Development the hybrid code metric for software reliability analysis

Vitaliy Yakovyna, Vasyl Buta

Software Department, Lviv Polytechnic National University,
UKRAINE, Lviv, S. Bandery street 12,
E-mail: yakovyna@lp.edu.ua

Abstract – A hybrid code metric to evaluate the software reliability that takes into account the relative cost of software development at different stages of the lifecycle has been developed. This allows to analyse the software reliability at the early lifecycle stages.

Key words – software reliability, software development lifecycle, software code metrics, deterministic software reliability models.

I. Introduction

Modern technological approaches to software development are complex and often iterative in nature. Activities such as design, coding, testing and maintenance of these approaches are often combined, or follow each other cyclically during the transition from iteration to iteration. The testing process can be carried out continuously as the source code is constantly being updated with new features and fixes. The greater the volume of the source code, the more effort is necessary to maintain the quality of a software system at an appropriate level. To solve this problem, there are different programming methodologies, automated testing, methods of defects detection and prediction of [1].

Application of development methodologies are aimed at reducing the number of defects insertion into the source code. The purpose of the automated testing is also a reduction in the number of defects introduced and prevention of recurrence of previously detected and corrected defects. In identifying of introduced defects to the source code defects than manual testing can only help the detection and prediction of [1]. Means of detecting defects work are based on static analysis of source code. In this case, searches for defects are conducted corresponding to a typical erroneous patterns, such as the use of an uninitialized variable. Therefore, only a small fraction of defects can be detected by such means.

Defect in the case of software is incorrect logic, inappropriate or incorrect command that at runtime may cause failure [2, 3]. In other words, defect is a source of failure, and failure – is an implementation of defects. When a failure occurs, it corresponds to a defect in the program, but the defect may not cause program failure and programnever goes down until the faulty statement is not met. Thus, in contrast to hardware, software failure statistics should take into account scenarios and code coverage tests, otherwise the software reliability analysis can lead to incorrect results, even using adequate body of mathematics.

Similarly to reliability of hardware, the software reliability in the time interval is characterized by probability of no-failure performance for a certain period of time under certain conditions [2, 4]. As a result of the program performance, the input state transforms into output state. Thus the program can be viewed as a function *f*, which transforms input to output where input is the set of all input states, and output is the set of all output states. Input state can be defined as a set of input variables or common commands/transactions in the program [5].

Software reliability models can be divided into two large classes – deterministic (static) and probabilistic (dynamic) [2, 3, 5, 6]. Probabilistic models represent the emergence of failures and defects removal as random events.

Deterministic models are used for study the:

1. elements of the program by counting the number of operators, operands and instructions;
2. flow control program by counting the branching paths and routing performance;
3. data flow programs by examining data sharing and data transfer.

Measuring the performance of deterministic type is derived from the analysis of source text of program, and does not include any random event or value. Deterministic class includes two models [5, 6]: Halstead software model and McCabe cyclomatic complexity model. In general, these models represent quantitative approach to measuring software. Halstead software model is used to evaluate the number of defects in program [5, 6], while the McCabe cyclomatic complexity model is used to determine the upper limit of the programs test number [5, 6].

Nowadays there are many numerical characteristics of software – software code metrics. Among these metrics the following metrics can be distinguished: dimensionally-oriented, complexity metrics and hybrid metrics that combine the advantages of other types. However there is no universal metrics, any controlled metric properties of programs should be monitored either depending on each other, or depending on the specific problem. Adequate analysis and prediction of software reliability is not possible without taking into account reliability characteristics and parameters of the real software realiability models. Therefore, the study of software code metrics and their impact on the software reliability is relevant task of software engineering.

II. Hybryd reliability metric

Complexity metrics are divided into three main groups [7–9]:

- metrics of program size;
- metrics of program control flow complexity;
- metrics of program data flow complexity.

Metrics of programs size are based on the determination of quantitative characteristics associated with the size of the program, and are differentiated by relative simplicity. Metrics of this group are focused on the analysis of the source code, so they can be used to assess the complexity of interim software development. The most well known metrics of this group include the number of program operators, the number of lines of source code, and a set of Halstead metrics [7, 8].

Metrics of program control flow complexity are based on an analysis of the control graph of the program.

Metrics of the second group can also be used to assess the complexity of interim software development. The typical representative of the second group is the McCabe cyclomatic complexity metric [7].

Metrics of program data flow complexity are based on an assessment of the usage, configuration and location of data in the program. In particular this can be applied to global variables. This group includes Chepin's metrics [9].

