General and Professional Education
1/2015 pp. 57-62
ISSN 2084-1469

TESTING EDUCATION: TEST CASE PRIORITIZATION USING MATRICES

Marek Żukowicz
Rzeszów University of Technology
Faculty of Electrical and Computer Engineering
Department of Computer and Control Engineering
ul. W. Pola 2, 35-959 Rzeszów
e-mail: bobmarek@o2.pl
Dawid Warchoł
Rzeszów University of Technology
Faculty of Electrical and Computer Engineering
Department of Computer and Control Engineering
ul. W. Pola 2, 35-959 Rzeszów
e-mail: dawwar@kia.prz.edu.pl

Abstract: In this paper we describe the problem of assigning priorities to test cases before the beginning of software testing. This problem occurs when testers do not have enough time to test every function. Therefore, it must be specified which test cases are the most important, which are medium important, and which can be omitted incurring little risk of the production environment failure. We describe a method of test case prioritization using a table called the report matrix. Such matrices can be used to introduce some quality and functionality measures of a version of software as well as to allow the comparison between different versions.

Keywords: test case prioritization, report matrix, quality measures.

Introduction to test case prioritization problem

Designing of tests is an activity that defines **what functionality has to be tested and how to perform the tests**. The process of test designing usually includes the aims and scope of the tests. These aims consist of strategic and test objectives as well as other designing criteria or acceptance criteria specified by the stakeholders. In IT companies there is often not enough time to perform every test that describes functionality of the tested system. The lack of time may be the result of adding new requirements to iterations or changing them, which sometimes takes place within companies. There exist systems tightly functionally integrated with various applications. The automation of such systems' tests is practically impossible or very hard and expensive which makes it unprofitable. Software developers often have to decide which test cases should be performed first in order to determine whether a given version of software is stable enough to reach the client with the right quality. Failure to deliver the software in time may cause the loss of customer's trust or expose a company to additional costs what stakeholders do not want to happen.
In practice it is sometimes hard to prioritize the particular test cases based only on the specification. Moreover, the priorities are not always optimally set which may lead to release of an unstable version of software. Such a situation is unwanted by every person associated with the IT project. The assignment of priorities to particular test cases with the appropriate levels can be done after the test conditions are specified and there is available information sufficient to enable the creation of low, medium, and high level test cases.
The purpose of this paper is to present the problem of assigning priorities to test cases, as well as to propose a quality measure of functionality based on previous tests. We

propose using matrices and the properties of particular rows for test case prioritization describing functions of the testing system in the case of long projects in which the **iterative-incremental** model of software development is used. Assigning priorities to test cases is a problem for which there exists no specific strategy. We present a method that can significantly contribute to solving the problem and lead to introducing new measures of software quality.

Related work

The problem of test case prioritization is the subject which attracts the interest of scientists. Several interesting solutions of the problem can be found in literature. The work [1] is an introduction to the regression tests problems. The authors explain why the test case prioritization should be of interest to testers and scientists. It is one of the first works that address this issue. The authors of [2] present an approach to separate functionality tests to white-box (full access to the source code), black-box (no access to the source code), and gray-box (limited access to the source code) tests. They describe an algorithm that maximizes coverage of the tested program functionality and minimizes the number of tests by measures called distances. These measures specify the extent to which tests meet the given requirements. In [4] the heuristic algorithm named "aggregate-strength prioritization" strategy (ASPS) is presented. It assigns priorities to test cases in the context of negative tests whose aim is to look for defects causing the failure of a tested system. The authors of [6] describe the prioritization issue as a set in which permutations are defined. The objective function, mentioned in this work, is defined in such a way that it can be used in the algorithm solving the problem of ordering the test cases. The method is also oriented to look for defects in tested software.

The work [3] is the review paper. It contains diagrams and charts showing the comparison of the selected algorithms related to our subject. In the paper [5] authors mathematically prove that there cannot exist one good strategy of solving every optimization problems of a particular type. Therefore, it is reasonable to search for new algorithms and strategies of solving problems such as testing optimization, an example of which can be test case prioritization. This represents the main motivation of our paper. The approach described in the following sections can be applied to white-box or black-box functional testing. It is not oriented to looking for defects, but to confirm the quality of tested software and to compare the quality of different versions of software.

