methods for selecting the best system,” in *Proceedings of the 23rd conference on Winter simulation*, 1991, pp. 177–186.
[39] T. Dyba, B. A. Kitchenham, and M. Jorgensen, “Evidence-based software engineering for practitioners,” *Software, IEEE*, vol. 22, no. 1, pp. 58–65, 2005.
[40] S. L. Pfleeger, “Understanding and improving technology transfer in software engineering,” *Journal of Systems and Software*, vol. 47, no. 2–3, pp. 111–124, Jul. 1999.

[41] F. D. Davis Jr, “A technology acceptance model for empirically testing new end-user information systems: Theory and results,” Massachusetts Institute of Technology, 1986.
[42] L. G. Tornatzky, M. Fleischer, and A. K. Chakrabarti, “Processes of technological innovation,” 1990.
[43] T. Gorschek, C. Wohlin, P. Carre, and S. Larsson, “A Model for Technology Transfer in Practice,” *IEEE Software*, vol. 23, no. 6, pp. 88–95, Nov. 2006.
[44] W. R. Adrion, “Research methodology in software engineering,” in *Summary of the Dagstuhl Workshop on Future Directions in Software Engineering” Ed. Tichy, Habermann, and Prechelt, ACM Software Engineering Notes, SIGSoft*, 1993, vol. 18, pp. 36–37.
[45] R. L. Glass, “The software-research crisis,” *Software, IEEE*, vol. 11, no. 6, pp. 42–47, 1994.
[46] V. R. Basili, “The experimental paradigm in software engineering,” in *Experimental Software Engineering Issues: Critical Assessment and Future Directions*, Springer, 1993, pp. 1–12.
[47] R. L. Glass, I. Vessey, and V. Ramesh, “Research in software engineering: an analysis of the literature,” *Information and Software technology*, vol. 44, no. 8, pp. 491–506, 2002.
[48] C. R. Kothari, *Research Methodology: Methods and Techniques*. New Age International, 2011.
[49] M. Aliaga and B. Gunderson, *Interactive statistics*. Prentice Hall, 1999.
[50] C., Runeson,Per, Höst,Martin, Ohlsson,Magnus C Wohlin and B., Wesslén,Anders Regnell, *Experimentation in Software Engineering*. New York: Springer, 2012.
[51] R. K. Yin, *Case study research: Design and methods*, vol. 5. Sage, 2009.
[52] P. Runeson and M. Höst, “Guidelines for conducting and reporting case study research in software engineering,” *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, 2009.
[53] V. R. Basili, G. Caldiera, and H. D. Rombach, “Experience factory,” *Encyclopedia of software engineering*, 1994.
[54] R. N. Charette, “This car runs on code,” *IEEE Spectrum*, vol. 46, no. 3, p. 3, 2009.
[55] B. W. Boehm, “A spiral model of software development and enhancement,” *Computer*, vol. 21, no. 5, pp. 61–72, 1988.
[56] R. Rana, M. Staron, N. Melleg\aard, C. Berger, J. Hansson, M. Nilsson, and F. Törner, “Evaluation of Standard Reliability Growth Models in the Context of

Automotive Software Systems," in *Product-Focused Software Process Improvement*, Springer, 2013, pp. 324–329.
[57] R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, and F. Törner, "Evaluating long-term predictive power of standard reliability growth models on automotive systems," presented at the 24th annual International Symposium on Software Reliability Engineering (ISSRE 2013), Pasadena, CA, USA, 2013.
[58] E. Ceylan, F. O. Kutlubay, and A. B. Bener, "Software defect identification using machine learning techniques," in *32nd EUROMICRO Conference on Software Engineering and Advanced Applications, SEAA'06.*, 2006, pp. 240–247.
[59] R. Kitchin and M. Dodge, *Code/space: Software and everyday life*. The MIT Press, 2011.
[60] K. S. Lew, T. S. Dillon, and K. E. Forward, "Software complexity and its impact on software reliability," *IEEE Transactions on Software Engineering*, vol. 14, no. 11, pp. 1645–1655, 1988.
[61] M. Camuffo, M. Maiocchi, and M. Morselli, "Automatic software test generation," *Information and Software Technology*, vol. 32, no. 5, pp. 337–346, 1990.
[62] C.-T. Lin and C.-Y. Huang, "Enhancing and measuring the predictive capabilities of testing-effort dependent software reliability models," *Journal of Systems and Software*, vol. 81, no. 6, pp. 1025–1038, 2008.
[63] T. Goradia, "Dynamic impact analysis: A cost-effective technique to enforce error-propagation," *ACM SIGSOFT Software Engineering Notes*, vol. 18, no. 3, pp. 171–181, 1993.
[64] M. Xie, "Software Reliability Models - Past, Present and Future," in *Recent Advances in Reliability Theory*, N. Limnios and M. Nikulin, Eds. Birkhäuser Boston, 2000, pp. 325–340.
[65] M. R. Lyu, *Handbook of software reliability engineering*, vol. 3. IEEE Computer Society Press CA, 1996.
[66] H. Pham, "Software reliability and cost models: Perspectives, comparison, and practice," *European Journal of Operational Research*, vol. 149, no. 3, pp. 475–489, 2003.
[67] X. Zhang, X. Teng, and H. Pham, "Considering fault removal efficiency in software reliability assessment," *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 33, no. 1, pp. 114–120, 2003.

