Fraunhofer Institut
Experimentelles
Software Engineering

Reliability Prediction Models for Product Lines - A Brief Overview

Author:
Bernd Freimut

In part supported by
ESAPS Project,
Eureka Σ! 1023 Programme,
ITEA Project 99005

IESE-Report No. 046.00/E
Version 1.0
July 16, 2000

A publication by Fraunhofer IESE

Fraunhofer IESE is an institute of the Fraunhofer Gesellschaft.
The institute transfers innovative software development techniques, methods and tools into industrial practice, assists companies in building software competencies customized to their needs, and helps them to establish a competetive market position.

Fraunhofer IESE is directed by
Prof. Dr. Dieter Rombach
Sauerwiesen 6
D-67661 Kaiserslautern

Executive Summary

When planning product lines, many criteria have to be taken into account in order to decide, for example, whether a specific feature should be developed for reuse or specifically for a single system. One criterion is the reliability of the entire product line. In order to take this criterion explicitly into account, reliability prediction models are required. These models aim at predicting the reliability of a feature based on the starting situation (e.g., whether the feature has to be implemented from scratch, whether existing code can be adapted or reengineered) and the development approach (e.g., develop for reuse or develop system specific).

This report works towards the definition of a reliability prediction model. Specifically, a brief survey of existing reliability prediction models is given. However, none of these approaches is readily implementable in the context of planning a product line. Therefore, as a first step a measurement program is proposed, in which the reliability of features is characterized. For this purpose, several measures resp. models for reliability are presented together with a list of factors that were reported in the literature to have an impact on the reliability of a software system.

Table of Contents

1	**Objective**	1
2	**Measuring Reliability**	2
3	**State-of-the-Art**	4
3.1	Software Re-use	4
3.2	Software Reliability Engineering	4
3.3	Quality Prediction Models	5
3.3.1	Impact of CMM on software defect reduction [7]	5
3.3.2	Bayesian Belief Networks [4]	6
3.3.3	COQUALMO [2]	6
3.3.4	Cobra [1]	7
3.3.5	Open issues	8
4	**Proposal for future activities**	9
5	**References**	12

1 Objective

The objective of this report is to work towards a method for building reliability prediction models in the framework of product line approaches. In the planning phase of a product line a decision has to be made whether a particular feature *f* should be implemented system specific or whether it should be developed for reuse so that the feature can be instantiated in several systems.

One criterion for this decision is how the reliability of the product line is affected in each case. Intuitively it can be argumented that a reusable feature might show higher reliability as defects are faster deteced and removed in multiple instantiations. On the other hand a reusable feature might also be more complex and thus more defect-prone.

Taking into account this criterion during the planning of a product line allows to assess, which impact a decision (e.g., whether to develop a feature for reuse or specifically for one system) has on the quality of the product line. Ideally, this decision should be made so that the number of failures in the product line (in a specified period of time) is minimized, the Mean-Time-To-Failure is maximized, or that the defect density is minimized.

The remaining sections are organized as follows: Section 2 discusses how to measure software reliability. Section 3 briefly surveys the state-of-the art in software reuse, software reliability engineering, and software quality prediction to identify existing solutions that might be applicable for the objective listed above. Section 4 lists proposals for the future work.

2 Measuring Reliability

The emphasis of a reliability prediction model is the software attribute reliability. Thus, this section briefly introduces several ways of operationalizing and measuring"reliability".

Software reliability: Software reliability is defined as the probability of failure-free software operation for a specified period of time in a specified environment [8].

Failure intensity: The expected number of failures per time unit or natural unit. E.g. 0.1 failures/hr, 2 failures per 1000 transactions.

MTTF: The expected length of an interval between two failures.

All these definitions are interchangable. Preferable in the domain of software reliability engineering is, however, failure intensity. This is because (a) a probability is hard to interpret and (b) the MTTF often does not exist [11]. On the other hand, MTTF is attractive as well since larger indicates better and it has an intuitive meaning.

In these definitions two aspects are important to consider: (a) the environment and (b) the time.

Environment: Different users might tend to use a system in a different way. Therefore, a reliability statement for one user might be different from the reliability statement of another user. Thus, the user (i.e., the environment) has explicitly to be taken into account when reliability statements are given. The environment can, for example, be modelled by the operational profile [10]. The operational profile is a listing of the functions provided by a system along with the probability with which each function is executed.