Different metrics reflect different aspects of software complexity. For a comprehensive account of the data aspects of the software evaluation more than one metric, and their combination should be used. Thus, the hybrid metrics of software code are based on simpler metrics and are their weighted sum.

Since the following typical cost distribution of software development process is generally accepted [10]: 17% – design, 8% – coding, 25% – testing, and 50% – support, the following hybrid metric for reliability assessing based on the Kokol metrics is proposed [11]:

$$HM = 0.17 \times C + 0.08 \times D + 0.25 \times V(G) + 0.5 \times Q,$$

where C – the estimation of the program complexity at the design stage [7]:

$$C = \frac{NI}{NF \times NI_{UN} \times K_{COM}},$$

(here NI is the total number of statistical variables that are passed to the interfaces between the program components; NI_{UN} – the unity value of variables that are passed to the interfaces between the components; K_{COM} – the complexity coefficient of the program);

D – Halstead difficulty metric that reflects the difficulty of program coding [9, 12]:

$$D = \frac{\eta_1}{2} \times \frac{N_2}{\eta_2},$$

(here η_1 – the number of distinct operators; η_2 – the number of distinct operands; N_2 – the total number of operands);

$V(G)$ – McCabe cyclomatic number [9]:

$$V(G) = e - n + 2,$$

(here e is the number of graph arcs, while n denotes the number of vertices);

Q – Chepin's metric as a measure of the complexity of understanding the program [12]:

$$Q = P + 2 \times M + 3 \times CN + 0.5 \times T,$$

(here P is the set of variables for calculations and for output, M is the set of modified or created within the program variables, CN is the set of control variables, while T denotes the set of unused within the program variables).

The proposed hybrid software reliability metric HM can be used during the whole lifecycle to analyse developed software reliability. This makes it possible to reduce the cost of reliable software developing.

Conclusion

The survey of software reliability analysis trends reveals, on the one hand, the need to increase the degree of adequacy of classical reliability models that treat software as a black box, and from the other hand the development of models and methods for software reliability analysis on the basis of architectural approaches and need to establish the correlation between code metrics and software reliability.

A hybrid metric for software reliability evaluation on the basis of Kokol metric is proposed taking into account the relative cost of software developing lifecycle stages. This makes it possible to assess the reliability of software at the early stages of its lifecycle, and thus reduce the cost of software developing with a given reliability value.

References

[1] S. I. Kirnosenko, V.S. Lukianov, "Prognozirovanie obnaruzhenija defektov v programmnom obespechenii [Prediction of software defects revealing]" *Programmnyje producty i sistemy – Software products and systems*, no. 3, pp. 67–71, 2011.

[2] H. Pham, *System software reliability*. London: Springer-Verlag London Limited, 2006.

[3] T. H. Sheakh, S.M.K. Quadri, and V. Singh, "A Study of Analytically Improving the Reliability of Software" *International Journal of Research and Reviews in Computer Science*, vol. 3, no. 1, pp. 1404–1406, February 2012.

[4] A.M. Polovko and S.V. Gurov, *Osnovy teorii nadezhnosti [Reliability theory basics]*. Saint Petersburg: BHV-Petersburg Publ., 2008.

[5] H. Pham and M. Pham, "Software Reliability Models for Critical Applications", Idaho National Engineering Laboratory, EG&G Idaho Inc., EGG—2663 Technical Rep., 1991.

[6] Cobra Rahmani, Azad Azadmanesh "Exploitation of Quantitative Approaches to Software Reliability", University of Nebraska at Omaha, Survivable Networked Systems Rep. (CIST-9900), 2008.

[7] Alan J. Perlis, et al., *Software Metrics*. Cambridge, MA: The MIT Press, 1981.

[8] M. L. Hutcheson, *Software Testing Fundamentals: Methods and Metrics*. Hoboken, NJ: Wiley, 2003.

[9] A. Abran, *Software Metrics and Software Metrology*. Hoboken, NJ: Wiley-IEEE Computer Society, 2010.

[10] D. Wright, *Software Life Cycle Management*. Ely: IT Governance Publishing, 2011.

[11] P. Kokol, "Using spreadsheet software to support metrics life cycle activities" *SIGPLAN Not.*, vol. 24, no 5, pp. 27–32, 1989.

[12] S. H. Kan, *Metrics and Models in Software Quality Engineering*. Indianapolis: Addison-Wesley Professional, 2002.