IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 22, Issue 1, Ser. II (Jan - Feb 2020), PP 06-09
www.iosrjournals.org

On One Model Of Software Fault Identification Theoretical Model

Manana Khachidze, Magda Tsintsadze, Maia Archuadze, Gela Besiashvili
Computer science department/Iv.Javakhishvuli Tbilisi State University, 1 Chavchavadze av, Tbilisi Georgia

Abstract: *Typically the software testing process is aiming Bug Detection, Bug Prevention, User Satisfaction, Software quality and reliability, Recommendations. However, just discovering bug is not enough unless the causes are not identified. Any software can be considered as a complex dynamic system. Accordingly, the testing process is described as a complex multy parameter task. Based on software testing results, paper proposes representation of software qualityas so-called "quality concepts" that form a kind of a library.The "quality concepts" are connected to appropriate "error concepts" that serve a specific purpose of "poor quality" (error or inaccuracy in software provision). The concepts of 'quality' and 'error' are developed using the analytical heuristics method. Subsequently, these concepts will create a library of "defective software templates".*

Key words: *Detect errors, Analytical Heuristics Method, Concepts*

Date of Submission: 15-01-2020 Date of Acceptance: 03-02-2020

I. Introduction

Software is the most complex "live" system whose viability, valid performance, reliability depend on many parameters - from the correctness of the task set before its creation, to the technical and logical flaws of the hardware. Therefore, it is only logical to compare software testing to human health testing. To get a complete picture it is important to identify values of typically different (measurable, descriptive, visual, etc.) parameters by assessing their interdependencies.

It is well known that software is mainly divided into three types depending on the common function, type and application (Application software, System software and Computer programming tools). Considering the fact that each type includes several more subtypes as well, one software may contain different subtypes of software, the number of software types might be considered as infinite. Furder we will be considering an Application software.

Application software can be of general purpose (database management software, text processing software, web browsers, etc.) or have a specific purpose (accounting for various goods, workflow, etc.). All types of software must meet the quality requirements.

Our work is an attempt to present a model that will be serving for mutual compliance of the parameters that reflect the software quality attributes.

II. Software Quality Attributes Classification

When evaluating software, it is important to distinguish five major classes of quality attributes: Functional attributes, Operational attributes, Usability attributes, Business attributes, Structural attributes [1]. Lets denote each attribute by $A_i, i = 1, \dots, 4$ (in our notation Business attributes, which are features of software product design, use, and development, are not included). In turn some of these attributes include sub-attributes as well. Denote these sub-attributes by $a_{i,j}, i = 1, \dots, 4; j = 1, \dots, n_i$ (n_i is the number of sub-attributes of i-th attribute).

Lets represent a complete list of software quality attributes based on our proposed notations:

Functional attributes (A_1), which are features of software input/output, include two types of attributes - Boolean and statistical. The presence of first type of attribute for software is not mandatory, but the second one will be presented (more or less) anyway. The Boolean attribute is: Correctnes ($a_{1,1}$) and Robustness ($a_{1,2}$). As for the statistical one, it is expressed in Dependability and Security. Dependability by itself contains two sub-attributes: reliability ($a_{1,3}$) and safety ($a_{1,4}$), and Security contains four sub-attributes: - Confidentiality ($a_{1,5}$), Integrity ($a_{1,6}$), Authentication ($a_{1,7}$), Availability ($a_{1,8}$).

Operational Attributes (A_2), which represent the software operating conditions, are characterized by four attributes: Latency ($a_{2,1}$), Throughput ($a_{2,2}$), Efficiency ($a_{2,3}$), Capacity ($a_{2,4}$).

Usability attributes (A_3), describe the software product's usability degree and its ability to adapt to user requirements. These types of attributes are the following sub-attributes describing: Ease of Use ($a_{3,1}$), Ease of Learning ($a_{3,2}$), Customizability ($a_{3,3}$) , Calibrability ($a_{3,4}$), Interoperability ($a_{3,5}$).

Structural attributes (A_4), which characterize the software structure, are expressed in four sub-attributes: Design Integrity $(a_{4,1})$; Modularity, which consists of two sub-attributes Cohesion ($a_{4,2}$), Coupling ($a_{4,3}$); Testability - It also has two sub-attributes: Controllability ($a_{4,4}$) and Observability ($a_{4,5}$); Adaptability ($a_{4,6}$).

With help of received set of attributes $\{a_{i,j}\}, i = 1, \dots, 4; j = 1, \dots, n_i$ we can describe any other S_k Software (in terms of quality). For simplicity let's denote $a_{i,j}$ attributes with α_h , $h = 1, \dots, 23$ (23 is the number of software quality attributes defined by us.

