UNITECH 2017 GABROVO

INTERNATIONAL SCIENTIFIC CONFERENCE
17-18 November 2017, GABROVO

MEASUREMENT OF SOFTWARE RELIABILITY

Aleksandar Dimov
University of Sofia "St. Kliment Ohridski"
Faculty of Mathematics and Informatics
Institute of Mathematics and Informatics
Bulgarian Academy of Sciences

Krasimir Baylov
University of Sofia "St. Kliment Ohridski"
Faculty of Mathematics and Informatics

Abstract

Software reliability is important quality factor to be considered about contemporary software intensive systems. However currently there does not exist enough support in terms of formal measurement, models and practices to facilitate software reliability estimation. The paper describes a pattern of a methodology for evaluation of reliability of complex software intensive systems, which is based on existing knowledge on specification and modeling of reliability. Practical significance of this pattern is that it may be applied when it is uncertain if the system is ready to be released to the market.

Keywords: Software Reliability, Software quality factors, Measurement of software reliability.

INTRODUCTION

Quality of software intensive systems has recently become a very important context in the domain of software systems. Quality attributes of software are sometimes also called non-functional requirements (or characteristics/properties) and should be distinguished from functional requirements. Functional requirements define what the system should do, non-functional requirements put some additional constraints on how the system should perform.

Traditionally, quality of software has been given a primary importance with respect to such application domains like embedded, safety critical, real time systems and so on. However, in recent times, terms as testability, maintainability, usability and others gain bigger attention. The list of quality characteristics probably would be never complete and depends highly on the application domain.

For example the notion of maintainability is quite undecided in different domains. The ISO/IEC 25010:2011, standard [1], defines the quality attribute of maintainability as more general than modifiability, while the work described in [2] considers exactly the opposite – "Maintainability is a subset of what we call modifiability". The domain of web services is even more different. Recently there was released a standard by OASIS (Advancing open standards for the information society) for Web Services Quality Factors [3], where neither maintainability, nor modifiability was defined. The notion of interoperability was defined there, instead.

This shows that although important, software quality is still a general and quite abstract notion, which may have different meanings. Currently there doesn't exist a universally accepted measurement framework of software quality, although there exist a lot of examples for software metrics in general [4]. A common approach for formalization of software quality is needed, which will establish unified measures for different quality characteristics of software systems.

Dependability is one significant quality parameter, for large variety of safety-critical and embedded software systems and is defined as the ability of a computing system to deliver services that can justifiably be trusted [5]. It is depicted by a number of attributes – availability, safety, confidentiality, integrity, maintainability and reliability:

- Availability represents readiness of the system to deliver correct service.
- Safety is concerned about absence of catastrophic consequences on the

user(s) and the environment in case of system failure.
- Confidentiality is the absence of unauthorized disclosure of information,
- Integrity means absence of improper system state alterations;
- Maintainability is the ability of the system to undergo repairs and modifications.
- Reliability is the continuity of correct service delivering, i.e. the belief that a software system will behave as per specification over a given period of time.

This paper presents a pattern for description and measurement of dependability attributes – namely reliability. The pattern is based on existing knowledge on specification and modeling of reliability. In next section of the paper we give more information on the notion of reliability; Section 3 presents the pattern for reliability measurement; Section 4 discuss its applicability and finally section 5 concludes the paper.

RELIABILITY MODELING METHODS

Traditionally reliability is measured as a probabilistic value and may be represented by one of the following values:

- Probability of failure
- Failure rate
- Mean time to failure

As a probabilistic value, it needs large amount of statistical data in order to calculate its value. The most common way to obtain such large amounts of data is by software testing [6, 7] and other methods include simulation, users feedback and experts opinion.