[68] M. R. Lyu, “Software reliability engineering: A roadmap,” in *Future of Software Engineering, 2007. FOSE’07*, 2007, pp. 153–170.

[69] S. H. Kan and others, *Metrics and Models in Software Quality Engineering, 2/e*. Pearson Education India, 2003.

[70] P. Kapur, H. Pham, S. Anand, and K. Yadav, “A unified approach for developing software reliability growth models in the presence of imperfect debugging and error generation,” *IEEE Transactions on Reliability*, vol. 60, no. 1, pp. 331–340, 2011.

[71] A. L. Goel, “Software reliability models: Assumptions, limitations, and applicability,” *IEEE Transactions on Software Engineering*, no. 12, pp. 1411–1423, 1985.

[72] J. D. Musa, A. Iannino, and K. Okumoto, *Software reliability*. McGraw-Hill New York, 1987.

[73] K. Sharma, R. Garg, C. K. Nagpal, and R. K. Garg, “Selection of Optimal Software Reliability Growth Models Using a Distance Based Approach,” *IEEE Transactions on Reliability*, vol. 59, no. 2, pp. 266–276, 2010.

[74] C. Stringfellow and A. A. Andrews, “An empirical method for selecting software reliability growth models,” *Empirical Software Engineering*, vol. 7, no. 4, pp. 319–343, 2002.

[75] T. M. Khoshgoftaar and T. G. Woodcock, “Software reliability model selection: a cast study,” in *Proceedings of International Symposium on Software Reliability Engineering, 1991.*, 1991, pp. 183–191.

[76] N. Ullah, M. Morisio, and A. Vetro, “A Comparative Analysis of Software Reliability Growth Models using Defects Data of Closed and Open Source Software,” in *35th Annual IEEE Software Engineering Workshop (SEW)*, 2012, pp. 187–192.

[77] R. Rana, M. Staron, N. Mellegård, C. Berger, J. Hansson, M. Nilsson, and F. Törner, “Evaluation of standard reliability growth models in the context of automotive software systems,” in *Product-Focused Software Process Improvement*, Springer, 2013, pp. 324–329.

[78] IEEE Reliability Society, “IEEE Recommended Practice on Software Reliability.” The Institute of Electrical and Electronics Engineers, Inc, 2008.

[79] A. L. Goel and K. Okumoto, “Time-dependent error-detection rate model for software reliability and other performance measures,” *IEEE Transactions on Reliability*, vol. 28, no. 3, pp. 206–211, 1979.

[80] S. Yamada, M. Ohba, and S. Osaki, “S-shaped reliability growth modeling for software error detection,” *IEEE Transactions on Reliability*, vol. 32, no. 5, pp. 475–484, 1983.
[81] D. C. Boes, F. A. Graybill, and A. M. Mood, “Introduction to the Theory of Statistics,” *Series in probabili*, 1974.
[82] P. H. A. Meyfroyt, “Parameter Estimation for Software Reliability Models,” 2012.
[83] A. Henningsen and O. Toomet, “maxLik: A package for maximum likelihood estimation in R,” *Computational Statistics*, vol. 26, no. 3, pp. 443–458, 2011.
[84] S. S. Gokhale and K. S. Trivedi, “Log-logistic software reliability growth model,” in *Proceedings of Third IEEE International High-Assurance Systems Engineering Symposium, 1998.*, 1998, pp. 34–41.
[85] Y. Miyazaki, A. Takanou, H. Nozaki, N. Nakagawa, and K. Okada, “Method to estimate parameter values in software prediction models,” *Information and Software Technology*, vol. 33, no. 3, pp. 239–243, Apr. 1991.
[86] S. Hwang and H. Pham, “Quasi-renewal time-delay fault-removal consideration in software reliability modeling,” *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans.*, vol. 39, no. 1, pp. 200–209, 2009.
[87] P. Liggesmeyer and M. Trapp, “Trends in embedded software engineering,” *IEEE Software*, vol. 26, no. 3, pp. 19–25, 2009.
[88] K. Grimm, “Software technology in an automotive company: major challenges,” in *Proceedings of the 25th international conference on Software Engineering*, pp. 498–503.
[89] N. Mellegård, M. Staron, and F. Törner, “A Light-Weight Defect Classification Scheme for Embedded Automotive Software and Its Initial Evaluation,” in *IEEE 23rd International Symposium on Software Reliability Engineering (ISSRE)*, 2012, pp. 261–270.
[90] S. Yamada, K. Tokuno, and S. Osaki, “Imperfect debugging models with fault introduction rate for software reliability assessment,” *International Journal of Systems Science*, vol. 23, no. 12, pp. 2241–2252, 1992.
[91] K. Ohishi, H. Okamura, and T. Dohi, “Gompertz software reliability model: Estimation algorithm and empirical validation,” *Journal of Systems and Software*, vol. 82, no. 3, pp. 535–543, 2009.
[92] S. Gamito, “Growth models and their use in ecological modelling: an application to a fish population,” *Ecological Modelling*, vol. 113, no. 1, pp. 83–94, 1998.

