Available online at www.sciencedirect.com

ScienceDirect

IFAC Papers Online CONFERENCE PAPER ARCHIVE

IFAC-PapersOnLine 49-30 (2016) 001–005

Quantification of the quality characteristics for the calculation of software reliability

N. Jazdi, Senior Member IEEE, N. Oppenlaender, M. Weyrich

University of Stuttgart, Institute of Industrial Automation and Software Engineering
Pfaffenwaldring 47, 70569 Stuttgart, Germany
e-mail: {nasser.jazdi; niels.oppenlaender; michael.weyrich}@ias.uni-stuttgart.de

Abstract: This paper presents a method to describe and increase the software quality of automation systems by a quantitative determination of the software reliability. Therefore, the software development process has been analyzed to identify the essential factors influencing the software reliability. These are the usage of a procedure model, the test factor and the human factor of the software developers themselves. These factors have been considered to create a neuro-fuzzy-based concept, which characterizes and consolidates them to realize an estimation of the software quality. The possibility to estimate the software quality, especially dynamically already during the software development process, will entail a huge benefit to programmers and manufacturers. Effects, methods or circumstances influencing the software quality in a negative or positive way can be numerically described and rated. On basis of that, necessary optimizations in the development process can be performed in time to guarantee a high quality of the software.

Keywords: Quality characteristics; Quantification, IoT, Reliability, Human factor

1. INTRODUCTION

Every new development in technology leads to new challenges. To be successful on global market, high-quality products and processes are fundamental (Bauer 2011), (Hillmann 2014). In conclusion, the qualitative properties in particular determine the success of modern automation systems. The characteristics of reliability or availability are therefore driving innovations for future applications. For automation technology, especially for smart systems in the context of "Industry 4.0" or the internet of things, it is very important to determine the quality at an early stage. If feasible, already during the software development process to prevent high costs of in field error fixing. Adapted methods for reliability analysis are necessary to provide this possibility. The reason of that is, that future automation systems are highly flexible e.g. in adapting to changing conditions as well as because these systems are increasingly realized of intelligent components (Jazdi 2014). Two key aspects of this are the autonomous character and the interaction with humans. These interactions are not only taking place during the start-up, the operation, the maintenance or the shut-down of a system. The human influence starts already during the development. Software development follows requirements which are defined by humans and furthermore software programmers implement the code. Therefore, the human factor has to be considered in the reliability analysis as well as the procedures within the development process itself to achieve a high qualitative software.

2. STATE OF THE ART

Quality in general is defined as the degree to which a set of inherent characteristics of an object fulfils requirements (ISO 2015). In the context of globalized competition and smart systems, more and more complex systems on basis of high-demanding requirements have to be developed.

These systems contain mechanical, electrical and information processing components which gain intelligence and autonomous characteristics by the integration of, for example, smart sensors and actors (Jazdi 2014). Especially the software has to be designed for the special needs of "Industry 4.0" and the internet of things (Weyrich 2016). These needs differ from those of conventional systems in general purpose (Lee 2008). The aspect of inherence in the quality definition describes the fact that these characteristics are always present and can be measured. This leads to the need that the essential quality characteristics have to be quantified. Without measurement and quantification no statement about quality is possible. For example a high-quality software shows fewer errors and has a high reliability. On basis of that, the software reliability can be considered as a metric for the software quality (Jazdi 2012). The possibility to quantitatively estimate the software reliability is therefore an important aspect to develop high qualitative systems.

The reliability of a system is defined as the ability of a system or a component to function under stated conditions for a specific period of time (IEEE 1990). A smart system in the context of "Industry 4.0" is an automated systems that enables connection of the operations of the physical reality, in hardware, with computing and communication infrastructures, in software (Lee 2008), (Bahati, 2011), (GNACE 2011). Therefore a separated view for the hardware and the software is necessary.

Hardware reliability is determined by the consideration of influencing factors like temperature, overvoltage, overcurrent

10.1016/j.ifacol.2016.11.113

or erosion. On basis of empirical data and the associated available knowledge about the error rate of mechanical and electrical components, the hardware reliability can be calculated with the usage of a probability model (Jazdi 2012). In contrast to hardware, the software reliability is exposed to many variables which requires another methods to calculate. Due to the fact, that software errors has no physical causes and no empirical data is available, especially in early development stages, there are no probability models which can provide a reliable estimation of the software reliability. Furthermore, an error which leads to a failure of the software is depending on various additional conditions in the execution context. Therefore, the software reliability depends on many uncertainties. It is only possible to collect data during test phase of the software and calculate the reliability by the following Formula (Jazdi 2016):

$$R_{Software} = \frac{corrected\ executions}{total\ excetions} \quad (1)$$

For the above given method, it has to be observed if an inherent error has been activated during the test execution (Jazdi 2012). By this definition, the hypothesis can be derived, that the software reliability is estimated by the consideration of the error frequency and the related number of executed test cases. Error frequency and the related number of test cases lead to the assumption, that the development of the software itself has the highest influence on the software reliability. Software errors are either made by the developer during the design phase (specification errors) or during the implementation (programming errors).

