Software Engineering 2012, 2(4): 165-173
DOI: 10.5923/j.se.20120204.10

Mobile Applications Testing Processes Metrics and Optimization Criteria

V. A. Filippov[1,*], E. E. Khatko[2]

[1]Moscow Institute of Electronics and Mathematics of National Research University a higher School of Economy, Moscow, Russian Federation
[2]Moscow Institute of Physics and Technology, State University, Moscow, Russian Federation

Abstract Because of rapid mobile technologies expansion, there is a gap between the complexity of mobile applications and the complexity of employed testing techniques. This paper is aimed at reducing the gap from the theoretical point of view. The paper comprises an analysis of mobile applications testing processes, mobile applications testing metrics, along with the full test coverage criterion. It also contains an integral criterion of the testing processes optimization which is based on the idea of summing the corresponding sub-processes times. The presented criterion leads to an assumption of the tests generation approach efficiency. Therefore a partial criterion of the tests generation process is proposed. The mathematical model of this partial criterion is based on the properties of different algebraic expressions. The numerical results section includes processes comparison and some estimates.

Keywords Mobile applications, Testing, Tests automation, Tests generation

1. Introduction

In the last few years, mobile technologies have been rapidly expanding in everyday life. Almost every person on earth has a mobile telephone. Mobile devices are becoming more and more complex as new types of devices such as Smart phones, Communicators, Tablets, etc., have appeared. Such devices hardware has configurations similar to yesterday's desktops', and thus they should be treated as complex hardware and software systems controlled by operating systems. In the era of conventional phones, testing processes were simple as well. Manual testing processes predominated. However, accelerated evolution of mobile devices leads to formation of a technology gap between the complexity of mobile applications and the complexity of the corresponding testing methods. Manual testing is no longer enough these days. A new complex approach with special test automation tools should come in its place. Therefore the problem of testing methods optimization is very important from the exploratory point of view and urgent from the practical point of view.

An analysis of existing testing approaches of mobile applications is presented in the first section of this paper. An iterative scheme is proposed as the most common for mobile applications (MA) development. MA development and testing approaches are formalized based on the iterative scheme.

MA testing metrics and the corresponding full test coverage criterion is presented in the second section.

Using different algebraic expressions, the third section comprises the mathematical model of the iterative development scheme. A similar model is proposed in (1), but in testing processes are described apart from the development processes.

An integral criterion of development and testing processes optimization based on the provided mathematical model is presented in the fourth section. An assumption of test generation techniques efficiency is made based on the presented criterion. This assumption can also be stated from the (2) research which also describes testing processes optimization techniques. But it's worth mentioning that techniques in (2)are presented in a common way, without providing any mathematical model.

The partial criterion of tests generation process efficiency is developed onwards. The test generation problem is presented in (3) and (4). But (3) is circuits domain specific, therefore some specificity of circuits was used, while providing the optimization proposals.(4)provides some basic formal generation approaches with no concern to their optimization.

2. An Analysis of Development and Testing Processes of Mobile Applications

2.1. Analysis of Development Processes

* Corresponding author:
filbob@infoline.su (V. A. Filippov)
Published online at http://journal.sapub.org/se

The process of mobile applications (MA)development is usually performed iteratively (5).This scheme is appropriate for small-scale projects of MA. The peculiarity of this process is in accelerated progress to a released product with several iterations of the full development cycle. The detailed iterative scheme is provided in Figure 2.

The cycle of iterations begins after functional requirements are determined. The iterations include the build process, the system testing of new functionality and regression testing. One of the iterations always ends with a decision that the next iteration would be inappropriate. This leads to a new beta version release. In the iterative scheme MA meets testing on a system stage, bypassing the module and integration testing stages. Therefore the MA testing process is usually run on the application UI level.

Recently, the MVC (6)architecture has been greatly expanding because of its scalability and simplicity. The following are the definitions of the MVC components in the context of MA testing.

Models – data base program inter faces. Models provide the means of data management for the application. To test the application data management processes, one should divide all data accepted by the application into equivalence classes (7)and check the application responses with each class.

