Survey to Software Reliability

Martin Jedlicka, Oliver Moravcik, Peter Schreiber
Faculty of Materials Science and Technology
Institute of Applied Informatics and Automation
Slovak University of Technology in Bratislava
Pavlinska 16, 91724 Trnava, Slovakia
martin.jedlicka@qintec.sk, oliver.moravcik@stuba.sk, peter.schreiber@stuba.sk

Abstract. *This paper introduces key facts from reliability and software reliability theory. Next we discuss the field of software reliability evaluation. Finally we describe some most popular reliability models.*

Keywords. reliability, software reliability, software reliability evaluation, metrics, software reliability models.

1 Introduction

Today's industry systems more and more depend on software which is sometimes very complex. Software complexity increases also with the risk factor of the environment where the whole system is deployed. From these reasons the requirements for software reliability cannot be missed out when designing such system.
Software reliability differs from hardware reliability because it reflects the design perfection, rather than manufacturing perfection. Similar to hardware the reliability of software should be evaluated and measured, even it is not so simple task as it is in hardware because software is hard to touch.

Many organizations and individuals developed methods for software reliability evaluation and this paper should introduce some of them.

2 Reliability theory

For engineering purposes, the reliability is defined as "The probability that a device will perform its intended function during a specified period of time under stated conditions". [1]

This may be expressed in mathematical way as

$$R(t) = \int_t^{\infty} f(x)dx \qquad (1)$$

where $f(x)$ is the function representing the failure probability density and t represents the length of time (which is assumed to start from time zero).

If we want better understand the reliability field, we have to focus on four aspects coming out of definition above: [1]

1. Reliability is a probability. This means that failure occurs randomly, it can be a individual or recurring event. The incidence for failures varies in time according to the chosen probability function. Reliability engineering is then concerned with meeting the specified probability of success, at a specified statistical confidence level.
2. Reliability is predicated on "intended function". This means that this is generally understood as the mean operation without any failure.
3. Reliability applies to a specified period of time or other unit. This means that reliability of the system is guaranteed e.g. for a specified time, kilometres, cycles, etc.
4. Reliability is restricted to operation under stated conditions. This constraint is clear because it is not possible to design a system for unlimited conditions. The operating environment must be taken in focus when designing and testing the system.

3 Software Reliability

Software reliability is a key part in software quality. There are many different views for software quality but almost in every one the reliability plays important role as the stand-alone quality characteristic or attribute. The standard ISO/IEC 9126:1991 defines reliability as “The capability of the software product to maintain a specified level of performance when used under specified conditions”. [2] In this model the reliability is the one of the six main quality characteristics and contains 4 subcharacteristics: [2]

- **Maturity** - The capability of the software product to avoid failure as a result of faults in the software.
- **Fault tolerance** - The capability of the software product to maintain a specified level of performance in cases of software faults or of infringement of its specified interface.
- **Recoverability** - The capability of the software product to re-establish a specified level of performance and recover the data directly affected in the case of a failure.
- **Reliability compliance** - The capability of the software product to adhere to standards, conventions or regulations relating to reliability.

IEEE Std 982.2-1988 [3] defines reliability as “The probability that software will not cause the failure of a system for a specified time under specified conditions”. IEEE 982.1-1988 also defines Software Reliability Management as "The process of optimizing the reliability of software through a program that emphasizes software error prevention, fault detection and removal, and the use of measurements to maximize reliability in light of project constraints such as resources, schedule and performance." Under these definitions, software reliability focuses on three aspects:

- **Error prevention**
- **Fault detection and removal**
- **Measurements to maximize reliability** (specifically measurements supporting the first two aspects)

3 Software Reliability Evaluation

It is known that software evaluation and measuring is problem which is not specific just for reliability field. There is no definition of software nature even the definition which aspects of software impacts on software reliability.

The current practices of software reliability measurement can be divided into four categories: [4]

1. Product metrics
2. Project management metrics
3. Process metrics
4. Fault and failure metrics

3.1 Product Metrics

Software size is taken as the basic indicator of complexity, development effort and reliability. For measuring software size these metrics are typically used: Lines Of Code (LOC), or LOC in thousands (KLOC), more often Source Lines Of Code (SLOC) where the comments and other non-executable statements are not counted. But the problem is that there is not a standard and general way how to compare e.g. two software products not written in the same programming language.