Simulation takes into account that reliability does not depend only on the structure of the software but also on the runtime information such as frequency of component reuse, execution time spent interactions between the components, etc. Users' feedback is a technique to get information about software reliability parameters of a system, by gathering data, after it has been shipped to the market and during its real usage. Data about system failures is gathered by bug reports submitted by users to a bug report subsystem and bug reports may be classified according to specific levels of severity. Experts opinion takes into account that for simple enough portions of code, reliability may be verified via code review or formal verification of source code [8]. Experts opinion for evaluation of reliability is suitable when applied on simple portions of code (usually up to few hundreds lines) and results are expected to give a reliability of 100%.

Currently, there exist two very broad categories of methods for estimation of the reliability of software systems which may be generally called white-box and black-box models.

Black-box models are also often referred as reliability-growth models and there exist a large number of them [10]. They are used to reason about reliability of software systems, without taking into account their internal processes or structure. This means that they rely purely on statistical data in order to evaluate reliability of the overall system as a monolithic whole. Normally, black box models are aimed on application over failure datasets, generated by testing of the system. Nevertheless, given that simulation and users feedback methods produce the output that is required by the model, they may also be applied for reliability estimation via black box methods.

On the other side, the group of white-box models [12, 13] consists of several kinds of methods that are used to estimate the reliability of software systems, based on the knowledge of their internal structure and processes going on inside them. This knowledge may be expressed by different means, such as architecture models, test case models, etc. Usually architecture-based software reliability estimation takes the following main steps [12]:

(1) Identification of computational modules (components) within software architecture;
(2) Description of the actual architectural model – this includes how components are interconnected and interact with each other;
(3) Definition of components failure behavior – at this step the reliability

parameters of components and their measures are identified;

(4) Combination of the failure behaviour with the architectural model.

Application of white box models has a lot of advantages, among them are:

- Ability to reuse information about reliability parameters of both the system and the components that constitute it;
- Ability to find these modules that influence systems reliability the most, i.e. – possibility to isolate and remove reliability "bottlenecks" within the system and.

However, one of the most remarkable assumptions of most white-box models is that they suppose reliabilities of individual system constituent parts are known in advance. Even in case of very fine grained architecture and decomposition of the system to small enough components, their reliabilities should be estimated in some way, for example via black box-models.

RELIABILITY ESTIMATION PATTERN

Here we describe the pattern for calculation of reliability of complex software intensive systems, with respect to the information given in the previous section.

Context

A software system is being developed and a decision is needed when it is appropriate to release it to the market.

Problem

As mentioned in the previous section there exist four methods for collection of data for calculation of software reliability (testing simulation, users feedback and experts opinion). Although testing is the most common of these methods, it is not always the most appropriate. For example, in the embedded systems domain, some systems may have ultra-high requirements towards reliability, for instance levels of $1\text{-}10^{-12}$ and more. It has been shown [9] that for achieving such requirements purely by statistical processing of testing data it is needed to test with non-correlated input (e.g. non-correlated test-cases) for hundreds or even thousands of years. This of course is infeasible and represents a major obstacle towards application of reliability into everyday development practice that should be solved.

It is widely known that the cost of testing in software life-cycle may be even higher than development or design in particular. In many cases this is because it is really never known when enough testing has been done. It could be useful to have some formal means when to stop the testing phase and declare readiness to release the software to the market. Thus main practical significance of the notion of reliability is that it could be applied in order to determine the moment when the system is ready for the market.

However, as discussed above, testing is not always applicable in many application domains and sometimes additional methods to gather reliability measurement data should be applied. Next section describes the formal procedure that should be followed to calculate software systems reliability.

Solution

Software reliability of the system should be continuously calculated and when it reaches a required threshold, the system is considered ready to be shipped.

The following activities should be performed (fig. 1) :

S1: Define system architecture – this is a complex activity. If there is a good documentation of the system and enough design has been performed prior to its implementation, then the architecture already exist. In such case the modules of the system are identified. At this step a white box model for software reliability estimation should be also selected. For a complete list of white box models, refer to [13].

(1) C1: For each component in the architecture determine if it is possible to be tested into a real environment. For many application domains testing into a real world environment is not possible, for example for security or safety reasons.

(2) C2: For each component which is not possible to be tested determine if there exist a simulation environment for it.