The method, which will be presented in the following chapter, uses fuzzy logic and neural networks to estimate the software reliability. With the help of fuzzy logic uncertainties, so called fuzzy factors, can be described as linguistic variables and linguistic hedges (IEC 1997). There are also patterns and relationships between these factors which have to be recognized and considered. The recognition can be ensured by the help of an artificial neural network, which is used as a state-of-the art in scenarios of pattern recognition or classification techniques for data mining (Gosch 2014). With the help of a neural network, the dependencies of the factors can be identified by a learning process.

The use of neuro-fuzzy-based concepts in the context of software reliability is already used in a way to predict the software reliability based on growth models by considering software failure data, which can be applied during the test phase (Roy 2015), or by considering given software failure data sets (Lakshmanan 2015). But in these corresponding works, the software reliability estimation is only based on the information about failed or correct executions, like given in the equation above. Due to the fact, that software errors depend on uncertainties, these uncertainties has to be considered necessarily as influencing factors. The influencing factors of the development procedure can be identified as the usage of a procedure model during software development, the test factor of the software examination and the human factor. By quantifying these influencing factors, an estimation of the software reliability is possible.

Reusability is furthermore another important aspect of software development. By using reusable components, empirical data and information can be available which has to be included in the method to estimate the reliability as accurate as possible (Jazdi 2012).

3. APPROACH OF QUANTIFICATION OF THE QUALITY CHARACTERISTICS

The first step on this approach was to identify the factors influencing the software quality. On the next step a methodology was defined to quantify these factors. In the following the most important factors are introduced.

1.1 Usage of a procedure model

There are several procedure models which are used to control the complexity of the software and to apply methods during the software development. The usage of a procedure model enables the developer to implement high-quality software and to decrease the costs and time (Jazdi 2012). Every procedure model, e.g. the waterfall model or an agile process like SCRUM, divides the development in defined phases. These phases are different in time and level of detail, depending on the kind of the procedure model. In every phase a software part is developed and evaluated. Only if the evaluation of the software part at the end of a phase is successful and the part is released, the next phase will start. Therefore, it is necessary, to use a general reference model which can be adapted to quantify every kind of procedure model. The CMMI (Capability Maturity Model Integration) for development can be used as an example to show this is possibility (SEI 2010). With the help of five maturity levels the process can be quantified by referring the corresponding values to the stages. The lowest maturity level represents the absence of a procedure model with the value 1. The next stage, level 2, is related to a managed procedure model containing for example project or configuration management aspects. If these processes can be adapted to the current project, e.g. by validation or verification techniques or the further education of the developers, the procedure model can be rated with level 3. If statistical control metrics are used, for example for the project management, the 4th level is reached. The level and value 5 is referred to the highest maturity level, which represents an optimized procedure. This means, on basis of a statistical control, the procedure is improved by analysis and eliminating the causes of errors.

1.2 Test factor

Test is defined as a determination according to requirements (ISO 2015). With the help of testing many errors within a software can be identified and corrected. In conclusion, the estimation of the reliability is based on data of tests which provides information about error or a failure. The reduction of errors in software leads therefore to a higher reliability and quality of that software. Due to the increasing complexity of software, complete testing is not possible.

The test cases which have the highest possibility to identify the most errors have to be executed. The best possible testing of the software can be guaranteed by that. A prioritization and selection of test cases is needed. But even in case of an optimal and almost fully-tested software, no statements about an error or a failure of the system is possible (Jazdi 2012). Even if a huge amount of errors have been corrected. This depends on the already described circumstance, that a software failure is linked to the execution context. There is no metric available considering these facts, so the test factor has to be divided in the parts of the complexity of the tested software and the test execution itself to enable a quantification.