Iterative-incremental model of software development

The iterative-incremental model assumes that in particular time another versions of software with new functions are created. In this method, the changes in requirements are possible between the increments of functions. The particular processes accompanying the software development can be modified and work schedule can be updated. During each increment a programmer can use the knowledge acquired in previous increments. This knowledge comes both from the process of software development, and from the possibility of working on a particular finite part of the system.

From the software development methodology point of view, the incremental method begins with the specification of requirements and the creation of initial general project of the entire system called the base project. Then, a subset of functions is chosen with respect to given criteria. The next step, according to the cascade model, is the creation of detailed project, based on which the part of the system with the chosen functionality is later developed. After the tests are performed, the created part of the system can be given to the client.

Regression testing

A **software regression** means the loss of some features in a new version of software that usually results in an error message, a logical error or the lack of action. Regressions are caused by changing some parts of the program code. As a consequence of these changes some functions may stop working. To find such errors, the **regression testing** may be used. It is a type of software testing able to determine the correctness of functions that were present in previous versions of a program. Regression testing reveals defects caused accidentally during the process of program optimization. Specifying which test cases should be performed first in regression testing, can significantly increase the likelihood that a potential user will get a working stable version of software in relatively short time.

Applying matrices to solve the problem of test case prioritization

Formalization of the problem

Suppose that we have a problem of assigning priorities to test cases. Therefore, some symbols must be introduced in order to formalize the problem. Let $T = \{t_1, ..., t_m\}$ be the set of all test cases which describes the functional tests in a particular version of system. We define the mapping:

$$g : T \to \{l, m, h\} \tag{1}$$

which assigns priorities to test cases in a way that *h*, *m*, and *l* are high, medium, and low priorities, respectively. Practically, in systems there always exist crucial functions that has to work correctly and thus, the set *T* must be divided in the following way:

$$T = T_1 \cup T_2, T_1 \cap T_2 = \Phi \tag{2}$$

Where T_1 denotes the test cases with the high priority which cannot be changed; T_2 defines the test cases whose priorities can change depending on the given criteria. Let $R = \{R_1, ..., R_p\}$ be the set of the test reports. For the report R_j we define the mapping $r_j : T \to \{0,1\}$ which describes the acceptance of a test case: 1 - a test passes (acceptance conditions are satisfied), 0 - a test fails, (acceptance conditions are not satisfied). If the amount of test cases with the fixed high priority is equal to *s* in a particular iteration (e.g., one month) of software development, then the set $T_2 = \{t_{s+1}, ..., t_m\}$ and only these cases might have their priorities changed. There is, however, a possibility that some test from T_1 will be moved to T_2 for some reason, e.g., changing the work system or organization system in a company to which the software is dedicated. Therefore, the set T_1 should also be considered in the process of assigning priorities. According to the iterative development model, new functions are sometimes added to software. These functions may not be present in some number of first reports. For this reason, we conventionally assume that the priorities are assigned to the test cases which are present in at least three reports. The report matrix $A(T) = [a_{ij}]$ is defined, where:

$$a_{ij} = r_j(t_j). \tag{3}$$

For each row of the matrix *A* the function $f : T \to [0,1]$, $c : T \to \{1,2,3,...\}$ is defined, where:

$$f(t_i) = \sum_{j=p+1-c(t_j)}^{p} \frac{a_{ij}}{c(t_i)} \tag{4}$$

while specify the number of reports containing the test case t_i and *p* is the number of all reports. The function *c* is necessary because a test case can appear in a third, fourth or later report or it may be omitted in some report for any reason. This can happen, e.g., when a new client does not require some functionality and there is not enough time to test every function. The function *f* should be interpreted as a measure of quality of the tested functionality that entails the risk of system failure. Such a failure may be caused by an error in a particular function present in a new version of software.

An example of applying report matrices to assign priorities to test cases

Assume that we have to perform the tests of software after its 10 iterations of development and 30 already performed tests, from which 10 has the high priority. Assume also that the execution time of all tests is relatively long, and therefore, the order of their execution will depend on their priorities. A report from the previous tests can be seen in table 1. Rows 1 to 10 describe the test cases with fixed high priority (from the set T_i).