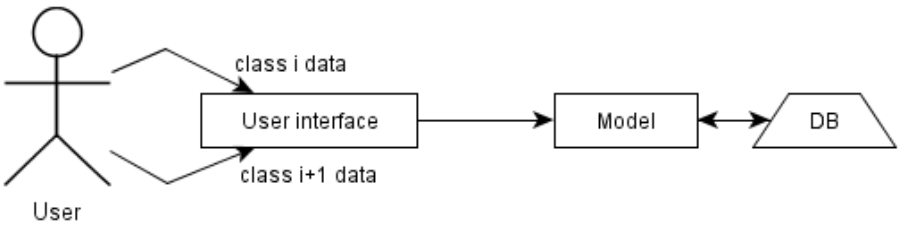

Figure 1. Different equivalence classes

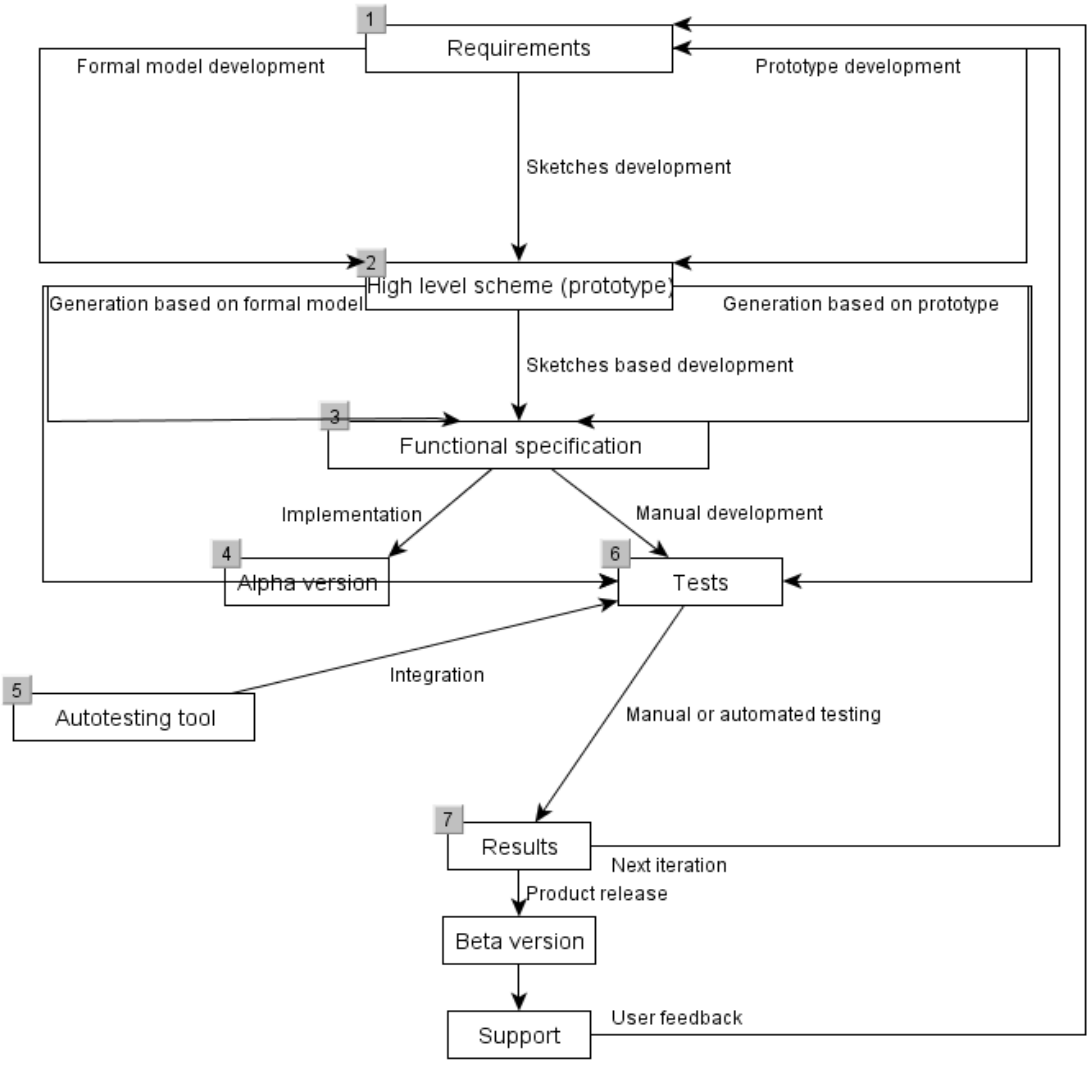

Figure 2. Detailed iterative process scheme

Controllers – components that encapsulate the application logic. Controllers are responsible for the application logic; therefore they control the global UI structure of the application as well.

