http://www.moderntechno.de/index.php/meit/article/view/meit25-02-022
DOI: 10.30890/2567-5273.2023-25-02-022

UDC 004.2

PRACTICAL ANALYSIS OF SOFTWARE QUALITY

Kudryavtsev O. A.
Postgraduate student
Cherkasy state technological university,
Cherkasy, bul. Shevchenka, 460,18000

Abstract. *Success of any project is defined by its ability to satisfy the consumer needs. That is why providing high quality is a necessary task for any production, including software engineering. Software quality is a complex subject. Standards distinguish quality of processes of development, internal and external quality of software, quality of software at stage of using. There is a set of metrics defining quality of software for each of the quality components. Obtained structure is called a model of software quality. Software metric is a measure allowing to receive a numerical value of some aspect of software or its specifications, as well as method of calculation of such value. Metrics allow receiving numerical values of each software aspect or its specifications. Metrics of software complexity are of special interest. Complexity is an important factor which can impact other quality parameters such as accuracy, reliability, correctness, maintainability.*

Keywords: *software quality; software testing; quality metrics*

Introduction Software testing is one of quality control techniques. It includes test management, test design, test execution and test analysis.

Purposes of software testing:

- Detection of defects.
- Raising quality level.
- Supplying information for decision making.
- Preventing the occurrence of defects.
- Supplying information about software quality to end customer.

Software testing lifecycle is a part of software lifecycle. They must be synchronized with each other. Design and development of testing may be as complicated and time-consuming as software development itself. If testing is not done with initial release of software, many problems will be identified at later stages of development. As a result, product release is often postponed due to long period of software bug fixing which almost neutralizes the advantages of iterative development.

Purpose of this paper The purpose is to present developed software. Its main function is to calculate metric measuring quality of software code.

Statement of basic materials Let us consider the detailed definition of “Software testing”.

At different times and in different sources, testing is defined differently. Let us look at some of the definitions.

Lisa Crispin and Janet Gregory defined software testing as a process of program execution with a goal to find mistakes [2].

Cem Kaner said that testing is an intellectual discipline with a purpose to receive reliable software without spending too much effort on its checking [3].

Glenford Myers, Tom Badgett, Corey Sandler noted that software testing is a conformity check between real and expected program behavior at finite number of

tests completed in a certain way [1].

Robert Culbertson, Chris Brown and Gary Cobb in their book "Rapid testing" described software testing process as a process of observation of program execution in special conditions and scoring of any aspects of its work based on such observation [4].

In a textbook "Software verification" testing is a process which is aimed to identify situations where program behavior is wrong, undesirable or where it does not conform to specification [5].

Boris Beizer in his work noted that software testing is a process which includes all lifecycle activities, both static and dynamic, relating to planning, preparation and evaluation of software and related work results in order to determine that they meet the declared requirements and to define defects [6].

Automated testing is one of the types of software testing. Automated testing uses software tools for testing and verification of correctness of the results. It simplifies and accelerates testing. Main advantage of automated testing is the ability of repetition of testing without human involvement. Automated testing at the code level is the traditional and most popular way among developers. The second way of automatization of testing is to imitate user actions using special instruments (GUI-testing).

Let us consider the concept of software product. Studying quality of a software product at development phase implies working with the following metrics:

- Complexity.
- Correctness.
- Usability.
- Reliability.
- Performance.
- Mobility.

Of all these metrics it is complexity which is best subjected to formal evaluation. Moreover, complexity management is the key to receiving adequate values of other metrics, such as correctness, reliability, performance.

The better software is engineered the lower is its complexity: the lower is its complexity the easier developer navigates the program and writes new code, the lower the probability of creating new bug and the higher the chance to identify existing one. Lower software complexity significantly decreases development time and its maintenance cost.

Software complexity metrics are divided into three main groups:

- Metrics of software size.
- Metrics of complexity of software control flow.
- Metrics of complexity of software data flow.

The central points are used as the main statistical characteristics of the distribution of metrics for classes in the collection:

- Mathematical expectation – mean value of random variable:

$$x = \frac{1}{n} \sum_{i=1}^{n} x_i$$

– Variance – an indicator of scattering of values of random variable relative to its mathematical expectation:

$$x^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

– Skewness characterizes occurrences when different reasons favor more frequent appearances of values above or below mean. Right-skewness (positive) means that lower values of distribution appear more frequently, left-skewness (negative) – higher ones:

$$A = \frac{\sum_i (x_i - \bar{x})^3}{n\sigma^3}$$

– Kurtosis characterizes the degree of sharpness of the peak of the distribution of a random variable and is determined by the formula:

$$E = \frac{\sum_i (x_i - \bar{x})^4}{n\sigma^4} - 3$$

Dimension driven metrics directly measures software product and its development process. The most used software source code metric is Source Lines of Code (SLOC). It displays the size of the software project.