[93] Y. Yano, T. Oguma, H. Nagata, and S. Sasaki, “Application of logistic growth model to pharmacodynamic analysis of in vitro bactericidal kinetics,” *Journal of pharmaceutical sciences*, vol. 87, no. 10, pp. 1177–1183, 1998.

[94] S. R. Dalal and C. L. Mallows, “When should one stop testing software?,” *Journal of the American Statistical Association*, vol. 83, no. 403, pp. 872–879, 1988.

[95] P. K. Kapur and V. K. Bhalla, “Optimal release policies for a flexible software reliability growth model,” *Reliability Engineering & System Safety*, vol. 35, no. 1, pp. 49–54, 1992.

[96] T.-S. Quah, “Estimating software readiness using predictive models,” *Information Sciences*, vol. 179, no. 4, pp. 430–445, 2009.

[97] Y. K. Malaiya, N. Karunanithi, and P. Verma, “Predictability of software-reliability models,” *IEEE Transactions on Reliability*, vol. 41, no. 4, pp. 539–546, 1992.

[98] R. Rana, M. Staron, M. Niklas, C. Berger, J. Hansson, M. Nilsson, and F. Törner, “Evaluation of standard reliability growth models in the context of automotive software systems,” presented at the International Conference on on Product-Focused Software Process Improvement, Phaphos, Cyprus, 2013.

[99] W.-L. Wang, D. Pan, and M.-H. Chen, “Architecture-based software reliability modeling,” *Journal of Systems and Software*, vol. 79, no. 1, pp. 132–146, Jan. 2006.

[100] S. Yamada, *Software reliability modeling: fundamentals and applications*. Springer, 2014.

[101] C. A. Asad, M. I. Ullah, and M.-U. Rehman, “An approach for software reliability model selection,” in *Proceedings of the 28th Annual International Computer Software and Applications Conference, COMPSAC 2004.*, 2004, pp. 534–539.

[102] P. Popov, L. Strigini, J. May, and S. Kuball, “Estimating bounds on the reliability of diverse systems,” *IEEE Transactions on Software Engineering*, vol. 29, no. 4, pp. 345–359, Apr. 2003.

[103] H. Pham, L. Nordmann, and Z. Zhang, “A general imperfect-software-debugging model with S-shaped fault-detection rate,” *IEEE Transactions on Reliability*, vol. 48, no. 2, pp. 169–175, 1999.

[104] P. Van Der Spek and C. Verhoef, “Balancing Time-to-Market and Quality in Embedded Systems,” *Systems Engineering*, vol. 17, no. 2, pp. 166–192, 2014.

[105] M. Xie, G. Y. Hong, and C. Wohlin, “A practical method for the estimation of software reliability growth in the early stage of testing,” in *PROCEEDINGS The*

Eighth International Symposium On Software Reliability Engineering, 1997, pp. 116–123.
[106] C. Robson, *Real World Research: A Resource for Social Scientists and Practitioner-Researchers*. Wiley, 2002.
[107] M. Staron, J. Hansson, R. Feldt, A. Henriksson, W. Meding, S. Nilsson, and C. Hoglund, "Measuring and Visualizing Code Stability–A Case Study at Three Companies," in *Joint Conference of the 23rd International Workshop on Software Measurement and the 2013 Eighth International Conference on Software Process and Product Measurement (IWSM-MENSURA).*, 2013, pp. 191–200.
[108] U. Eklund, N. Jonsson, J. Bosch, and A. Eriksson, "A reference architecture template for software-intensive embedded systems," in *Proceedings of the WICSA/ECSA 2012 Companion Volume*, 2012, pp. 104–111.
[109] R. A. McGee, U. Eklund, and M. Lundin, "Stakeholder identification and quality attribute prioritization for a global Vehicle Control System," in *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*, 2010, pp. 43–48.
[110] H. Gustavsson and U. Eklund, "Architecting automotive product lines: Industrial practice," in *Software Product Lines: Going Beyond*, Springer, 2010, pp. 92–105.
[111] P. Tomaszewski, P. Berander, and L.-O. Damm, "From Traditional to Streamline Development—opportunities and challenges," *Software Process: Improvement and Practice*, vol. 13, no. 2, pp. 195–212, 2008.
[112] M. Bäumer, P. Seidler, R. Torkar, R. Feldt, P. Tomaszewski, and L.-O. Damm, "Predicting fault inflow in highly iterative software development processes: an industrial evaluation," in *Proceedings of the 19th IEEE International Symposium on Software Reliability Engineering: Industry Track*, 2008.
[113] J. D. Musa and K. Okumoto, "A logarithmic Poisson execution time model for software reliability measurement," in *Proceedings of the 7th international conference on Software engineering*, 1984, pp. 230–238.
[114] M. K. Taghi and B. A. Edward, "Logistic regression modeling of software quality," *International Journal of Reliability, Quality and Safety Engineering*, vol. 6, no. 04, pp. 303–317, 1999.
[115] P. Kapur, H. Pham, A. Gupta, and P. Jha, "Software Reliability Growth Models," in *Software Reliability Assessment with OR Applications*, Springer, 2011, pp. 49–95.