Time: as time is one critical factor in the definition of software reliability its measurement should be performed with care. Generally, the following types of time can be identified: the *execution time* for a software system is the CPU time that is actually spent by the computer in executing the system; the *calendar time* is the time people normally experience in term of years, months, weeks, days, etc; and the *clock time* is the elapsed time from start to end of computer execution in running the system [8]. Finally it is also possible to express time in terms of natural units where a natual unit is a unit that is related to the output of a software-based product and hence the amount of processing done. Examples could be pages of output (e.g., 1 failure per 1000 pages printed), transactions such as reservations, sales or deposits, and telephone calls. [10]

Often field failures are reported with a description of the failure so that it can be fixed. No execution time, which is preferable to make accurate predictions with software reliability models, is recorded. Therefore, many organisations for wide-distribution software (i.e. installed at different sites with many users and different operational profiles) or commercial systems (where the user profile is usually not known) focus on the number of remaining faults or the defect density as a measure for software quality [8] [5]. MTTF is rather used in safety-crucial systems such as air traffic control, avionics, weapons [5] and in telecommunications.

Thus, a measure for reliability would be the fault density:

fault density: The number of faults detected in the system in a specified time period per unit of size.

The fault density can be determined based on defects that are detected in the field. In this case the number of faults can be approximated by the number of unique failure causes [5]. It is also possible to determine the fault density in-process. In this case all faults detected in system test would be used to determine the fault density. This in-process fault density can be expected to be correlated with the fault density experienced in field and is earlier available.

As alternative measure the PUM: problems per user month [5] defined as the total problems that the customers reported (true defects and non-defect-oriented problems) for a time period over the total number of license month of the software during the period. It can be computed for each month after release and averaged yearly.

3 State-of-the-Art

3.1 Software Re-use

The first step was to get an overview on what has already been done in the domain of software re-use and its impact on software relliability. To do so, a literature search was performed for work that has been published with variation of the keywords “software reliability” and “software reuse” in the literature database INSPEC. This search, however, did not retrieve search results seeming interesting for the problem at hand.

Also in the standard literature on software reuse, the impact of re-use on reliability is rather informally described. For example [6], mentions that reuse has an effect on software reliability measured in terms of the MTTF. Additionally, a measure for MTTF is provided. However, no impacting factors or methods to predict reliability is given.

3.2 Software Reliability Engineering

The next step was to look whether in the domain of software reliability engineering existing results can be used as a basis for the problem at hand.

Software reliability engineering is a sub-discipline of software engineering that is centered around the software attribute reliability. Especially software reliability measurement is concerned with estimating and prediting software reliability.

However, the work in this field is mainly concerned with estimating/predicting the reliability of a software system *during the system testing phase*. In this phase defect data (mainly the execution time of the software when a failure occured) is used to predict/estimate the reliability. For our purpose such models are not adqequate as we (a) want to make predictions earlier than the system test phase (i.e., the planning phase of a product line) and (b) do not have defect data for the feature to be planned and developed available.

In the context of our objective, the work in this discipline could have been interesting to answer the question: “Based on a given defect density, can we predict the number of failures/MTTF if a feature is used in several systems?”

For the specific problem, how the reliability changes if a component is instantianted several times in the context of a product line, no answer is given. Rather, a system (or component) installed at several sites is seen as one component undergoing more intensive testing. Thus, a reliablilty prediction is made by interleaving the failures from the different sites [11]. Thus, virtually the several physical components are treated as one logical component

In recent conferences (ISSRE99) specific tracks for 'Early reliability prediction' were offered which, however, did not go beyond the testing phase.

These negative search results, however, do not rule out that there might be something relevant.

3.3 Quality Prediction Models

A next step (actually a by-product of the survey described before) was to identify modelling approaches that aim also at building reliability prediction models prior to the testing phase and with data potentially available at the planning phase of a product line.

Three aspects are important in building such a model: 1) identifying factors that have an impact on reliability, 2) specifying a functional or causal relationship between these factors and reliability, 3) evaluating the accuracy of the predicion model.

In the following subsections the identified approaches are briefly outlined.

3.3.1 Impact of CMM on software defect reduction [7]

The purpose of this work and the resulting model was to investigate the impact of the CMM key process areas on the number of field defects. Thus, the impacting factors on reliability were formulated in terms of the CMM KPA goal questions, which were measured on a ordinal frequency scale {almost always, frequently, about half, occasionally, rarely if ever}. All the answers to the questions were numerically coded and then aggregated to obtain one score per project (defined as PROCESS). Additionally for each project the number of field defects in the first 24 month after release, the size of the product, and the technical capability of the developers (measured subjectively on a 5-point Likert-scale ranging from very low capability to very high capability) was determined.