The more complex a software is, the more likely it is that the software has errors. A software which is too complex to understand cannot be tested sufficiently. In case of that, the metric of the cyclomatic complexity is used for a long time to evaluate the complexity of a software (McCabe 1976). It is suggested by experience, to split a software into parts whenever the cyclomatic complexity is equal to 10 or higher (Watson 1996). Consequently, a cyclomatic complexity under 10 can be considered as a good manageable, and therefore testable, software. To quantify the complexity of a software, the cyclomatic complexity will be calculated and linked to the already introduced rating system of the previous section. A cyclomatic complexity less than 10 is equal to a good testable software, which corresponds to the value 5. A cyclomatic complexity of more than 25 can be equal to a level of 1, which represents a very complex software which is barely testable. The levels in between decrease in steps of 5 cyclomatic complexity values until level 5 is reached.

The other part of the testing factor is the test execution itself containing the test specification techniques and the type of testing, e.g. a black-box or white-box test. The quantification is performed by the same rating system with a range of levels / values from 1 to 5. A random test execution is rated with the level 1. For example a C1-coverage can be rated with the level 5. The levels 2-4 in between can be related for example to a C0-coverage (level 2), the use of a test specification (level 3) or the traceability between requirements and test cases (level 4). The detailed characteristics of a level have to be investigated and defined in further work.

1.3 Human factor

Software errors are made by humans during specification or implementation. The human factor is therefore an important influencing factor for the software reliability. To estimate the human factor, an analysis of the abilities of a software developer has to be performed (Jazdi 2012). Development experience can reduce the number of errors. In fact there are many other individual factors which have to be considered, to describe the productivity and efficiency of a developer. For example should the developer be able to work under consideration of a procedure model. On the basis of the other quality characteristics, the developer experience can also be classified by the levels 1-5 which represents the development factor (dev-factor).

A dev-factor of 1 represents a low development experience which is linked to higher number of errors. The highest productivity and the lowest number of errors is assigned to level 5. There are several possible aspects which have to be considered to determine the dev-factor, for example the development experience containing the knowledge and the further education of the developer. Other influencing factors can be the working conditions like the available resources and the teamwork, the complexity of the software to develop consisting of the difficulty of the problem and the lines of code.

4. APPROACH OF QUANTIFICATION OF THE QUALITY CHARACTERISTICS

In this section the determination of the software reliability on basis of the quantified quality characteristics is presented by introducing an iterative concept in figure 1.

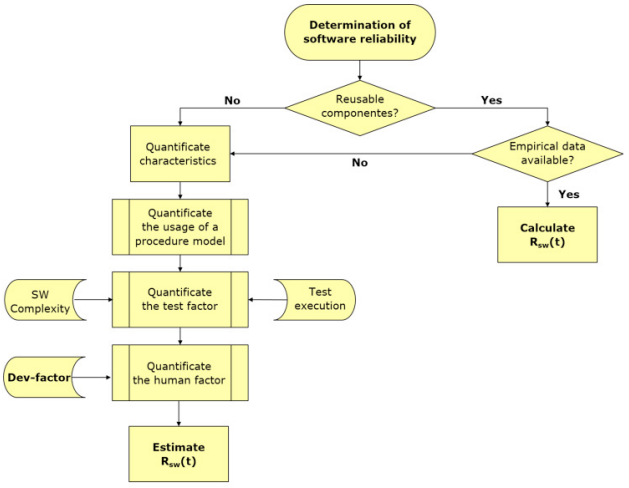

Fig. 1. Flow graph for the determination of the software reliability.

In the first step of the method, the use of reusable components is checked. If reusable components are used, empirical data is available and the software reliability can be calculated by this information with the help of reliability models. If there are no reusable components, the quality characteristics of the project under consideration has to be quantified in the introduced form. These are the quantification of the usage of a procedure model, the test factor consisting of the complexity and execution as well as the dev-factor as quantified description of the human factor. To describe the human factor as a value, a neuro-fuzzy concept is used which is shown in figure 2.

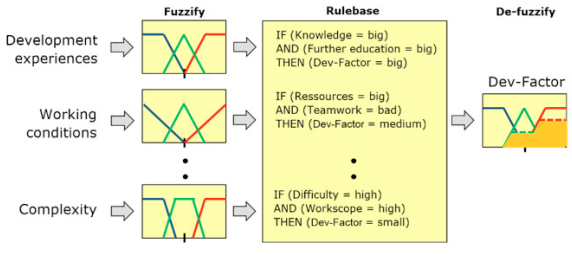

Fig. 2. Fuzzy system for imprecise statements according to the human factor.