Views – graphical representations of the application states characterized by the UI elements sets.

The following is a common user-application interaction algorithm:

1. A User discovers a view with UI elements (buttons, edit fields, etc.). These elements can be treated as holders of the application controllers' methods. Each holder initiates a request to a controller.
2. The User's request pulls a controller's method; then the controller processes the request data and requests the database through the model if necessary.
3. Having processed the request the controller chooses the next view to "show" to the user as a response to his request.

The reviewed architecture allows testing internal modules of an application on the UI level. The main feature here is to divide all accepted data into equivalence classes.

2.2. Analysis of Testing Processes

The following is the detailed iterative process scheme: Figure 2.

This figure represents the formal model of the iterative development scheme. Basic development milestones with activities needed for moving between them are presented.

In this scheme it's possible to choose a transition between each process state. The overall application development time depends on the path from the 1st state to the 7thone and on the number of cycles. There are different paths that guide the process from 1 to 7, and each of them defines the corresponding testing method. Let's consider the most common of these methods.

2.2.1. Manual Documentation, Manual Testing

1. Requirements elicitation
2. Sketches development
3. Functional specification manual development
4. Test cases manual development
5. Manual testing

This method is very flexible; it allows achieving any test coverage. But it's very time consuming. This method will be referred to as the MM method.

2.2.2. Formal modal based Development

1. Requirements elicitation
2. Formal model development
3. Specification generation based on the formal model
4. Test cases generation based on the formal model
5. Automated testing

This method is simple to automate, but it's not very flexible in the MA context, because for mal models are not suitable for describing user inter faces. Therefore one can't reach required testing coverage with tests derived from a formal model. This method won't be referred toin this paper anymore.

2.2.3. Manual documentation, Automated testing

1. Requirements elicitation
2. Sketches development
3. Functional specification manual development
4. Test cases manual development
5. Tests automation
6. Automated and manual testing

Tests automation requires textual description fall the actions being automated. Basically, all tests have to be manually developed (as in the MM method) before the automation begins. This method requires a lot of time for tests development at first, but saves time during regression testing later. Therefore this approach is effective when a lot of iterations take place. This method will be referred to as the MA method.

2.2.4. Generated documentation, Manual and Automated Testing

1. Requirements elicitation
2. Prototype development
3. Specification generation based on the prototype
4. Test cases generation based on the prototype
5. Automated and manual testing

Proto type sallow describing an application use rinter faces quickly and precisely. Therefore it allows achieving the desired testing coverage. This method also allows generating different types of test cases, if applying special rules for prototype creation. This brings in extra flexibility to the testing process. This method will be referred to as the PR method.

2.3. Analysis of Tests Automation Methods

There are different tests automation tools; in particular some of them are reviewed in (1)and (8).

In general, there are two main approaches to tests automation:

The program matic automation approach is based on a programming language use, particularly its special libraries. Tests automation is then reduced to programming special modules, classes, and testing scripts. This is a flexible approach, in the sense, that it does not require creating new scripts in case of functionality changes.

The playback tools automation approach is based on so called "playback" tools. This approach does not require software engineering skills. Tests auto mation is reduced to running a test on a DUT, but proxy-ing the steps to an appropriate recorder. Recorded tests can be replayed later on a DUT. With this approach, creating a test is basically its execution. The main drawback of this method is that any application change requires recording the test again.