With this data set a regression model was built

$$\mathrm{DEFECTS} = e^{a_1} \times \mathrm{SIZE}^{a_2} \times \mathrm{PC}^{a_3} \times \mathrm{PROCESS}^{a_4}$$

The purpose of this model was, however, not to predict the number of defects but to test whether the PROCESS has an impact.

Discussion: as the purpose of the model is not prediction and it models all process impacts by one variable, using it for the problem at hand might not be appropriate. Moreover, since it is likely that the key process areas are the same for one organization it is unlikely that they allow to distinguish between development system-specific or development for reuse.

3.3.2 Bayesian Belief Networks [4]

One approach to predict defect counts in software modules based on expert opinion and empirical data was made by means of Bayesian Belief Networks (BBNs). A BBN is a graphical network together with an associated set of probability tables. The nodes in the network represent uncertain variables and the arcs between them represent cause/relevance relationships between the variables. The probability tables for each node provide the probabilities of each state of the variable for that node. For nodes without parents these probabilities are just the marginal probabilities. For nodes with parents these are conditional probabilities for each combination of parent state values.

Discussion: Although interesting as this method can incorporate expert opinion, which is at the planning time the most available type of data, it is unclear how the probabilities can be determined. Moreover, a lot of effort is likely to be put into the development of the network where the question is first: what factors affect the number of defects?

3.3.3 COQUALMO [2]

A quality prediction model for defect density that has been proposed recently in the context of the COCOMO model and is still undergoing research is he COQUALMO model [3][2].

The objective of this model is to predict at the planning phase the number of defects in a system. This is achieved by two sub-models. A Defect Introduction (DI) Model estimates the number of defects that are inserted in the system and a Defect Removal (DR) Model estimates the number of defects removed by several defect removal activities.

The impacting factors in the DI model are the COCOMO factors from the cost estimation model COCOMO II and a size estimate. Each of the COCOMO factors is assigned a value from {very high, high, nominal, low, very low}. Experts estimated using the Delphi-Method multiplicative factors how the level of the factors increases or decreases the number of defects compared to a nominal project (i.e., a project in which the factor is nominal).

The impacting factors in the DR model are the estimated number of introduced defects from the DI model, the software size estimate, and the defect removal profile levels. The latter are parameters in terms of percentage of defects found by the types of defect removal activities.

The model can then be described as

$$\text{DEFECTS} = \text{DI-DR}$$

with

$$\text{DI} = \sum_{j \in \{\text{Analysis,Design,Code}\}} A_j \times \text{SIZE}^{B_j} \times \prod_{i=1}^{21} \text{DI-Driver}_{ij}$$

where A_j is the basline adjustment factor, B_j is initially set to 1 and accounts for economies/diseconomies of scale, and DI-Driver$_{ij}$ denotes the impacting factors, and

$$\text{DR}_j = C_j \times \text{DI}_j \times \prod_i (1 - \text{DR-driver}_{ij})$$

where C_j is a baseline constant for the jth artifact and DR-Driver$_{ij}$ the defect removal fraction for defect removal profile i and artifact type j (Analysis, Design, Code).

However the models has only been proposed so far, factors for DR-Drivers and DI-Drivers (i.e., the COCOMO factors) have been assessed with experts from multiple organizations. However, no validation of the model has taken place so far.

Discussion: the factors for the DI-model contain factors related to the product to be developed and the personnel developing them. These factors might vary for a feature that is to be developed either system-specifically or for re-use. It is questionable under which conditions people can make estimates for the DR and DI-Drivers and whether they yield a model with acceptable accuracy.

3.3.4 Cobra [1]

Although being a method for cost estimation, COBRA and derived approach for product line quality models [13] provide similarities with the approach discussed above. The basic idea is to develop a causal model with factors impacting reliability (similar to the network in BBNs but smaller) and assess the impact of each factor quantitatively. This assessment is done in two steps. First, an ordinal level is selected for each factor and then this level is quantified by means of multipliers. This approach is similar to the one in QUALCOMO, in which also first the level of each COCOMO factor is assessed on an ordinal scale and where experts provided multipliers using the Delphi-Method [3], a technique for expert knowledge elicitation [9]

However, the drawbacks of this approach (also valid for QUALCOMO):

- Is it feasible to estimate the multipliers? (The answer to this question also might depend on the specific measure of reliability selected). Within cost estimation it is quite natural to assume a nominal (i.e., perfect) project and consider how various risk factors might enlarge the development time. Actually, this was one observation that lead to the definition of Cobra. For quality estimation it is still to be adressed whether the multipliers can be estimated. A simple test would be here to take some sample questions and let some interviewees assign multipliers to these questions and debrief them how difficult the estimation was.
- Are there impacting factors that differ on the feature level? Are there factors that differ based on the development approach (system-specific, development for reuse)? This should be explored first by selecting from the factors in Table 1.