[116] C.-Y. Huang, S.-Y. Kuo, and M. R. Lyu, "An assessment of testing-effort dependent software reliability growth models," *Reliability, IEEE Transactions on*, vol. 56, no. 2, pp. 198–211, 2007.

[117] C.-Y. Huang, "Performance analysis of software reliability growth models with testing-effort and change-point," *Journal of Systems and Software*, vol. 76, no. 2, pp. 181–194, 2005.

[118] P. K. Kapur, D. N. Goswami, A. Bardhan, and O. Singh, "Flexible software reliability growth model with testing effort dependent learning process," *Applied Mathematical Modelling*, vol. 32, no. 7, pp. 1298–1307, Jul. 2008.

[119] H. Motulsky and A. Christopoulos, *Fitting Models to Biological Data Using Linear and Nonlinear Regression: A Practical Guide to Curve Fitting*. Oxford University Press, 2004.

[120] S. Hu, "Akaike information criterion," *Center for Research in Scientific Computation*, 2007.

[121] R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, and F. Törner, "Comparing between MLE and NLR estimation procedures for Applying SRGMs," presented at the IWSM-MENSURA 2013, Ankara, Turkey, 2013.

[122] H. Okamura, T. Dohi, and S. Osaki, "Software reliability growth model with normal distribution and its parameter estimation," in *Quality, Reliability, Risk, Maintenance, and Safety Engineering (ICQR2MSE), 2011 International Conference on*, 2011, pp. 411–416.

[123] R. Rana, M. Staron, C. Berger, J. Hansson, and M. Nilsson, "Analysing defect inflow distribution of automotive software projects," in *Proceedings of The 10th International Conference on Predictive Models in Software Engineering*, 2014.

[124] M. Xie, *Software reliability modelling*, vol. 1. World Scientific, 1991.

[125] P. N. Misra, "Software reliability analysis," *IBM Systems Journal*, vol. 22, no. 3, pp. 262–270, 1983.

[126] L. H. Putnam, "A general empirical solution to the macro software sizing and estimating problem," *IEEE Transactions on Software Engineering*, no. 4, pp. 345–361, 1978.

[127] M. Trachtenberg, "A general theory of software-reliability modeling," *IEEE Transactions on Reliability*, vol. 39, no. 1, pp. 92–96, 1990.

[128] L. H. Putnam and W. Myers, *Measures for excellence: reliable software on time, within budget*. Prentice Hall Professional Technical Reference, 1991.

[129] L. Zhao, “A new approach for software testability analysis,” in *Proceedings of the 28th international conference on Software engineering*, 2006, pp. 985–988.
[130] R. E. Mullen, “The lognormal distribution of software failure rates: origin and evidence,” in *Software Reliability Engineering, 1998. Proceedings. The Ninth International Symposium on*, 1998, pp. 124–133.
[131] R. E. Mullen, “The lognormal distribution of software failure rates: application to software reliability growth modeling,” in *Software Reliability Engineering, 1998. Proceedings. The Ninth International Symposium on*, 1998, pp. 134–142.
[132] Y. Zhou and J. Davis, “Open source software reliability model: an empirical approach,” *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 4, pp. 1–6, 2005.
[133] V. S. Kharchenko, O. M. Tarasyuk, V. V. Sklyar, and V. Y. Dubnitsky, “The method of software reliability growth models choice using assumptions matrix,” in *Computer Software and Applications Conference, 2002. COMPSAC 2002. Proceedings. 26th Annual International*, 2002, pp. 541–546.
[134] N. Karunanithi, Y. K. Malaiya, and D. Whitley, “Prediction of software reliability using neural networks,” in *Software Reliability Engineering, 1991. Proceedings., 1991 International Symposium on*, 1991, pp. 124–130.
[135] B. Littlewood and J. L. Verrall, “A Bayesian reliability growth model for computer software,” *Applied statistics*, pp. 332–346, 1973.
[136] L. Kuo, J. C. Lee, K. Choi, and T. Y. Yang, “Bayes inference for S-shaped software-reliability growth models,” *IEEE Transactions on Reliability*, vol. 46, no. 1, pp. 76–80, 87, Mar. 1997.
[137] M. Neil, N. Fenton, S. Forey, and R. Harris, “Using Bayesian belief networks to predict the reliability of military vehicles,” *Computing & Control Engineering Journal*, vol. 12, no. 1, pp. 11–20, 2001.
[138] S. K. Card, J. D. Mackinlay, and B. Shneiderman, *Readings in information visualization: using vision to think*. Morgan Kaufmann, 1999.
[139] A. Hora, N. Anquetil, S. Ducasse, M. Bhatti, C. Couto, M. T. Valente, and J. Martins, “Bug Maps: A Tool for the Visual Exploration and Analysis of Bugs,” in *2012 16th European Conference on Software Maintenance and Reengineering (CSMR)*, 2012, pp. 523–526.
[140] R. E. Garcia, M. C. F. de Oliveira, J. C. Maldonado, and M. Mendonça, “Visual analysis of data from empirical studies,” in *International Workshop on Visual Languages and Computing*, 2004.