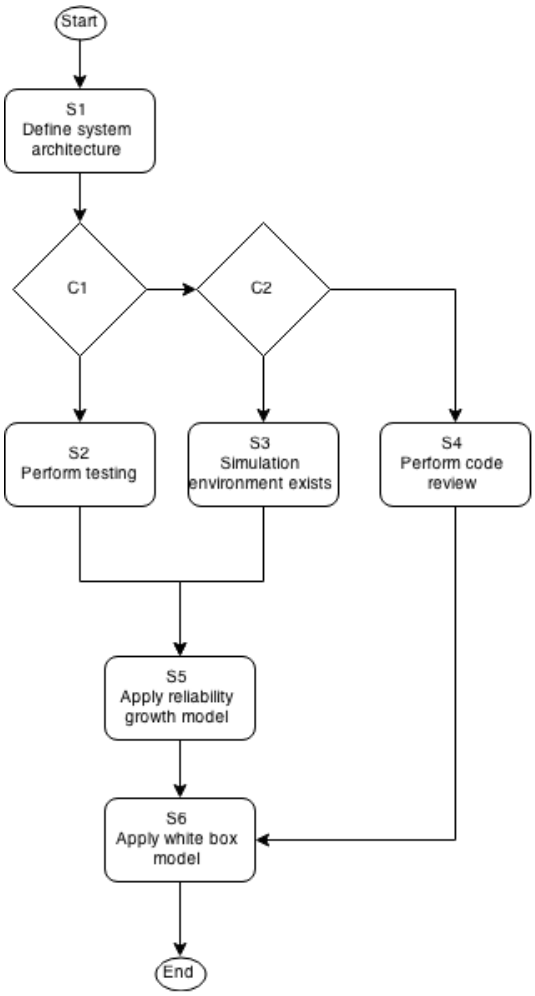

Fig. 1. *Activities for calculation of reliability for complex software intensive systems*

(3) S2, S3 and S4 are the steps where the reliability evaluation data is collected. Currently we do not know available formal models for reliability estimation via experts' opinion. In that case, possible results from S4 are either reliability is 0 or 1. Zero means the component is absolutely unreliable, i.e. it should be significantly refactored and 1 means that the component is 100% reliable. Indeed, in general case, it is possible to estimate this via code reviews for a relatively simple pieces of code (in the range of several hundred lines of code). In S2 there could be set a stop criteria for testing, depending on the target system.

(4) S5: Based on simulation and testing data, reliability growth (black-box, see section 2) model should be applied. For instance an appropriate procedure for selection of reliability growth model is described in [11] and it could be used at this step of the pattern.

(5) S6: The last step is to apply a white box model in order to evaluate the overall reliability of the system.

DISCUSSION

The pattern described in previous section should be applicable in all domains of software engineering to support the verification and validation phase. This way, it is applicable for wide variety of software systems. However, there exist some difficulties and this section makes a brief discussion of them.

Formal introduction of the notion of reliability in development of software intensive systems is mainly hampered by the fact that most reliability estimation models have theoretical impact. This section shows in more detail some of the reasons about that.

Usually, black box models make a number of simplifying assumptions, in order to calculate reliability. Such assumptions include:

(1) Bug-fixing code is always correct - when removing the fault that led to the failure, no new faults are introduced into the code. This is known to be never applicable in practice, as most of the failures are result of complex interaction between many parts of the system. Nevertheless how good the architecture is, developers rarely may predict what will happen when they make changes into the code, in order to fix the failure. The typical, current solution is to introduce additional testing procedures and architectural decisions, to verify that the new code will execute with no failures.

(2) Aging – generally speaking, software code does not wear out and most reliability models assume so. However some abstract notions of software aging should be taken into account. For instance, the execution environment may change while the system is running – new software may be installed or also shut down, memory leaks may occur, or malicious attacks may take place. Other examples include systems that are going to be used for a long time and their core business logic may appear to be written using an outdated programming language and/or technology.