As an example of the fuzzy determination of the human factor, the development experiences of the developer, the working conditions within the project environment and the, by the developers perceived, complexity of the software are given. These factors are directly influencing the human factor and contain further additional factors which have to be considered, too. For example the development experience can be divided into the terms of knowledge and further education of the developers themselves. There are also patterns and relationships between the influencing factors, which also have to be taken in account. The recognition of them can be ensured by the neural network, so that dependencies of the factors can be identified by the learning process. Available information about the input and output of a system and structural relationship is provided as input of the neural net. The factors like knowledge or education can be linked to the characteristic factors, e.g. development experience, which is the input of the above introduced fuzzy logic approach.

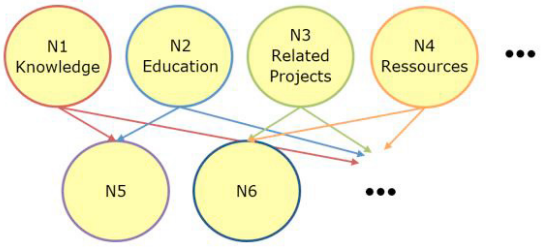

Fig. 3. Example of a neuronal net to recognize patterns and relationships.

The adaption of the artificial neural network and the fuzzy logic enables the calculation of the dev-factor, which quantifies the human factor. The software reliability can be estimated now as described in figure 1 by using the quantified quality characteristics.

5. EVALUATION OF THE APPROACH

In the context of an industrial research cooperation, the possibilities to evaluate the software quality as well as the prioritization of quality characteristics of a vehicle transmission system were examined. In particular individual quality characteristics of the software system functionality were evaluated. The quality evaluation was based on the ISO 9126 standard, but some quality characteristics were not considered. On focus were the functionality, reliability and the changeability of software. With the help of the "Goal-Question-Metric" method, objectives were established to evaluate the quality. Therefore questions were asked to obtain the necessary metrics and data. After the identification of relevant metrics, different rules for determining the relevant attributes of individual factors as well as to evaluate the quality characteristics were established. Based on a fuzzy logic control system, a rule-based quality assessment was prepared. With fuzzy systems, it is not only possible to use linguistically formulated information directly, it is also possible to establish a simple and comprehensible set of rules. A good supporting tool for modelling such systems is MATLAB with the fuzzy logic toolbox. After defuzzification, the values of various quality characteristics were determined. To generate the derived values of the different quality characteristics, different weights were assigned to them. For weighting the individual quality characteristics, the Analytic Hierarchy Process (AHP) method is used. The AHP is a method of decision theory which enables the simplification of a multi-criteria decision systematically and thereby making the decision rational and transparent. A fine-quality evaluation was conducted by this. A statement about the quality of individual test subjects and of the entire software system functionality can be made. For the prioritization of the individual quality characteristics, another set of rules was established on basis of identified factors. Using the fuzzy logic and the AHP, the influencing factors could be weighted differently during the test. The set of rules of the prioritization was made dynamically adaptable by this.

In another industrial work, the opportunities for quality assessment and the prioritization of quality characteristics based on a real project were examined. The different raw data were determined and collected from an 8HP automatic transmission project by considering various development departments. For that a prototype system has been developed which is able to evaluate the quality characteristics of the project and prioritize them. The evaluation results were compared with statements of software developers. The results of the prototype evaluation reflect the majority of the statements made by the software developers. Especially the results of the quality are rated as good. After the interview with software developers, the advantages and disadvantages of the prototype can be summarized as follows.

The following advantages were noted:

- Consideration of influencing factors: In the prototype many factors influencing the quality and priority were considered. Without tool support, e.g. the errors which are discovered by the customers sometimes cannot be considered by the software developers.
- Separation of quality and priority: Quality is a factor influencing the priority. For priority determination other metrics have to be considered, too, such as error rates and security relevance.
- Independent weighting for the various test subjects: In the prototype the weighting of the factors of quality assessment and priority determination can be set. The weighting affects only the result of an individual test subject.

The following disadvantages were noted:

- Elaborate weighting by number of parameters: In this project, over 300 influencing factors of the quality were evaluated. To get a better result, the developer has to set the weight of each factor for each test subject individually, depending on the test requirements. This is very costly.
- Overview of metrics by the developers: During the review process of the prototype, many software metrics were created which are shown as results, partially in the result or in the console. It is difficult for developers to get an overview of the metrics.

- Across projects data of the modules: The basic data of the test subjects are recognized for specific projects from Clear-Quest. The basic data of the modules are recognized across projects from CMT ++. Therefore, the allocation does not fit completely.