Thus we'll have:

$$S_k \rightarrow \check{\alpha}_1 \& \check{\alpha}_2 \& \cdots \& \check{\alpha}_{23}$$

where

$$\check{\alpha}_i = \begin{cases} \alpha_i \ , if\ \alpha_i\ is\ the\ S_k\ software\ characteristic\ attribute; \\ \bar{\alpha}_i \ , if\ \alpha_i\ is\ not\ the\ S_k\ software\ characteristic\ attribute. \end{cases}$$

The different S_k software quality compliance will allow us to formulate a general notion in the "language" of analytical heuristics [2, 3], meaning software with the corresponding quality of the five evaluations presented. The aggregation of such implicants will serve as a base for method for identifying software flaws [4].

III. Software Failure, Error, And Fault

The set of software Failure, Error, and Fault can be infinite as the software itself, but there is still their classification. To describe our method, we stopped at the most widely used Software Failure, Error, and Fault [5, 6] - Performance Errors, Interaction Errors, Losing control errors, Syntactic Errors, Error managing errors, Computation Errors and Management circulation errors. Let's also try to formalize their most often causes as well [7, 8].

Denote Flaws by $F_m, m = 1, \dots, 7$.Each type of flaw might describe number of "problems" it contains. Lets denote these "problems" by $f_{m,n}, m = 1, \dots, 7; n = 1, \dots, l_n$ (l_n is the number of "problems" in m-th flaw).

For instance: Performance Errors (F_1) might contain problems: Long Load time ($f_{1,1}$), Poor response time ($f_{1,2}$), Poor scalability ($f_{1,3}$), Bottlenecking ($f_{1,4}$); Interaction Errors (F_2) includes problems associated with Functionality ($f_{2,1}$), Communication ($f_{2,2}$), Command Structure and Entry ($f_{2,3}$), Missing Commands ($f_{2,4}$), Program Rigidity ($f_{2,5}$), Performance ($f_{2,6}$), Output ($f_{2,2}$) , etc.

Using these notations, we can present the results of each specific software test as follows [9]:

Totally the number of problems identified by us is 40. Lets for simplicity introduce the following notations: denote the $f_{i,j}$ problem by p_x , $x = 1, \dots, 40$.

$Test(S_k)$, will be denoting the operation of testing of S_k software. The set of problems identified duringthe testing process might be presented with help of the implicant $\check{p}_1 \& \check{p}_2 \& \cdots \& \check{p}_{40}$

where

$$\check{p}_i = \begin{cases} p_i \ , if\ p_i is\ the\ S_k\ software\ testing\ resalt\ problem; \\ \bar{p}_i \ , if\ p_i\ is\ not\ the\ S_k\ software\ testing\ resalt\ problem. \end{cases}$$

By collecting such descriptions of the test results we will have a database that can serve as a basis of the knowledge base.

IV. The Method of Analytical Heuristics

Suppose we have some set of software $\{ S_k\}, k = 1, \dots, K$. Lets represent each software test results using notations offered in the previous paragraph, thus:

$$\begin{aligned} Test(S_1) &\rightarrow \check{p}_1^1 \& \check{p}_2^1 \& \cdots \& \check{p}_{40}^1, \\ Test(S_2) &\rightarrow \check{p}_1^2 \& \check{p}_2^2 \& \cdots \& \check{p}_{40}^2, \\ &\cdots \\ Test(S_K) &\rightarrow \check{p}_1^k \& \check{p}_2^k \& \cdots \& \check{p}_{40}^k \end{aligned}$$

Also assess the software quality using software quality attributes and represent them using quality describing implicants (The quality attribute in the description is considered as "positive", and if the software fails to meet any quality attribute in the application it will be "negative – not positive"):

$$S_1 \rightarrow \breve{\alpha}_1^1 \& \breve{\alpha}_2^1 \& \cdots \& \breve{\alpha}_{23}^1$$
$$S_2 \rightarrow \breve{\alpha}_1^2 \& \breve{\alpha}_2^2 \& \cdots \& \breve{\alpha}_{23}^2$$
$$\cdots$$
$$S_K \rightarrow \breve{\alpha}_1^k \& \breve{\alpha}_2^k \& \cdots \& \breve{\alpha}_{23}^k$$

Group the quality implants in a uniform manner (software in accordance with the same quality evaluations) and present it in a normal disjunctive form. For clarity: if we suppose $S_1^{'}, S_2^{'}, \ldots, S_z^{'}$ software have common Qx quality evaluations, then their general description will be:

$$\breve{\alpha}_1^{1'} \& \breve{\alpha}_2^{1'} \& \cdots \& \breve{\alpha}_{23}^{1'} \vee \breve{\alpha}_1^{2'} \& \breve{\alpha}_2^{2'} \& \cdots \& \breve{\alpha}_{23}^{2'} \vee \breve{\alpha}_1^{z'} \& \breve{\alpha}_2^{z'} \& \cdots \& \breve{\alpha}_{23}^{z'}$$

By minimizing this normal disjunctive form, we'll get the "concept" describing the Qx estimation described by the quality attributes. The set of relevant concepts of various assessments will be one of the key parts of our 'knowledge base'.