3. Testing metrics of Mobile Applications

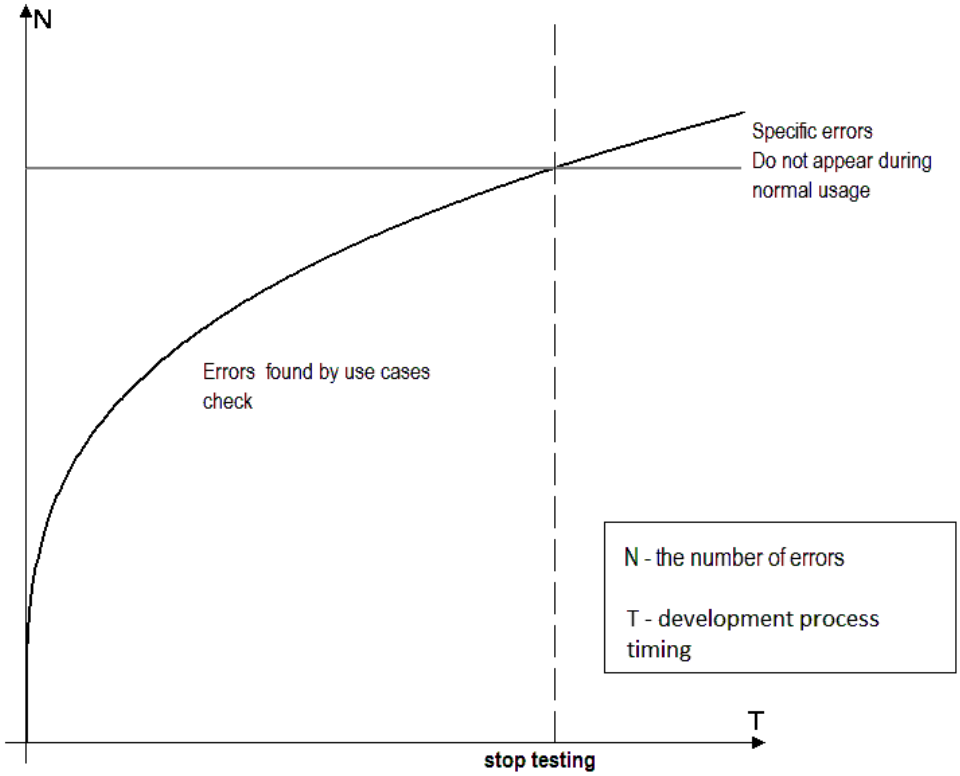

Figure 3. Testing process ending point

The goal of MA testing in the context of the rapid MA development process is not to find all the errors of the application, but to find the errors that are likely to occur during user-application interactions. Thus only some of equivalence classes are considered during the testing process. Such testing goal does not exclude all the errors; however, it's unlikely that the end user will ever find any of the remaining errors. The following figure represents the moment of the test process completion.

This figure contains the dependence of the number of application errors from the time of application arrival to the end user.

This way it's possible to formulate the following statements which define mobile applications testing metrics and a full testing coverage criterion:

- The testing goal is to find the errors that occur during user-application interactions. Appropriate equivalence classes reduction should be done when developing tests.
- All the remaining equivalence classes should be taken into consideration when testing
- Full testing coverage can be achieved with UI-level testing

These statements lead to the following definitions of MA testing metrics and the test coverage criterion:

Mobile applications testing metrics – the number of checked application responses to UI elements impact from an end user (assuming that the user impact is based on common application use cases).

Full test coverage criterion is to check the responses to the requests handled by all UI elements with appropriate equivalence classes based on the application use cases.

4. Integral Criterion of Testing Process Efficiency

Let's denote the path from state 1 to state 7 in Figure 2as $p_{12} - p_{23} - p_{34} - ... = p_{ij} (ij \in [12, 23, 34, 36, 46, 56, 67])$. Let $T(p_{ij})$ be the time and $N(p_{ij})$ - the coverage, corresponding to N - checked responses. Thereat, time and coverage testing process optimization can be represented as:

$$\begin{cases} N(p_{ij}) \to \max \\ T(p_{ij}) \to \min \end{cases},$$

$(ij \in [12, 23, 34, 36, 46, 56, 67])$.

Since N is an integer, the expression above can be expressed as:

$$T = \frac{\sum_{ij \in [12,23,34,36,46,56,67]} t_{ij}}{N} \to \min \quad (1)$$

Where t_{ij} is time spent on the corresponding transition that depends on the applied testing method. N - overall coverage of checked responses.

This expression defines an **integral criterion** of the testing process efficiency and can be used to compare different testing approaches in terms of time and test coverage.

Let's review each summand in the fraction numerator. Let t be abstract time corresponding to a time period of the manual requirement creation that describes functionality corresponding to one UI element. Apparently, all summands from the (1) expression except t_{56} depend linearly on t. $t_{46} \approx 0$, because it's the time period of transfer ring the application to a test house (can be neglected). So far as t_{56} operation should be run once, it can be neglected as well. Let's consider the following factors that represent summands dependencies on t:

C_{md} - the model or prototype development factor. Represents relative model or prototype development time per one UI element

C_{sd} - the specification development factor. Represents relative specification development time per one UI element

C_{imp} - the implement ation factor. Represents relative implementation time per one UI element

C_{td} - the tests development factor. Represents relative tests development time per one UI element

C_{test} - the testing process factor. Represent relative time required to check all use cases correspondent to one UI element