[141] A. Mockus, R. T. Fielding, and J. Herbsleb, “A case study of open source software development: the Apache server,” in *Proceedings of the 22nd international conference on Software engineering*, 2000, pp. 263–272.
[142] A. Mockus, R. T. Fielding, and J. D. Herbsleb, “Two case studies of open source software development: Apache and Mozilla,” *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 11, no. 3, pp. 309–346, 2002.
[143] K. Herzig, S. Just, and A. Zeller, “It’s not a bug, it’s a feature: How misclassification impacts bug prediction,” in *Proceedings of the 2013 International Conference on Software Engineering*, 2013, pp. 392–401.
[144] G. Antoniol, K. Ayari, M. Di Penta, F. Khomh, and Y.-G. Guéhéneuc, “Is it a bug or an enhancement?: a text-based approach to classify change requests,” in *Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds*, 2008, p. 23.
[145] E. W. Weisstein, “Likelihood -- from Wolfram MathWorld.” [Online]. Available: http://mathworld.wolfram.com/Likelihood.html. [Accessed: 02-Jul-2014].
[146] K. P. Burnham and D. R. Anderson, *Model selection and multimodel inference: a practical information-theoretic approach*. Springer, 2002.
[147] J. J. Dziak, D. L. Coffman, S. T. Lanza, and R. Li, “Sensitivity and specificity of information criteria,” *The Methodology Center and Department of Statistics, Penn State, The Pennsylvania State University*, 2012.
[148] N. J. Cox, “Speaking Stata: density probability plots,” *Stata Journal*, vol. 5, no. 2, pp. 259–273, 2005.
[149] B. Baudry, Y. Le Traon, and G. Sunyé, “Testability analysis of a UML class diagram,” in *Software Metrics, 2002. Proceedings. Eighth IEEE Symposium on*, 2002, pp. 54–63.
[150] G. Wittig and G. Finnie, “Estimating software development effort with connectionist models,” *Information and Software Technology*, vol. 39, no. 7, pp. 469–476, 1997.
[151] A. Asthana and J. Olivieri, “Quantifying software reliability and readiness,” in *Communications Quality and Reliability, 2009. CQR 2009. IEEE International Workshop Technical Committee on*, 2009, pp. 1–6.
[152] A. E. Atwater, M. J. Safrit, T. A. Baumgartner, and C. West, *Reliability theory*. American Alliance for Health, Physical Education, and Recreation, 1976.
[153] J. H. Bailey and R. A. Kowalski, “Reliability-growth analysis for an Ada-coding process,” in *Reliability and Maintainability Symposium, 1992. Proceedings., Annual*, 1992, pp. 280–284.

[154] R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, and F. Torner, “Evaluating long-term predictive power of standard reliability growth models on automotive systems,” in *Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on*, 2013, pp. 228–237.
[155] M. Staron and W. Meding, “Short-term defect inflow prediction in large software project-an initial evaluation,” in *International Conference on Empirical Assessment in Software Engineering (EASE)*, 2007.
[156] M. Staron and W. Meding, “Predicting Monthly Defect Inflow in Large Software Projects–An Industrial Case Study,” in *Industry Track Proceedings of the 27 International Symposium on Software Reliability Engineering*, 2007.
[157] M. Staron, W. Meding, and K. Palm, “Release Readiness Indicator for Mature Agile and Lean Software Development Projects,” in *Agile Processes in Software Engineering and Extreme Programming*, Springer, 2012, pp. 93–107.
[158] C.-Y. Huang and S.-Y. Kuo, “Analysis of incorporating logistic testing-effort function into software reliability modeling,” *IEEE Transactions on Reliability*, vol. 51, no. 3, pp. 261–270, 2002.
[159] Y. W. Lee, D. M. Strong, B. K. Kahn, and R. Y. Wang, “AIMQ: a methodology for information quality assessment,” *Information & management*, vol. 40, no. 2, pp. 133–146, 2002.
[160] M. Staron and W. Meding, “Ensuring reliability of information provided by measurement systems,” in *Software Process and Product Measurement*, Springer, 2009, pp. 1–16.
[161] R. Rana, M. Staron, C. Berger, J. Hansson, W. Meding, and C. Höglund, “Selecting software reliability growth models and improving their predictive accuracy using historical projects data,” *Submitted to Journal of Systems and Software*, 2014.
[162] A. Güneş Koru and J. Tian, “An empirical comparison and characterization of high defect and high complexity modules,” *Journal of Systems and Software*, vol. 67, no. 3, pp. 153–163, 2003.
[163] N. E. Fenton and N. Ohlsson, “Quantitative analysis of faults and failures in a complex software system,” *Software Engineering, IEEE Transactions on*, vol. 26, no. 8, pp. 797–814, 2000.
[164] C. Andersson and P. Runeson, “A Replicated Quantitative Analysis of Fault Distributions in Complex Software Systems,” *IEEE Transactions on Software Engineering*, vol. 33, no. 5, pp. 273–286, May 2007.