For each row of table 1, the value of *f* is calculated based on the equation (4). The results are presented in table 2 and table 3.

Table 1. Report matrix from the 10 previous iterations of software development

Number of test case / Number of report	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9	R_{10}
1	1	1	1	0	1	1	0	1	0	1
2	1	1	1	1	1	1	1	1	1	1
3	0	1	1	1	1	1	1	1	1	1
4	1	1	1	1	1	0	1	1	1	0
5	1	0	1	1	1	1	0	0	1	1
6	1	1	1	0	0	0	0	1	1	1
7	1	1	1	1	1	0	1	1	1	1
8	1	1	1	1	1	1	1	1	0	0
9	1	1	1	1	0	1	0	1	1	1
10	1	0	0	1	1	1	1	1	1	1
11	0	1	1	1	1	1	1	1	1	1
12	0	1	1	1	0	1	1	1	1	0
13	1	1	1	0	1	1	0	0	1	1
14	1	0	0	0	1	1	1	1	1	1
15	1	1	1	1	1	1	1	1	1	1
16	1	0	0	1	1	1	0	1	1	1
17	1	1	1	1	1	1	1	1	1	1
18	1	1	1	1	1	0	1	0	1	1
19	1	1	1	1	0	0	1	1	1	1
20	X	0	1	1	1	1	1	1	1	1
21	X	0	0	1	1	1	1	0	1	1
22	X	1	0	1	1	1	1	1	1	1
23	X	1	1	1	1	1	1	1	1	1
24	X	0	1	0	1	1	1	1	1	1
25	X	1	1	0	1	1	1	1	1	1
26	X	1	1	0	1	1	0	1	1	0
27	X	X	1	1	1	1	1	1	1	1
28	X	X	1	1	1	1	0	1	1	1
29	X	X	X	1	0	1	0	1	1	1
30	X	X	X	1	1	1	0	0	0	1

Table 2. Values of function f for the test cases (1)

Number of test case	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Value of *f*	0,7	1	0,9	0,8	0,7	0,6	0,9	0,8	0,8	0,8	0,9	0,7	0,7	0,7	1

Table 3. Values of function f for the test cases (2)

Number of test case	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Value of *f*	0,7	1	0,8	0,8	0,89	0,67	0,89	1	0,78	0,89	0,67	1	0,88	0,71	0,57

Having a table with calculated values of *f* from the previous reports, we shall adopt the criteria of function *g* according to which the priorities will be assigned to test cases. Such activity is usually performed by test director, manager or analyst (in large companies). An example definition of g is presented in equation 5:

$$g(t_i) = \begin{cases} l, & f(t_i) \geq 0{,}9 \\ m, & f(t_i) \in (0{,}7;0{,}9) \\ h, & f(t_i) \leq 0{,}7 \end{cases} \qquad (5)$$

Thus, after the 10 iterations, the test cases will have the priorities as shown in table 4 and table

5. According to function g, the order of testing in the next iteration will be as follows:
1) 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14, 16, 21, 26, 30 - high priority;
2) 18, 19, 20, 22, 24, 25, 28, 29 - medium priority;
3) 11, 15, 17, 23, 27 - low priority.

Table 2. Designated priorities of the test cases (1)

Number of test case	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Priority	h	l	H	m	h	h	l	m	m	m	l	h	h	h	l

Table 3. Designated priorities of the test cases (2)

Number of test case	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Priority	H	l	m	m	m	h	m	l	m	m	h	l	m	m	h

Having the matrix $A(T)$ with all test cases, we can proceed in the same way. Ranges, according to which function g assigns the priorities, can be chosen freely. They may depend on the testers' desired degree of software quality assurance. The question begs - what should we do if the time intended for testing is limited whilst there are many test cases with high and medium priority? Such decision belongs to the person responsible for the quality assurance. The ranges may be differently defined so that the assignment of high and medium priorities will not be rigorously conditioned. There is also a possibility that only the new functions will be tested and only the regression tests for high priority cases will be performed. In practice, it is very rare that the majority of the test cases do not satisfy the acceptance criteria in every iteration. However, if this occurs, one should consider the cessation of maintenance of such defective system.