Another product metric is Function point method for measuring the functionality of a proposed software development. This method can be used for estimation of software size as soon as the function points (outputs, inquiries, inputs, files, and interfaces) are identified. The main advantage is that it is independent on programming language. The small disadvantage is that it is hardly used for real-time applications.

Reliability is closely related to software reliability. The most representative metric for measuring software complexity is McCabe's Complexity Metric. It directly measures the number of linearly independent paths through a program's source code by representing the code as control flow graph consisting of nodes (corresponds to programme commands) and edges (connection between two nodes if the second command might be executed immediately after the first command) [5]

Last product metrics are test coverage metrics which evaluate the software reliability by performing tests on software product. The function representing the expecting reliability be expressed in mathematical way as [6]

$$R = 1 - \frac{f}{n} \tag{2}$$

where f represents the number of failed tests and n represents the overall number of executed tests. The desired value of R is close to 1.

This paper [7] presents following software reliability metrics:

- number of test cases/source lines of code (R1) – indicates the density of coverage the source code by test cases
- number of test cases/number of requirements (R2) - indicates if the all requirements were tested appropriately
- test lines of code/source lines of code (R3) - indicates if the ratio R1 was inaccurate as there might have been a few test cases that might have been comprehensive
- number of asserts/source lines of code (R4) - serves as a control for both R1 and R2 so that in case there are few test cases but each have a large number of successful calls to the source program then the metric suite does not penalize the developer
- code coverage (C) - measures the ratio of the number of lines of the source code that are executed by the test code so that most of the source program is covered by testing and indicates the accuracy of R3.

3.2 Project management metrics

Long time experience shows that good project management during software development can result in good and quality products. This reliability depends mainly on use of appropriate processes e.g. development process, risk management process, configuration management process, documentation process etc.

3.3 Process Metrics

This approach assumes that the quality of the software product is a direct function of the used process or processes. Then the process metrics can be used to estimate the reliability of software. Examples of process metrics affecting software: [8]

- Number of times the program failed to rebuild overnight
- Number of defects introduced per developer hour
- Number of changes to requirements
- Hours of programmer time available and spent per week
- Number of patch releases required after first product ship

3.4 Fault and Failure Metrics

These metrics try to determine the moment when the software enters the failure-free period of its operation. They concentrate on collecting and analyzing faults and failures found after testing during development process but mainly on collecting faults and failures during the operation phase when the software is used by real users.

Typical metric is Mean Time Between Failures (MTBF) which estimates the time between two successive failures of a system. It is a key reliability metric for systems that can be repaired or restored.

Test strategy is highly sensitive on the software functionality coverage by tests. If the tests don't cover the whole software functionality and all are passed, there is still likelihood that failure will appear during the time

4 Software Reliability Models

Reliability Models were first developed and applied in hardware systems but they are often adopted also for software. It has been developed many models during the time but these can be divide into two main groups: [4]

1. **Prediction models** – they predict reliability at some future time by using historical data from past projects
2. **Estimation models** – they estimate reliability at either present or some future time by using data from the current software development effort (mainly used in test phase)

Estimation models, which are more appropriate for software, can be divided into four categories: [9]

1. **Error Seeding** - estimates the number of errors in software from the errors which are seeded into software. All errors are divided into indigenous (real) and induced (seeded) errors. The unknown number of indigenous errors is estimated from the number of induced errors and the ratio of errors obtained from debugging data.
2. **Failure Rate** - is used to study the software failure rate per fault at the failure intervals. As the number of remaining faults changes, the failure rate of the software changes accordingly.
3. **Curve Fitting** - uses statistical regression analysis to study the relationship between software complexity and the number of faults in a program, as well as the number of changes, or failure rate.
4. **Reliability Growth** - measures and predicts the improvement of reliability programs through the testing process. Reliability growth also represents the reliability or failure rate of a system as a function of time or the number of test cases.

Most of the models use the Poisson process

$$P(t) = \frac{(\lambda(t))^n}{n!} \cdot e^{-\lambda(t)} \qquad (3)$$

where $P(t)$ is failure probability, $\lambda(t)$ is the intensity function or failure rate and n is number of failures.