SLOC (Source Lines of Code) metric. Initially, SLOC had been used when programming languages had been relatively simple. One line of code in such languages had been equal to one language command. Later programming languages developed and became much more flexible, so one line of code can now contain multiple language commands. Taking this into consideration, two versions of SLOC metric were developed:

1. Number of "physical" SLOC (abbreviated as LOC, SLOC, KLOC, KSLOC, DSLOC) is a total number of source code lines, including commentaries and empty lines (when measuring the indicator for the number of empty lines, as a rule, a restriction is used – their number is taken into account, not exceeding 25% of the total number of lines in the measured code block).

2. Number of “logical” SLOC (abbreviated as LSI, DSI, KDSI, where SI – Source Instructions) is a total number of commands. It depends on the used programming language. In this case, if language does not allow putting multiple command on one line, the number of “logical” SLOC will be equal to the number of "physical" SLOC except for empty lines and commentaries. If the programming language supports putting multiple commands on one line, one “physical” line must be measured as multiple “logical” ones if it has more than one language command.

With respect to calculating SLOC metric here, it may be noted that there is no single universally agreed and oriented approach acceptable for different programming languages. In the most cases, SLOC is defined as a total number of lines of code except for empty lines and commentaries. Usually a number of “physical” SLOC is counted and multiple operands on the same line are not taken into account.

There are many products of SLOC metric for obtaining different indicators of a project. The main products are:

- ✓ Number of Blank Lines of Code (BLOC);
- ✓ Number of Comment Lines of Code (CLOC);

- ✓ Number of Lines with Both Code and Comments (C&SLOC);
- ✓ Number of lines with declarative source code;
- ✓ Number of lines with imperative source code;
- ✓ Percentage of commentaris (number of lines with commentaries multiplies by 100 and divided by a number of lines with code);
- ✓ Mean value of lines for functions (methods);
- ✓ Mean number of lines for modules
- ✓ Mean number of lines for classes.

Initial data for calculation of LOC metrics:

Performance = Length (thousands, LOC) / Expenses (people-month);

Quality = Errors (units) / Length (thousands, LOC);

Unit cost = Cost (thousands) / Length (LOC);

Documentation = Pages of a document (pages) / Length (thousands, LOC).

Description of developed software After the analysis of environments for writing automated tests, Selenium IDE was chosen.

Selenium IDE (integrated development environment) is an easy-to-use browser extension which helps to develop test scenarios for web pages. This instrument allows recording a certain scenario of user behavior at a website and then play the recorded actions automatically and edit it later according to requirements.

During development of a complex project, it becomes inconvenient to contain all code in one file. That is why it is split to separate modules. To build all these modules, developers uses build command. Besides, building process may include compilers. Instead of JS/CSS their dialects (CoffeeScript, SCSS, LESS, SASS) can be used.

Jenkins is an open-source instrument for continuous integration written in Java. It was forked away from Hudson after a dispute with Oracle. Jenkins provides services of continuous integration for software development.

For realization and running of an automated test web browser, Mozilla Firefox was chosen. Selenium IDE extension was installed. After opening Selenium environment, it is possible to create a new project immediately and start recording actions for execution.

After creating a project, an address for beginning execution of actions was determined. It was a Jenkins localhost address.

After a page with this address opens, it is necessary to enable actions recording that will be executed by a user through website interface by pressing a “Start recording” button in a Selenium environment window. It is necessary to open project in Jenkins in a recording actions mode and assemble a new collection with the latest changes in the code, wait until collection finishes executing and go to a number of its latest version. Then it is necessary to open a console and follow a link to SonarQube, where one can open and see the results of scanning of a project in details (Figure 1).

Now it is necessary to copy required data to write them to a code of an automated test. For this, it is needed to enable the inspector in the interface of Selenium window and choose data that need to be saved for their further processing. After choosing data at the website via the inspector in a current step in automated test code, an element of code of a web-site is recorded. There necessary data are stored.

At each next running of the automated test at this step, new data which will be displayed in this element of the website code will be copied.