- Feedback of test depth: In the prototype there is only an indirect feedback of the quality of the conducted test cases (test coverage of the software). But the actual test depth is not taken into account.

6. CONCLUSION AND OUTLOOK

This paper presented a method to calculate the reliability of industrial automation systems. Since the software reliability is the crucial aspect that can influence the entire system reliability, it deserves special attention. For this reason, this paper discussed the influencing factors of the software reliability as well, for example the software development process was analysed to identify the most important influencing factors. The mentioned factors have been considered to create a neuro-fuzzy-based concept, which characterizes and consolidates them to realize an estimation of the software quality. In summary, the main messages are:

- Possibility to estimate the software quality, especially dynamically already during the software development process, which will entail a huge benefit to programmers and manufacturers.

- Effects, methods or circumstances influencing the software quality in a negative or positive way can be numerically described and rated.

- Necessary optimizations in the development process can be performed in time to guarantee a high quality of the software.

However, the quality characteristics have to be quantified in more details. For that, existing experiences have to be used e.g. from more finished projects. Especially the human factor, which is quantified by a development factor here, has to be investigated more in detail. A fundamental examination of the error frequency and the number of errors within the development process has to be performed in consideration of the identified influencing factors and conditions. To perform a reliable verification of the concept, real development scenarios have to be executed with a sufficient number of candidates.

REFERENCES

Baheti, R., Gill, H. (2011) "Cyber-physical systems," The Impact of Control Technology, Pages 161-166

Bauer J.M., Bas; G., Durakbasa; M.N., Kopacek; P. (2015). "Development Trends in Automation and Metrology," IFAC (International Federation of Automatic Control), IFAC-PapersOnLine 48-24 (2015) 168–172

German National Academy of Science and Engineering – GNACE (2011) "Cyber-Physical Systems - Driving force for innovation in mobility, health, energy and production, " Acatech Position Paper,

Gosh, S., Biswas, S., Sarkar, D., Sarkar, P.P. (2014) "A novel Neuro-fuzzy classification technique for data mining," Egyptian Informatics Journals, Production and Hosting by Elsevier B.V. on behalf of the Faculty of Computers and Information, Cairo University

Hillmann M., Stühler, S., Schloske, A., Geisinger, D., Westkämper, E. (2014) "Improving Small-quantity Assembly Lines for Complex Industrial Products by Adapting the Failure Process Matrix (FPM) A Case Study," 47th CIRP Conference on Manufacturing Systems CIRP 17 (2014) 236 – 241

Jazdi, N. (2014) "Cyber Physical Systems in the Context of Industry 4.0," IEEE International Conference on Automation, Quality and Testing, Robotics, Cluj-Napoca, Romania

Jazdi, N. (2016) "Lecture notes reliability and safety of automation systems," Institute of Industrial Automation and Software Engineering, University of Stuttgart

Jazdi, N., Maga, C. (2012) "Towards Reliability of Mechatronic Systems," Automation Quality and Testing Robotics (AQTR), 2012 IEEE International

Institute of Electrical and Electronics Engineers IEEE Standard Computer Dictionary (1990)" A Compilation of IEEE Standard Computer Glossaries," IEEE, New York

International Electrotechnical Commission (IEC) (1997), IEC 1131 - PROGRAMMABLE CONTROLLERS, Part 7 - Fuzzy Control Programming

ISO 9000 (2015) "Quality management systems – Fundamentals and vocabulary" ISO9000

Lakshmanan, I., Ramasamy, S. (2015) "An artificial neural-network approach to software reliability growth modelling," 3rd International Conference on Recent Trends in Computing 2015, Ghaziabad, India

Lee, E. (2008) "Cyber Physical Systems: Design Challenges," University of California, Berkley Technical Report No. UCB/EECS-2008-8

McCabe, T.J. (1976) "A Complexity Measure," IEEE Transactions on Software Engineering, pages 308–320

Roy, P., Mahapatra, G.S., Dey, K.N. (2015) "Neuro-genetic approach on logistic model based software reliability prediction," University of Calcutta, India

Software Engineering Institute (SEI) Carnegie Mellon University (2010), "CMMI® for Development, Version 1.3," SEI, Carnegie Mellon University

Watson, A.H., McCabe, T.J. (1996) "Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric," NIST Special Publication

Weyrich, M., Ebert, C. (2016) "Reference Architectures for the Internet of Things," IEEE Software Technology - Computer Society