3.3.5 Open issues

The above approaches can be seen as attempts to estimate reliability for a single system that is mainly developed from scratch. In the context of product lines, this still does not answer:

- How can the reliability be predicted if the feature is developed via adaption from existing code?
- How can the reliability be predicted if the feature is developed via reengineering from legacy code?

These two additional starting situations need also to be taken into account. Here two basic questions are: How reliable is the existing code? and What is the impact of the adaptation/reengineering on reliability?

4 Proposal for future activities

Since none of the approaches discussed above seems readily to be implementable, data collection activities should take place. The collected data can then be used to i) investigate which factors impact reliability, ii) it will provide understanding of the phenomenon reliability so that estimations to be performed within QUALCOMO or Cobra can be performed more likely, iii) the data could be used to determine additional model parameters such has the ones in the QUALCOMO, and iv) the data can be used for a simple prediction model.

The idea behind using the collected data for prediction purposes is, for example, to create a boxplot of several modules' reliability (as dependent variable) and factors such as the development mode (from scratch, reused, adapted) and take the mean value for each development mode as prediction for the corresponding mode of development.

A GQM goal could look like: Analyse the *features of a product line* with respect to their *reliability* for the purpose of *understanding/prediction* from the viewpoint of a product-line planner in the enviromnent x.

The quality focus can be defined as any of the measures introduced in Section 2. Since the reliability for commercial systems is usually measured in terms of fault density, it might also be a reasonable measure for product lines. Moreover, it simplifies data collection as measures like software reliability and MTTF require detailed time and enviroment charracterisations.

For example it might be for each feature measured:
– the fault density in field
– the fault density during system test.

For these measures, a resonable size measure has to be selected, a period for defect data collection determined (e.g., the first 12 month after release), and it has to be determined how defects/faults/failiures are to be counted.

Additionally, it can be a subgoal established that investigates the relationship between the number of failures or problems in a feature and the number of instantiations.

As important variation factor the development starting situation (from scratch, adapt, reengineer) should be considered.

Additional factors that have been reported in the work sketched in Section 3 to influence reliability are listed in the table below. This table is simply a collection of the factors listed in the referenced work and in [12][14]. Thus, the entries might sometimes refer to the same concept. In order to find factors relevant for the problem at hand it might be reasonable to select those items that are likely to differ for different features. The categories Platform, Product, Personnel seem to be most relevant here.

Table 1:
Factors on reliability

Category	Factor	Description	Source
Platform	Required Software Reliabililty (RELY)	COCOMO II Factor	[3]
	Database Size (DATA)	COCOMO II Factor	[3]
	Required Reusability (RUSE)	COCOMO II Factor	[3]
	Documentation Match to Life-cycle needs (DOCU)	COCOMO II Factor	[3]
	Product Complexity	COCOMO II Factor	[3]
	Program Complexity	(e.g., McCabe, size)	[14]
	Percentage of reused code		[14]
Product	Execution Time Contraint (TIME)	COCOMO II Factor	[14]
	Mainstorage Contraint (STOR)	COCOMO II Factor	[3]
	Platform Volatility (PVOL)	COCOMO II Factor	[3]
	Platform Difficulty (PDIF)	COCOMO II Factor	[3]
	Program category	such as operating system, dbms system, language processor)	[14]
	Difficulty of Programming	measured as effort/(duration*duration)	[14]
Personnel	Analyst Capability (ACAP)	COCOMO II Factor	[3]
	Programmer Capability (PCAP)	COCOMO II Factor	[3]
	Application Experience (AEXP)	COCOMO II Factor	[3]
	Platform Experience (PEXP)	COCOMO II Factor	[3]
	Language and Tool Experience (LTEX)	COCOMO II Factor	[3]
	Personnel Continuity (PCON)	COCOMO II Factor	[3]
	Personnel Capability (PERS)	COCOMO II Factor	[3]
	Personnel Experience (PREX)	COCOMO II Factor	[3]
	Programmer Skills	(e.g., average number of years of experi-ence)	[14]
	Programmer Organisation	defined as the proportion of high-quality programmers	[14]
	Domain Knowledge	programmers knowledge of input space and output target	{14]
	Human Nature	e.g., ability to aviod making mistakes, careless work omission	[14]
Project	Use of Software Tool (TOOL)	COCOMO II Factor	[3]
	Multisite Development (SITE)	COCOMO II Factor	[3]
	Required Development Schedule (SCED)	COCOMO II Factor	[3]
	Disciplined Methods (DISC)	in addition to COCOMO II	[3]

