Defect Prediction over Software Life Cycle in Automotive Domain
State of the Art and Road Map for Future

Rakesh Rana[1], Miroslaw Staron[1], Jörgen Hansson[1] and Martin Nilsson[2]
[1]*Computer Science & Engineering, Chalmers, University of Gothenburg, Gothenburg, Sweden*
[2]*Volvo Car Group, Gothenburg, Sweden*

Keywords: Defect Prediction, Software Life Cycle, Automotive Software, Test Resource Allocation, Release Readiness.

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.

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 (Broy, 2006). With premium segment cars today carrying about 100 million lines of code, which is more than fighter jets and airliners (Charette, 2009). 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 resouces to meet the desried goals of projects. The different methods of software defect analysis and predictions have different characterstics. 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 predicitons. 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 efficently will enable software designers and developers to use this information to design more robust and user friendly features and functions.

2 BACKGROUND

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 (Boehm

Rana R., Staron M., Hansson J. and Nilsson M..
Defect Prediction over Software Life Cycle in Automotive Domain - State of the Art and Road Map for Future.
DOI: 10.5220/0005099203770382
In *Proceedings of the 9th International Conference on Software Engineering and Applications* (ICSOFT-EA-2014), pages 377-382
ISBN: 978-989-758-036-9

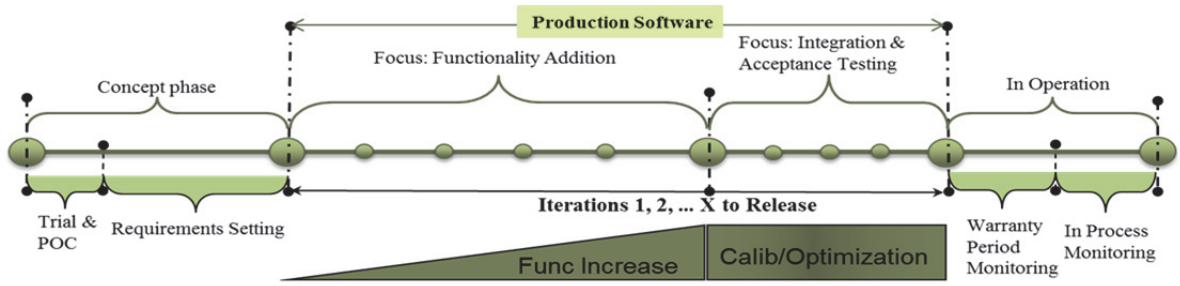

Figure 1: Time Line of Automotive Software Development Life Cycle.

1988) and approaches based on V-model (Dieterle, 2005; ISO 2011).

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 1) 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 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 (Staron and Meding, 2008). Long term predictive power of SRGMs within the automotive domain was studied in authors earlier works (Rana, Staron, Mellegård, et al. 2013; Rana, Staron, Berger, et al., 2013), 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 (Khoshgoftaar and Allen, 1999) used logistic regression for classifying modules as fault-prone, while Menzies, Greenwald and Frank (Menzies, Greenwald, and Frank 2007) 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 (Gondra, 2008) and (Ceylan, Kutlubay, and Bener 2006).

Fenton and Neil (N. E. Fenton and Neil, 1999) 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 (N. Fenton et al., 2008).

Our study complements earlier studies in defect predictions by illustrating when different methods of SDP are most appropriate over a software development life cycle.

3 DEFECTS PREDICITON 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 2 and the characteristics of each method are summarized in Table 1. 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.