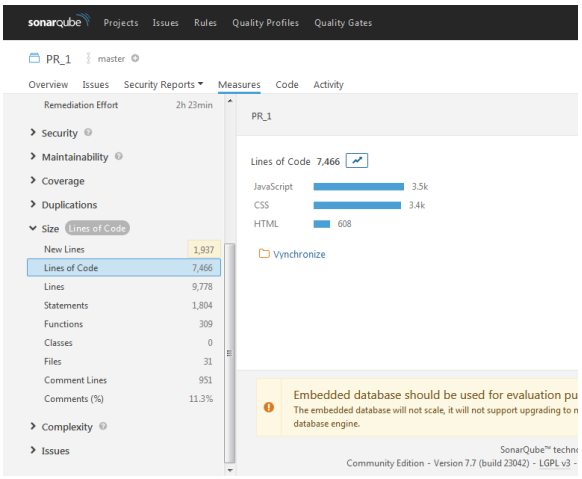

Fig. 1 – Window with results of scanning of a project in SonarQube

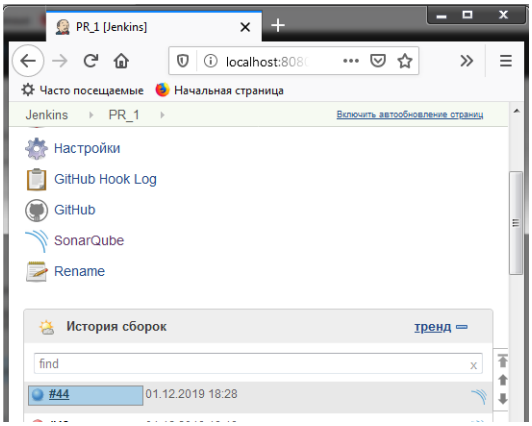

Fig. 2 – Recording actions mode in Selenium IDE

After saving the necessary data at Selenium, one should set a transition to the page where the form for entering the saved data is located in order to calculate this data using a formula in accordance with the selected metric. In the input field, one must enter previously saved data from SonarQube, such as:

- Performance;
- Quality;
- Cost;
- Documentation.

Summary and conclusions. For calculation of the data based on the results of analysis of project code at SonarQube the SLOC metric was selected.

The obtained data at SonarQube can be calculated by SLOC metric formula. These data need to be written to variables at Selenium IDE to be able to use them in data calculation form. Transferring data to the input fields is performed by identifying each field with an identifier and transferring the corresponding data to the corresponding fields. To execute a function that will calculate data from the form fields, one needs to create a button for calculation and assign an event that will launch the calculation function every time after clicking the button to it.

Fig. 3 – Results of calculations using the SLOC metric formula

Therefore, we can conclude that with the help of the developed system, it will be possible to calculate metrics that will represent the quality of the program code.

References

1. Glenford Majers, Tom Badzhett, Kory Sandler The art of software testing. Moscow: Dialektika., 2012. 272 c.
2. Lajza Kryspyn, Dzhanet Gregory Agile Testing: A Practical Guide for Software Testers and Agile Teams. Moscow: Viliams., 2010. 464 c.
3. Kaner Kem, Folk Dzhek, Nguen Eng Kek Software testing. Fundamental Concepts of Business Application Management. Kiiv: DiaSoft, 2001. 544 c.
4. Kalbertson Robert, Braun Krys, Kobb Gery Rapid testing. Moscow: Viliams, 2002. 374 c.
5. Synyczin S. V., Nalyuty`n N. Yu. Software verification Moscow: Binom, 2008. 368 c.
6. Bejzer B. Functional testing technologies for software and systems. Piter: SPb, 2004. 320 c.
7. Unit testing [electronic source]. Available: https://msn.khnu.km.ua/pluginfile.php/208290/mod_resource/content/2/%D0%9B%D0%A0%20%E2%84%964.pdf
8. Unit testing [electronic source]. Available: https://studfile.net/preview/14533269/
9. Types of testing and differences between them [electronic source]. Available: https://www.quality-assurance-group.com/vydy-testuvannya-ta-vidminnosti-mizh-nymy-shpargalka-z-testuvannya-chastyna-4/
10. Software system context [electronic source]. Available: https://stepik.org/lesson/106620/step/1?unit=81144
11. Principles of testing [electronic source]. Available: https://qalight.com.ua/baza-znaniy/pochemu-testirovanie-neobhodimo/
12. Odokienko S., Luta M., Savchenko Y. Practical analysis of the source of source software. Management of Development of Complex Systems, Kiyv, 44, 2020