(3) Next issue is that black-box reliability models assume that failures in the software system occur following a

particular a probability distribution. Thus probability of failure may be calculated with well-known mathematical formulas [10]. This way a single model is seldom appropriate for a large range of systems, because each system has its own failure behavior, which may differ from the distribution assumed by the model. In such a case, the model will give biased estimation about reliability. An approach to deal with this problem and select a reliability growth model is shown in [11].

(4) Software system under-estimation – as mentioned earlier in the paper, software reliability is defined as a continuity of a correct service and may be measured in percentage or time between failures. The output of reliability models is either estimation of mean time between failures or probability that the system will behave as per specification over some desirable time interval. However, none of the models take into account that an absolutely correct mathematical estimation of reliability is not really practically needed. Rather, it would be enough that the system is not over-estimated, i.e. it will not crash earlier than predicted.

CONCLUSION

The paper describes a pattern of a methodology for evaluation of reliability of complex software intensive systems. Practical significance of this pattern is that it may be applied when it is uncertain if the system is ready to be released to the market.

Directions for future research include:

- Generalization of the pattern for larger number of quality characteristics.
- Improvement of the pattern in order to cope with problems inherent for software reliability estimation models
- Development of formal model for reliability estimation based on experts' opinion.

ACKNOWLEDGEMENT

Research, presented in this paper was partially supported by the DFNI I02-2/2014 (ДФНИ И02-2/2014) project, funded by the National Science Fund, Ministry of Education and Science in Bulgaria.

REFERENCE

[1] ISO standard on System and software quality models, Cabral, available at https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en

[2] Bachmann, F., L. Bass and R. Nord. Modifiability Tactics. Technical Report CMU/SEI-2007-TR-002, September 2007.

[3] OASIS Web Services Quality Factors Version 1.0, available at http://docs.oasis-open.org/wsqm/WS-Quality-Factors/v1.0/cos01/WS-Quality-Factors-v1.0-cos01.html

[4] Fenton N. Software Metrics — A Rigorous Approach. Chapman & Hall, London, 1991.

[5] Avižienis, A., Laprie, J-C., Randell, B., Basic concepts and Taxonomy of dependable and secure computing, IEEE Trans on Dependable and Secure computing, Vol. 1, Issue 1, Jan - March 2004.

[6] Dimov, A., S. Chandran and S. Punnekkat (2010). How do we collect data for software reliability estimation?. In proc. of of the 11th international conference on Computer systems and technologies (CompSysTech 2010), 155-160, ACM ICPS, vol. 471, Sofia, Bulgaria.

[7] Myers G., et all, The art of software testing, John Wiley & Sons, New York, 2004

[8] Arun Babu, P., C. Senthil Kumar, and N. Murali. "A hybrid approach to quantify software reliability in nuclear safety systems." Annals of Nuclear Energy 50 (2012): 133-140.

[9] Butler, R., and Finelly, G., (1993), The infeasibility of quantifying the reliability of life-critical real-time software. IEEE Transactions on Software Engineering. 1(19), 3-12.

[10] Farr, W., (1996). Software reliability modeling survey. In: M.R. Lyu (Ed.), Handbook of Software Reliability Engineering, 71–117, McGraw-Hill, New York.

[11] Stringfellow, C., Andres, A. Amschler, An empirical method for selecting software reliability growth models, Empirical Sotware Engineering, Vol.7, Issue 4, pp.319-343, 2002.

[12] Goseva-Popstojanova, K., Hamill, M., and Xuan Wang, (2006). Adequacy, Accuracy, Scalability, and Uncertainty of Architecture-based Software Reliability: Lessons Learned from Large Empirical Case Studies, In Proceedings of the 17th International Symposium on Software Reliability Engineering, 197-203.

[13] Chandran, S., A. Dimov and S. Punnekkat. (2010). Modeling Uncertainties in the Estimation of Software Reliability – a Pragmatic Approach. Proceedings of the 4th IEEE International Conference on Secure Software Integration and Reliability Improvement (SSIRI). Singapore. June 9-11, 2010. 227-236.