Let K be the number of iterations in the iterative scheme. Let $\partial N_i (i = 1..K)$ be the number of extra responses (the result of built up functional requirements) to check. Let's sum all of the (1) summands, factoring out the time t, then (1)reduces to:

$$T = \frac{\sum_{ij \in [12,23,34,36,46,56,67]} t_{ij}}{N} = \\ = \frac{t(C_{md}N + C_{sd}N + C_{imp} + C_{td}N + C_{test}N)}{N} + \sum_{i=1..K} \frac{t(C_{md}\partial N_i + C_{sd}\partial N_i + C_{imp}\frac{\partial N_i}{N} + C_{td}\partial N_i + C_{test}N)}{\partial N_i} \tag{1.1}$$

It's worth mentioning that during the i-th iteration:

1. Implementation time of added function ality is $\frac{N}{\partial N_i}$ times less than implementation time of all functionality (on average).

2. Regression testing is always applied to all of the functions, and not only to new ones.

Let $\overline{\partial N}$ be the average number of added responses to check during K cycles; then let's convert (1.1), having $\sum_{i=1..K} C_i \frac{t\partial N_i}{N + \partial N_i} = C_i \frac{Kt\overline{\partial N}}{N + \overline{\partial N}}$ to:

$$T = \frac{t(C_{md}N + C_{sd}N + C_{imp} + C_{td}N + C_{test}N)}{N} + \sum_{i=1..K} \frac{t(C_{md}\partial N_i + C_{sd}\partial N_i + C_{imp}\frac{\partial N_i}{N} + C_{td}\partial N_i + C_{test}N)}{N + \partial N_i}$$

$$= t(C_{md} + C_{sd} + C_{td} + C_{test}) + \frac{tC_{imp}}{N} + \frac{Kt\overline{\partial N}}{N + \overline{\partial N}}(C_{md} + C_{sd} + \frac{C_{imp}}{N} + C_{td}) + KtC_{test}\frac{N}{\overline{\partial N} + N}$$

The average number of added responses is negligible compared to all the responses: $\frac{\overline{\partial N}}{N} \approx 0$, compared to N then:

$$T = t(C_{md} + C_{sd} + C_{td} + C_{test}) + \frac{tC_{imp}}{N} + \frac{Kt\overline{\partial N}}{N + \overline{\partial N}}(C_{md} + C_{sd} + \frac{C_{imp}}{N} + C_{td}) + KtC_{test}\frac{N}{\overline{\partial N} + N}$$

$$= t(C_{md} + C_{sd} + C_{td} + C_{test}) + \frac{tC_{imp}}{N} + \frac{Kt\overline{\partial N}}{N}(C_{md} + C_{sd} + C_{td} + \frac{C_{imp}}{N}) + KtC_{test}$$

$$= t\left[(C_{md} + C_{sp} + C_{td})(1 + \frac{K\overline{\partial N}}{N}) + C_{test}(K+1) + C_{imp}\frac{N + K\overline{\partial N}}{N^2}\right]$$

So the optimization problem can be represented as:

$$T = t\left[C_{test}(K+1) + (C_{md} + C_{sd} + C_{td})(1 + \frac{K\overline{\partial N}}{N}) + C_{imp}\frac{N + K\overline{\partial N}}{N^2}\right] \to \min \quad (2)$$

After reviewing each summand of the expression (2), it can be stated that the C_{test} factor and the appropriate process should be optimized in the first place, because it's next to the $(K+1)$ factor, and the sum of $(C_{р.мод.} + C_{п.спец.} + C_{п.тест.})$ and appropriate processes should be optimized in the second place.

It's also worth mentioning that in the case of tests automation the C_{td} -related process should include the processes of manual tests development and automation tests development. And the $C_{test} = C_{run} + C_{res}$ - testing process consists of "running tests" and "result estimation" processes.

Expression (2) represents quantification of the testing process time consumption, in the case of K iterations of the iterative development process with the coverage of N responses to the corresponding functional requirements.

When estimating the efficiency of the testing processes it's safe to neglect the $C_{imp}\frac{N + K\overline{\partial N}}{N^2}$ summand in expression (2), because it represents the development side of the whole process, but not the testing side.