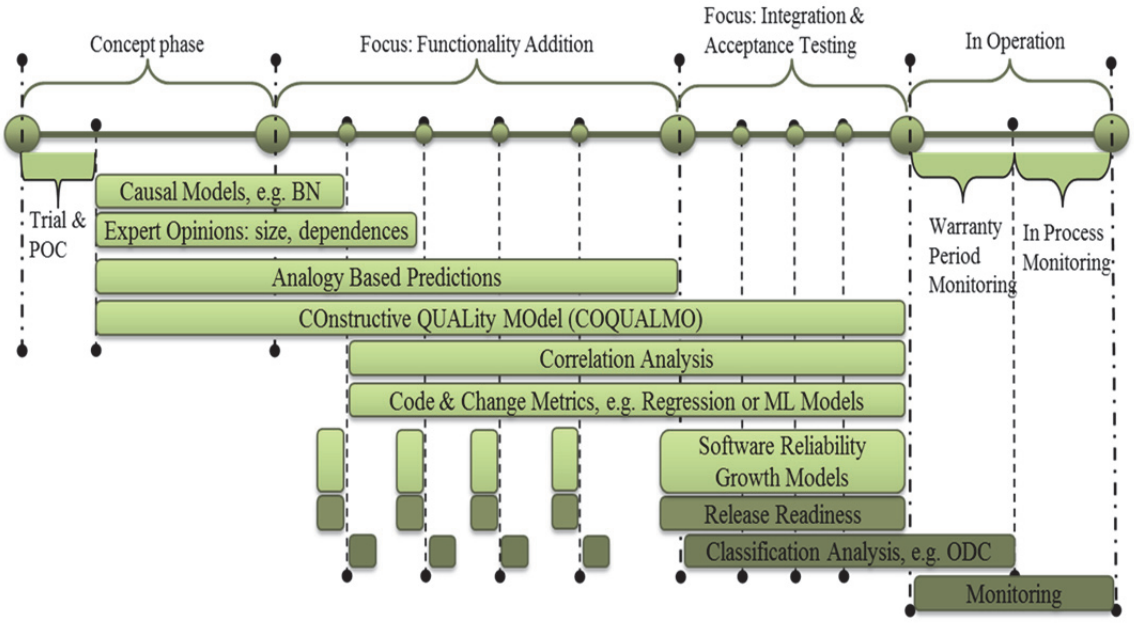

Figure 2: Methods for software defect predictions, applicability over SW life cycle in automotive domain.

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.

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 2 summarizes the levels and appropriate applications for each model type. The granularity level at which analysis can be done are:

Table 1: 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.

- 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 2: 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

4 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.

5 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.

ACKNOWLEDGEMENTS

The research presented here is done under the VISEE project which is funded by Vinnova and Volvo Cars jointly under the FFI programme (VISEE, Project No: DIARIENR: 2011-04438).

REFERENCES

1044-2009-IEEE Standard Classification for Software Anomalies. 2010.

Boehm, Barry W. 1988. “A Spiral Model of Software Development and Enhancement.” *Computer* 21 (5): 61–72.

Broy, Manfred. 2006. “Challenges in Automotive Software Engineering.” In *Proceedings of the 28th International Conference on Software Engineering*, 33–42.

Ceylan, Evren, F. Onur Kutlubay, and Ayse Basar Bener. 2006. “Software Defect Identification Using Machine Learning Techniques.” In *32nd EUROMICRO Conference on Software Engineering and Advanced Applications, SEAA’06.*, 240–47. IEEE.

Charette, Robert N. 2009. “This Car Runs on Code.” *IEEE Spectrum* 46 (3): 3.

Dieterle, Werner. 2005. “Mechatronic Systems: Automotive Applications and Modern Design Methodologies.” *Annual Reviews in Control* 29 (2): 273–77.

Fenton, N.E., and M. Neil. 1999. “A Critique of Software Defect Prediction Models.” *IEEE Transactions on Software Engineering* 25 (5): 675–89. doi:10.1109/32.815326.

Fenton, Norman, Martin Neil, William Marsh, Peter Hearty, Łukasz Radliński, and Paul Krause. 2008. “On the Effectiveness of Early Life Cycle Defect Prediction with Bayesian Nets.” *Empirical Software Engineering* 13 (5): 499–537. doi:10.1007/s10664-008-9072-x.

Gondra, Iker. 2008. “Applying Machine Learning to Software Fault-Proneness Prediction.” *Journal of Systems and Software* 81 (2): 186–95.

ISO. 2011. “International Standard-ISO 26262-Road Vehicles-Functional Safety”. International Organization for Standardization.

Khoshgoftaar, Taghi M., and Edward B. Allen. 1999. “Logistic Regression Modeling of Software Quality.” *International Journal of Reliability, Quality and Safety Engineering* 6 (04): 303–17.

Menzies, Tim, Jeremy Greenwald, and Art Frank. 2007. “Data Mining Static Code Attributes to Learn Defect Predictors.” *IEEE Transactions on Software Engineering* 33 (1): 2–13.

Rana, Rakesh, Miroslaw Staron, Christian Berger, Jörgen Hansson, Martin Nilsson, and Fredrik Törner. 2013. “Evaluating Long-Term Predictive Power of Standard Reliability Growth Models on Automotive Systems.” In Pasadena, CA, USA.

Rana, Rakesh, Miroslaw Staron, Niklas Mellegård, Christian Berger, Jörgen Hansson, Martin Nilsson, and Fredrik Törner. 2013. “Evaluation of Standard Reliability Growth Models in the Context of Automotive Software Systems.” In *Product-Focused Software Process Improvement*, 324–29. Springer.

Staron, Miroslaw, and Wilhelm Meding. 2008. “Predicting Weekly Defect Inflow in Large Software Projects Based on Project Planning and Test Status.” *Information and Software Technology* 50 (7): 782–96.