[165] T. Galinac Grbac, P. Runeson, and D. Huljenić, "A second replicated quantitative analysis of fault distributions in complex software systems," *Software Engineering, IEEE Transactions on*, vol. 39, no. 4, pp. 462–476, 2013.
[166] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Where the bugs are," in *ACM SIGSOFT Software Engineering Notes*, 2004, vol. 29, pp. 86–96.
[167] R. M. Bell, T. J. Ostrand, and E. J. Weyuker, "Looking for Bugs in All the Right Places," in *Proceedings of the 2006 International Symposium on Software Testing and Analysis*, New York, NY, USA, 2006, pp. 61–72.
[168] E. J. Weyuker, T. J. Ostrand, and R. M. Bell, "Comparing the effectiveness of several modeling methods for fault prediction," *Empir Software Eng*, vol. 15, no. 3, pp. 277–295, Jun. 2010.
[169] J. Zheng, L. Williams, N. Nagappan, W. Snipes, J. P. Hudepohl, and M. A. Vouk, "On the value of static analysis for fault detection in software," *Software Engineering, IEEE Transactions on*, vol. 32, no. 4, pp. 240–253, 2006.
[170] N. Nagappan and T. Ball, "Static analysis tools as early indicators of pre-release defect density," in *Proceedings of the 27th international conference on Software engineering*, 2005, pp. 580–586.
[171] R. Subramanyam and M. S. Krishnan, "Empirical analysis of ck metrics for object-oriented design complexity: Implications for software defects," *Software Engineering, IEEE Transactions on*, vol. 29, no. 4, pp. 297–310, 2003.
[172] N. Nagappan, T. Ball, and A. Zeller, "Mining metrics to predict component failures," in *Proceedings of the 28th international conference on Software engineering*, 2006, pp. 452–461.
[173] S. Kim, T. Zimmermann, E. J. Whitehead Jr, and A. Zeller, "Predicting faults from cached history," in *Proceedings of the 29th international conference on Software Engineering*, 2007, pp. 489–498.
[174] N. Nagappan and T. Ball, "Use of relative code churn measures to predict system defect density," in *Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on*, 2005, pp. 284–292.
[175] W. Snipes, B. Robinson, and E. Murphy-Hill, "Code hot spot: A tool for extraction and analysis of code change history," in *Software Maintenance (ICSM), 2011 27th IEEE International Conference on*, 2011, pp. 392–401.
[176] N. Nagappan and T. Ball, "Using software dependencies and churn metrics to predict field failures: An empirical case study," in *First International Symposium on Empirical Software Engineering and Measurement, ESEM 2007.*, 2007, pp. 364–373.

[177] S. Kim, E. J. Whitehead, and Y. Zhang, “Classifying software changes: Clean or buggy?,” *Software Engineering, IEEE Transactions on*, vol. 34, no. 2, pp. 181–196, 2008.
[178] T. Zimmermann and N. Nagappan, “Predicting defects using network analysis on dependency graphs,” in *Proceedings of the 30th international conference on Software engineering*, 2008, pp. 531–540.
[179] M. D’Ambros, M. Lanza, and R. Robbes, “An extensive comparison of bug prediction approaches,” in *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*, 2010, pp. 31–41.
[180] R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, F. Törner, W. Meding, and C. Höglund, “Selecting software reliability growth models and improving their predictive accuracy using historical projects data,” *Journal of Systems and Software*, vol. 98, pp. 59–78, Dec. 2014.
[181] R. N. Charette, *This Car Runs on Code*. http://spectrum.ieee.org/green-tech/advanced-cars/this-car-runs-on-code, 2009.
[182] H. Fennel, S. Bunzel, H. Heinecke, rgen Bielefeld, Jü, S. Fü, rst, K.-P. Schnelle, W. Grote, N. Maldener, T. Weber, F. Wohlgemuth, and others, “Achievements and exploitation of the AUTOSAR development partnership,” *Convergence*, vol. 2006, p. 10, 2006.
[183] B. Boehm and V. R. Basili, “Defect Reduction Top 10 List,” *Computer*, pp. 135–137, 2001.
[184] N. Mellegård, M. Staron, and F. Törner, “A Light-Weight Defect Classification Scheme for Embedded Automotive Software Development,” 2013.
[185] M. Hillenbrand, M. Heinz, N. Adler, K. D. Müller-Glaser, J. Matheis, and C. Reichmann, “ISO/DIS 26262 in the context of electric and electronic architecture modeling,” in *Architecting Critical Systems*, Springer, 2010, pp. 179–192.
[186] B. Schätz, “Certification of Embedded Software–Impact of ISO DIS 26262 in the Automotive Domain,” in *Leveraging Applications of Formal Methods, Verification, and Validation*, Springer, 2010, pp. 3–3.
[187] M. C. Hsueh, T. K. Tsai, and R. K. Iyer, “Fault injection techniques and tools,” *Computer*, vol. 30, no. 4, pp. 75–82, 1997.
[188] R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, and F. Törner, “Increasing Efficiency of ISO 26262 Verification and Validation by Combining Fault Injection and Mutation Testing with Model Based Development,” presented at the 8th International Conference on Software Engineering and Applications, Reykjavík, Iceland, 2013.