Having carried out a deeper analysis of the reviewed methods the description of which is beyond the scopes of this paper, the following assumptions have been made about the proposed methods and factors:

Method	Assumption
MM	$\begin{cases} C = C_{md} = C_{sd} = C_{td} \\ C_{run} = 2C \\ C_{res} = 0 \end{cases}$
MA (programmatic automation)	$\begin{cases} C = C_{md} = C_{sd} \\ C_{td} = C + \frac{10}{K}C \\ C_{run} = 0 \\ C_{res} = 0.5C \end{cases}$
MA (playback automation)	$\begin{cases} C = C_{md} = C_{sd} \\ C_{td} = C + 2C \\ C_{run} = 0 \\ C_{res} = 0.5C \end{cases}$
PR	$\begin{cases} C = C_{md} = C_{sd} \\ C_{td} = 0 \\ C_{run} = 0 \\ C_{res} = 0.5C \end{cases}$

Testing processes efficiency is the ratio of the MM process time and the given process time:

$Eff(K) = \frac{T_{MM}}{T_{process}}$, where $\frac{\overline{\partial N}}{N} = 0.1$ is assumed for estimation purposes.

$$\begin{cases} T_{MM} = 2C(K+1) + 3C(1+0.1K) = (2.3K+5)C \\ T_{MA_{programmatic}} = 0.5C(K+1) + 2C(1+0.1K) + C(1+\frac{10}{K})(1+0.1K) = \frac{0.7K^2+10}{K}C + 3.5C \\ T_{MA_{playback}} = 0.5C(K+1) + 5C(1+0.1K) = C(K+5.5) \\ T_{PR} = 0.5C(K+1) + 2C(1+0.1K) = C(0.7K+2.5) \end{cases}$$

The efficiency then will be expressed as:

$$\begin{cases} Eff_{MA_{programmatic}}(K) = \dfrac{2.3K+5}{\dfrac{0.7K^2+10}{K}+3.5} = \dfrac{2.3K^2+5K}{0.7K^2+3.5K+10} \\ Eff_{MA_{playback}}(K) = \dfrac{2.3K+5}{K+5.5} \\ Eff_{PR}(K) = \dfrac{2.3K+5}{0.7K+2.5} \end{cases} \quad (2.1)$$

5. Partial Criterion of Generation Process Efficiency

The efficiency of the PR method may be assumed from the result of integral criterion review. The base of the PR method is an effective tests generation technique. To be able to generate tests, one should develop an application prototype, based on appropriate rules(9), extract an EFSM (extended finite state machine) from the prototype(10)and represent this EFSM as a graph with parameters and conditions. Generated tests may then be used for manual or automated (programmatic approach) testing.

Numerical metrics of the generation technique in this case would be:

N - the number of test cases generated by the algorithm

L - the average length of a generated test case

For a particular technique, both of these parameters depend on the inputted graph G of the appropriate EFSM. Let's consider the graph complexity concept $comp(G)$ as a numerical representation of "how difficult it is to generate a full coverage test suite" with a particular generation technique for the provided graph. Graph complexity should consider the number of graph nodes, transitions, parameters and conditions. It also should be kept in mind that dynamic graphs (with parameters and conditions) are "more difficult" to traverse. It's suggested to use an rms expression for the graph complexity and to make use of a numerical factor to increase the priority of parameters and conditions over the number of nodes and transitions:

$comp(G) = \sqrt{K^2 + M^2 + (VC)^2 + (VP)^2}$,

where

K - the number of graph nodes

M - the number of graph transitions

C - the number of graph conditions

P - the number of graph parameters

V - the priority factor

Then the partial criterion of the generation process will be expressed as:

$$\boxed{F(G) = \frac{comp(G)}{N(comp(G)) \times L(comp(G))}} \quad (3)$$

where

G - the MA prototype's EFSM graph,

$comp(G) = \sqrt{K^2 + M^2 + (VC)^2 + (VP)^2}$ - complexity of the graph.

This figure represents a test generation process. The provided partial criterion (3)measures generation efficiency in context of this process.

In fact, the product $N(comp) \times L(comp)$ estimates the number of user requests to the application (e.g. button clicks) that should be done to achieve the required coverage. Therefore a lower value of the product corresponds to greater efficiency of the generation technique. The $comp(G)$ in the fraction numerator of (3)normalizes the efficiency value so that it would not be dependent on the graph itself.