Introduction of software quality and functionality measures using report matrices

Software quality measures and comparing of tests from previous iterations are of great importance since they allow to decide whether the software development or the tests are going in the right direction. Applying the matrix $A(T)$ may introduce some software quality measures allowing to compare different versions of program. Considering the entire matrix, the quality measure $m(A(T)): A(T) \to [0,1]$ can be defined by the following equation:

$$m(A(T)) = \frac{\sum_{i=1}^{p_k} \sum_{j=1}^{m_k} \gamma(a_{ij})}{p_k \cdot m_k - \partial(A(T)_k)} \qquad (6)$$

where:

p_k - number of reports after the iteration k;
m_k - number of test cases after the iteration k;
$\partial(A(T)_k)$ - number of $A(T)$ elements with X value after the iteration k;

$$\gamma(a_{ij}) = \begin{cases} 1, & \text{if } a_{ij} = 1 \\ 0, & \text{if } a_{ij} \in \{0, X\}. \end{cases}$$

Assume that we want to compare the software versions between the seventh and the eight iteration using table 1 and equation (6). For this purpose we have to calculate $m_7(A(T))$ and $m_8(A(T))$:

$$m_7(A(T)) = \frac{20}{30} = 66{,}66\%,$$

$$m_8(A(T)) = \frac{26}{30} = 86{,}66\%.$$

After the eight iteration, the quality of the version of software was greater by 20%.
Based on the above equations it is easy to introduce the way of comparing particular functions and areas. One should confine to test cases associated with the area that has to be compared. Introducing the denotation $m_k(A(T|S)): A(T) \to [0,1]$ we can define the measure of area after the iteration k:

$$m_k(T|S)=\frac{\sum_{j=1}^{h}\gamma(a_{kj})}{h-\partial(A(T|S)_k)} \qquad (7)$$

where:

$T|S$ - subset of test cases associated with particular function/area;

h - number of elements of the S.

One can easily notice that equation (7) is the generalization of equation (6). Comparing the subsequent versions of software in the context of the particular areas, there is also a possibility to collate the developers' skills (in larger projects), which can be useful for deciding about the training or the reorganization of programmers' work.

Conclusion

The purpose of the paper was to provide decision support about an order of performing test cases. Every field of industry is usually associated with some software. In the real-time systems testing takes about 80% of the time spent on a project. In the case of long-term projects, there is a need to manage testing during the software maintenance and versioning phase. There is often not enough time to perform every test. The authors propose a method for testing optimization using matrices as well as software quality and functionality measures. The advantages of using matrices for test case prioritization are as follows:

1.The method is easy to understand and implement, e.g., as a computer program or a spreadsheet.

2.Tests from different iterations can be compared with each other.

3.There is a possibility to compare different functions and, based on this comparison, to observe the performance of a particular function against others.

4.One can introduce the quality measure of a single function and its changes in the application lifecycle.

5.Particular versions of a tested application can be compared as a whole to versions from other iterations.

Nowadays, software testing is as much important as programming and the information technology is present in almost every area of life and industry. It is, therefore, worth writing about testing as well as increasing knowledge in this subject, not only from the technical, but also from the educational side by proposing new solutions for testing-related issues.

References

1. Agrawal H., Horgan J.R., London S., Wong W.E., *A Study of Effective Regression Testing in Practice,* IEEE International Symposium on Software Reliability Engineering, 1997.
2. Blostein D., Hassan A. E., Hemmati H., Thomas S.W., *Static test case prioritization using topic models,* Springer Science+Buisnes Media, LLC 2012.
3. Catal C., Mishra D., *Test Case prioritization: a systematic mapping study,* Springer Science+Buisnes Media, LLC 2012.
4. Chan A.T.S., Chen J., Lu Y., Huang R., Towey D., *Aggregate-strength interaction test suite prioritization*, The Journal and Systems and Software 99, 2015.
5. Ho Y., Pepyne D. L., *Simple Explantation of No Free Lunch Theorem of Optimization,* IEEE, Conference on Decision and Control, Orlando, Florida USA, 2001, IPCSIT vol.14 (2011) © (2011) IACSIT Press, Singapore 225.
6. Sun F., Yan L., *Regression testing Prioritization Based on model Checking for Safety-Crucial Embedded Systems*, 2013 Fourth International Conference on Digital Manufacturing & Automation.