Category	Factor	Description	Source
	Facilities (FCIL)	COCOMO II Factor	[3]
	Amount of programming effort	in man years	[14]
	Testing effort	e.g., number of test cases, man-years	[14]
	Programming Language		[14]
	Frequency of specification change		[14]
	Volume of program design documents		[14]
	Work Standard	(indicates artefacts to be produced in the process)	[14]
	Development mamagement		[14]
	Program work contents (stress)	e.g., schedule pressure, too much work, mental stress	{14]
	test environment	degree of compatibility between test enrironment and operational environment	[14]
	Testing tools		[14]
Scale Factors	Precedentness (PREC)	COCOMO II Factor	[3]
	Development Flexibility (FLEX)	COCOMO II Factor	[3]
	Architecture/Risk Resolution (RESL)	COCOMO II Factor	[3]
	Team Cohesion (TEAM)	COCOMO II Factor	[3]
	Process Maturity (PMAT)	COCOMO II Factor	[3]
	Development team size		[14]
Process	Level of programming technologies	?	[14]
	Design methodologies	e.g., structured design, functional design	[14]
	Requirements Analysis		[14]
	Requirements Management	CMM KPA	[7]
	Software Project Planning	CMM KPA	[7]
	Configuration Management	CMM KPA	[7]
	Defect Prevention	CMM KPA	[7]
	Inspections	CMM KPA	[7]
	Software Quality Assurance	CMM KPA	[7]
	Design Inspections	(i.e., relationship of detailed design to requirements)	[14]
	Testing methodologies		[14]
	Test coverage	percentage of code covered by tests	[14]

5 References

[1] L. C. Briand, K. E. Emam, and F. Bomarius. COBRA – a hybrid method for software cost estimation, benchmarking, and risk assessment. In *Proceedings of the 20th International Conference on Software Engineering*, pages 390–399, 1998.

[2] S. Chulani. COQUALMO (Contructive QUALity MOdel) a software defect density prediction model. In *Proceedings of ESCOM - SCOPE 99*, pages 297–306, 1999.

[3] S. Devnani-Chulani. Results of delphi for the defect introduction model (sub-model of the cost/quality model extension to cocomo ii). Technical Report USC-CSE-97-505, University of Southern California-Center for Software Engineering, 1997.

[4] N. E. Fenton. A critique of software defect prediction models. *IEEE Transactions on Software Engineering*, 25(5):675–689, sep 1999.

[5] S. H. Kan. *Metrics and Models in Software Quality Engineering*. Addison Wesley Publishing Company, 1995.

[6] E.-A. Karlsson, editor. *Software Reuse. A Holistic Approach*. John Wiley & Sons, 1995.

[7] M. Krishnan and M. I. Kellner. Measuring process consistency: Implications for reducing software defects. *IEEE Transactions on Software Engineering*, 25(6):800–815, nov 1999.

[8] M. R. Lyu. *Handbook of Software Reliability Engineering*. McGraw-Hill, 1995.

[9] M. A. Meyer and J. M. Booker. *Elicitating and Analyzing Expert Judgement: A Practical Guide*. Number 5 in Knowledge-Based Systems. Academic Press, Ltd., 1991.

[10] J. D. Musa. *Software Reliability Engineering*. McGraw Hill, 1998.

[11] J. D. Musa, A. Iannino, and K. Okumoto. *Software Reliability: Measurement, Prediction, Application*. McGraw-Hill, 1987.

[12] H. Pham. *Software Reliability*. Springer-Verlag, 2000.

[13] K. Schmid. A framework for product line quality model development. Technical Report IESE Report No. 47.00/E, Fraunhofer Institute for Experimental Software Engineering, 2000.

[14] X. Zhang and H. Pham. An analysis of factors affecting software reliability. *The Journal of Systems and Software*, pages 43–56, 2000.

Document Information

Title:	Reliability Prediction Models for Product Lines - A Brief Overview
Date:	July 16, 2000
Report:	IESE-046.00/E
Status:	Final
Distribution:	Public