[189] N. Mellegård, M. Staron, and F. Törner, *A light-weight defect classification scheme for embedded automotive software and its initial evaluation*. 2012.

[190] R. Megen and D. B. Meyerhoff, "Costs and benefits of early defect detection: experiences from developing client server and host applications," *Software Quality Journal*, vol. 4, no. 4, pp. 247–256, 1995.

[191] R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, and F. Törner, "Improving Fault Injection in Automotive Model Based Development using Fault Bypass Modeling," presented at the Submitted To: 2nd Workshop on Software-Based Methods for Robust Embedded Systems, Informatik 2013, Koblenz, Germany, 2013.

[192] R. Svenningsson, J. Vinter, H. Eriksson, and M. Törngren, "MODIFI: a MODel-implemented fault injection tool," *Computer Safety, Reliability, and Security*, pp. 210–222, 2010.

[193] H. Ziade, R. A. Ayoubi, R. Velazco, and others, "A survey on fault injection techniques," *The International Arab Journal of Information Technology*, vol. 1, no. 2, pp. 171–186, 2004.

[194] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE Transactions on Software Engineering*, vol. 37, no. 5, pp. 649–678, 2011.

[195] J. H. Andrews, L. C. Briand, and Y. Labiche, "Is mutation an appropriate tool for testing experiments?[software testing]," in *Proceedings. 27th International Conference on Software Engineering, ICSE 2005.*, 2005, pp. 402–411.

[196] R. A. DeMillo, R. J. Lipton, and F. G. Sayward, "Hints on test data selection: Help for the practicing programmer," *Computer*, vol. 11, no. 4, pp. 34–41, 1978.

[197] A. J. Offutt, "Investigations of the software testing coupling effect," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 1, no. 1, pp. 5–20, 1992.

[198] A. Joshi and M. P. E. Heimdahl, *Model-based safety analysis of simulink models using SCADE design verifier*. 2005.

[199] R. Kakade, M. Murugesan, B. Perugu, and M. Nair, "Model-Based Development of Automotive Electronic Climate Control Software," *Modelling Foundations and Applications*, pp. 144–155, 2010.

[200] J. Vinter, L. Bromander, P. Raistrick, and H. Edler, "FISCADE - A Fault Injection Tool for SCADE Models," in *3rd Institution of Engineering and Technology Conference on Automotive Electronics, 2007.*, 2007, pp. 1–9.

[201] A. Plummer, “Model-in-the-loop testing,” *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, vol. 220, no. 3, pp. 183–199, 2006.
[202] A. Brillout, N. He, M. Mazzucchi, D. Kroening, M. Purandare, P. Rümmer, and G. Weissenbacher, “Mutation-based test case generation for simulink models,” in *Formal Methods for Components and Objects*, 2010, pp. 208–227.
[203] C. Berger, M. Chaudron, R. Heldal, O. Landsiedel, and E. M. Schiller, “Model-based, Composable Simulation for the Development of Autonomous Miniature Vehicles.”
[204] C. Berger and J. Hansson, “COTS-Architecture with a Real-Time OS for a Self-Driving Miniature Vehicle,” in *Proceedings of Workshop ASCoMS (Architecting Safety in Collaborative Mobile Systems) of the 32nd International Conference on Computer Safety, Reliability and Security*, 2013.
[205] A. Bertolino, “Software testing research: Achievements, challenges, dreams,” in *2007 Future of Software Engineering*, 2007, pp. 85–103.
[206] M. J. Harrold, “Testing: a roadmap,” in *Proceedings of the Conference on the Future of Software Engineering*, 2000, pp. 61–72.
[207] B. Boehm, *Industrial software metrics top 10 list*. IEEE COMPUTER SOC 10662 LOS VAQUEROS CIRCLE, PO BOX 3014, LOS ALAMITOS, CA 90720-1264, 1987.
[208] T. Menzies, K. Ammar, A. Nikora, and J. DiStefano, “How simple is software defect detection,” *Submitted to the Emprical Software Engineering Journal*, 2003.
[209] P. Legris, J. Ingham, and P. Collerette, “Why do people use information technology? A critical review of the technology acceptance model,” *Information & management*, vol. 40, no. 3, pp. 191–204, 2003.
[210] H. Van der Heijden, “Factors influencing the usage of websites: the case of a generic portal in The Netherlands,” *Information & Management*, vol. 40, no. 6, pp. 541–549, 2003.
[211] D. Zhang and J. J. Tsai, “Machine learning and software engineering,” *Software Quality Journal*, vol. 11, no. 2, pp. 87–119, 2003.
[212] P. Attewell, “Technology diffusion and organizational learning: The case of business computing,” *Organization Science*, vol. 3, no. 1, pp. 1–19, 1992.
[213] I. Ajzen and M. Fishbein, “Understanding attitudes and predicting social behaviour,” 1980.
[214] G. G. Pijpers, T. Bemelmans, F. J. Heemstra, and K. A. van Montfort, “Senior executives’ use of information technology,” *Information and Software Technology*, vol. 43, no. 15, pp. 959–971, 2001.