Then the probability of no failure (the reliability), when $n = 0$, is

$$R(t) = e^{-\lambda(t)} \qquad (4)$$

4.1 Musa's Basic Model

The Musa's basic model assumes that the execution time between failures is piecewise exponentially distributed. Also all failures occur with equal likelihood, are independent of each other and are actually observed. This model is effective, simple and widely applicable.

Musa's basic model is expressed in mathematical way as [10]

$$\lambda(\mu) = \lambda_o \left(1 - \frac{\mu}{\nu_o} \right) \qquad (5)$$

It means that failure intensity λ is a function of failure intensity at the start of execution λ_o, the expected number of failures at a given point of time μ, and the total number of failures occurring in an infinite time ν_o. The necessary requirement of this model is existence of data gathered from running software (e.g from beta-testing).

4.2 Musa-Okumoto Model

The Musa-Okumoto model is also called logarithmic Poisson execution time model. It assumes that all failures occur with equal likelihood and are independent of each other.

Musa-Okumoto is expressed in mathematical way as [11]

$$\lambda(\mu) = \lambda_o e^{\theta\mu} \qquad (6)$$

where failure intensity λ is a logarithmic function of time in this model and failure intensity decay parameter θ.

4.3 Littlewood-Verall Bayesian Model

The Littlewood-Verall Bayesian model assumes that successive times between failures are independent random variables each having an exponential distribution.

The Bayesian approach to reliability assumes that contribution of each fault to the overall failure intensity is not known and can be modeled as originating from a given random distribution (with unknown parameters) of values.

The distribution for the *i*-th failure has a mean of *1/λ(i)*. The *λ(i)*s form a sequence of independent variables, each having a gamma distribution with the parameters *α* and □*(i)*. [12]

4.4 Non-homogenous Poisson Process Model

This model, also known as NHPP model, is based on non-homogeneous Poisson process. It also assumes that all failures occur with equal likelihood and are independent of each other. But is simplifies the real world by assuming that faults are corrected perfectly with no regression. [10]

The failure intensity (λ) of this model is a function of the number of faults collected (N) and the completion time for each interval: [12]

$$\lambda(t) = Nbe^{-bt}$$

This model requires existing fault data be gathered for extrapolation by the model. In particular, it requires fault counts on each testing interval and the completion time for each period.

Conclusion

As can be easily seen, the field of software reliability is very complex and plays the important role in quality of software products. In this paper, we have provided basic overview of facts about software reliability based on software reliability and software reliability models.

References

[1] **Reliability Engineering**, available at `http://en.wikipedia.org/wiki/Reliability_engineering`, Accessed: 15th June 2008.

[2] ISO/IEC 9126:1991 Information technology – Software product evaluation – Quality characteristics and guidelines for their use.

[3] IEEE Standard 982.2-1987 Guide for the Use of Standard Dictionary of Measures to Produce Reliable Software.

[4] Pan J.: **Software Reliability**, available at `http://www.ece.cmu.edu/~koopman/des_s99/sw_reliability/`, Accessed: 16th June 2008.

[5] **Cyclomatic Complexity**, available at `http://en.wikipedia.org/wiki/Cyclomatic_complexity`, Accessed: 16th June 2008.

[6] Tanuška P., Schreiber P.: **The Software Products Testing Process**, Materials Science and Technology 5/2005, available at `http://www.mtf.stuba.sk/docs//internetovy_casopis/2005/5/tanuska.pdf`, Accessed: 17th June 2008, ISSN 1335-9053.

[7] Nagappan N. **Toward a Software Testing and Reliability Early Warning Metric Suite**, Proceedings of the 26th International Conference on Software Engineering, 2004, pp. 60-62.

[8] **Software metric**, available at `http://en.wikipedia.org/wiki/Software_metric`, Accessed: 15th June 2008.

[9] **Overview of Software Reliability**, available at http://sw-assurance.gsfc.nasa.gov/disciplines/reliability/index.php, Accessed: 15th June 2008.

[10] Davis G.: **SENG 635: Software Reliability and Testing Tutorial, Part #2**, available at http://www.guydavis.ca/seng/seng635/tutorial2.doc, Accessed: 15th June 2008.

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

[12] Wagner S., Fischer H.: **A Software Reliability Model Based on a Geometric Sequence of Failure Rates**, available at www4.informatik.tu-muenchen.de/~wagnerst/publ/ada-europe06.pdf, Accessed: 16th June 2008.