Group Test (S_1),…, Test (S_k) Implants of relevant test deficiencies according to the results of the quality assessments. Thus, for example, in a disjunctive normal form, implicants describing software testing errors corresponding to Qx- will be united:

$$\breve{p}_1^{1'} \& \breve{p}_2^{1'} \& \cdots \& \breve{p}_{40}^{1'} \vee \breve{p}_1^{2'} \& \breve{p}_2^{2'} \& \cdots \& \breve{p}_{40}^{2'} \vee \cdots \vee \breve{p}_1^{z'} \& \breve{p}_2^{z'} \& \cdots \& \breve{p}_{40}^{z'}$$

By minimizing this normal disjunctive form, we get "concepts" that describe errors in software Qx quality evaluation. For each different Qy quality evaluation we will have a different 'concept', and the set of such concepts will serve as a second major part of our 'knowledge base'.

V. The Scheme For Checking Software Faults

Once we'll complete both parts of the Knowledge Base (Quality and Error Concepts) it will be possible to "predict" other software errors. The process is as follows:

1. The software is checked against the presence or absence of quality attributes and the result is written as an implicant;
2. Software quality identifying implicant will be compared to the "concepts" describing the "knowledge base" assessment and where appropriate the final quality Qx is fixed;
3. From the knowledge base of "concepts" of errors, a specific "concept" is selected, including a list of possible interrelated errors, in accordance of Software Quality Qx

VI. Conclusions And Future Work

The system developed on the basis of the proposed method will allow us to evaluate software assessed by certain attributes of software quality with help of one notion and in accordance with this notion identify the specific deficiencies that led to the specific "defect" in program quality.

The method is currently tested on mock software and the results are promising. However, for further perfection, it is necessary to identify all the attributes, software errors and inaccuracies in accordance of software types.

References

[1]. P.K. Chaurasia, R.A. Khan. Classification of Software Requirement Errors: A Critical Review. International Journal of Computer Applications (0975 – 8887) Volume 132 – No.7, December 2015. Pp. 9-14

[2]. Chavchanidze V. V. Analiticheskiye evristiki iskussnvennogo intellekta pri formirovanii ponyatiy, opoznanii obrazov i klassifikatsii obyektov. VINITI. № 2080-70 Dep. (Analytical Heuristics of the Artificial Intellect in the Formation of Notions, Identification of Images and Classification of Objects, VINITI, #2080-70 Dep.

[3]. Chavchanidze V. V. Analiticheskoye Resheniye Zadachi Formirovaniya Ponyatyi i Raspoznavaniya Obrazov (Analytical Resolution of the Problem of Identification of Notions and Recognition of Images). Proceedings of the Academy of Sciences of the Georgian SSR, vol.63, No.1, 1971.

[4]. A.J. Ko, B.A. Myers. A framework and methodology for studying the causes of software errors in programming systems. Journal of Visual Languages and Computing 16 (2005) 41–84

[5]. P. Tr¨oger, L. Feinbube, M. Werner, What activates a bug? a refinement of the laprie terminology model, in: Software Reliability Engineering (ISSRE), 2015 IEEE 26th International Symposium on, IEEE, 2015

[6]. Yang Feng, James A. Jones, Zhenyu Chen, Chunrong Fang. An Empirical Study on Software Failure Classification with Multi-Label and Problem-Transformation Techniques. 2018 IEEE 11th International Conference on Software Testing, Verification and Validation . 0-7695-6388-0/18/$31.00 ©2018 IEEE DOI 10.1109/ICST.2018.00039. pp. 320-330

[7]. D. Lo, H. Cheng, J. Han, S.-C. Khoo, and C. Sun. Classification of software behaviors for failure detection: a discriminative pattern mining approach. In Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 557–566. ACM, 2009.
[8]. Podgurski, D. Leon, P. Francis, W. Masri, M. Minch, J. Sun, and B. Wang. Automated support for classifying software failure reports. In Software Engineering, 2003. Proceedings. 25th International Conference on, pages 465–475. IEEE, 2003
[9]. Ali Mili, Fairouz Tchier. Software Testing: Concepts and Operations (Quantitative Software Engineering Series) 1st Edition. Published by John Wiley & Sons, Inc., Hoboken, New Jersey Published simultaneously in Canada. 2015. 400 p

Manana Khachidze, et.al, On One Model Of Software Fault Identification Theoretical Model." *IOSR Journal of Computer Engineering (IOSR-JCE)*, 22.1 (2020), pp. 06-09.