[215] K. C. Yang, “Exploring factors affecting the adoption of mobile commerce in Singapore,” *Telematics and informatics*, vol. 22, no. 3, pp. 257–277, 2005.
[216] P. Y. Chau and K. Y. Tam, “Factors Affecting the Adoption of Open Systems: An Exploratory Study.,” *Mis Quarterly*, vol. 21, no. 1, 1997.
[217] S. Zhang and M. J. Zaki, “Mining multiple data sources: local pattern analysis,” *Data Mining and Knowledge Discovery*, vol. 12, no. 2–3, pp. 121–125, 2006.
[218] T. M. Mitchell, “Machine learning and data mining,” *Communications of the ACM*, vol. 42, no. 11, pp. 30–36, 1999.
[219] A. C. Edmondson, A. B. Winslow, R. M. Bohmer, and G. P. Pisano, “Learning how and learning what: Effects of tacit and codified knowledge on performance improvement following technology adoption,” *Decision Sciences*, vol. 34, no. 2, pp. 197–224, 2003.
[220] S. Sonnenburg, M. L. Braun, C. S. Ong, S. Bengio, L. Bottou, G. Holmes, Y. LeCun, K.-R. Müller, F. Pereira, and C. E. Rasmussen, “The need for open source software in machine learning.,” *Journal of Machine Learning Research*, vol. 8, no. 10, 2007.
[221] M. Li and C. S. Smidts, “A ranking of software engineering measures based on expert opinion,” *IEEE Transactions on Software Engineering*, vol. 29, no. 9, pp. 811–824, 2003.
[222] N. Fenton, M. Neil, W. Marsh, P. Hearty, D. Marquez, P. Krause, and R. Mishra, “Predicting software defects in varying development lifecycles using Bayesian nets,” *Information and Software Technology*, vol. 49, no. 1, pp. 32–43, 2007.
[223] R. Rana, M. Staron, and M. Nilsson, “A framework for adoption of machine learning in industry for software defect prediction,” presented at the submitted to ICSOFT-EA, 2014, Vienna, Austria, 2014.
[224] L. G. Wallace and S. D. Sheetz, “The adoption of software measures: A technology acceptance model (TAM) perspective,” *Information & Management*, vol. 51, no. 2, pp. 249–259, Mar. 2014.
[225] B. C. Hardgrave and R. A. Johnson, “Toward an information systems development acceptance model: the case of object-oriented systems development,” *Engineering Management, IEEE Transactions on*, vol. 50, no. 3, pp. 322–336, 2003.
[226] D. Gefen and D. W. Straub, “Gender Differences in the Perception and Use of E-Mail: An Extension to the Technology Acceptance Model.,” *MIS quarterly*, vol. 21, no. 4, 1997.

[227] M. Y. Yi and Y. Hwang, “Predicting the use of web-based information systems: self-efficacy, enjoyment, learning goal orientation, and the technology acceptance model,” *International journal of human-computer studies*, vol. 59, no. 4, pp. 431–449, 2003.
[228] T. Menzies and A. Marcus, “Automated severity assessment of software defect reports,” in *IEEE International Conference on Software Maintenance, ICSM 2008.*, 2008, pp. 346–355.
[229] E. Mendes, N. Mosley, and S. Counsell, “Investigating early web size measures for web cost estimation,” in *Proceedings of EASE’2003 Conference, Keele*, 2003, pp. 1–22.
[230] T. Menzies, B. Caglayan, E. Kocaguneli, J. Krall, F. Peters, and B. Turhan, “The promise repository of empirical software engineering data,” *West Virginia University, Department of Computer Science*, 2012.