6. Numerical Results

Let's plot the reviewed methods efficiency using the expression

$$\begin{cases} Eff_{MA_{programmatic}}(K) = \dfrac{2.3K+5}{\dfrac{0.7K^2+10}{K}+3.5} = \dfrac{2.3K^2+5K}{0.7K^2+3.5K+10} \\ Eff_{MA_{playback}}(K) = \dfrac{2.3K+5}{K+5.5} \\ Eff_{PR}(K) = \dfrac{2.3K+5}{0.7K+2.5} \end{cases} \quad (2.1):$$

$$\begin{cases} Eff_{MA_{programmatic}}(K) = \dfrac{2.3K^2+5K}{0.7K^2+3.5K+10} \\ Eff_{MA_{playback}}(K) = \dfrac{2.3K+5}{K+5.5} \\ Eff_{PR}(K) = \dfrac{2.3K+5}{0.7K+2.5} \end{cases}$$

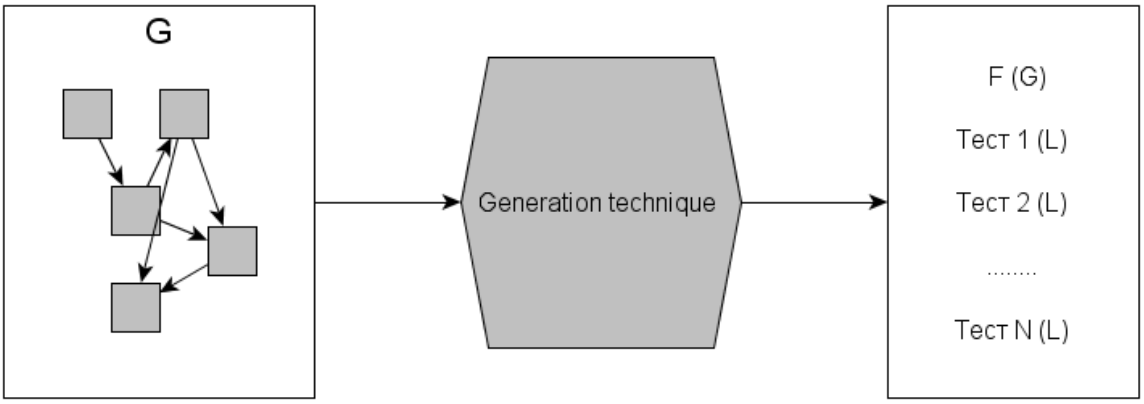

Figure 4. Generation process efficiency metrics

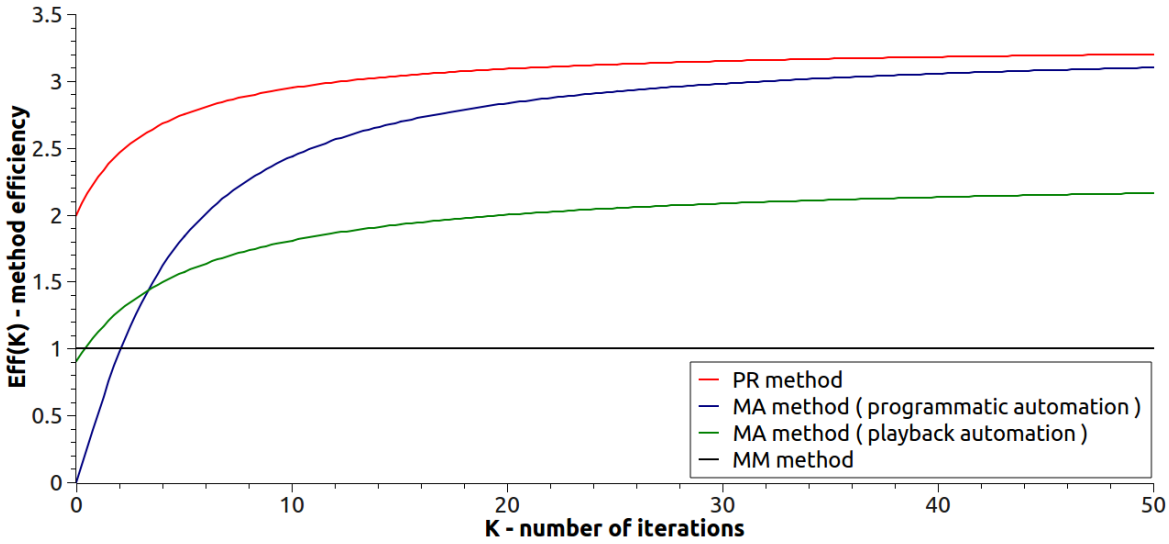

Figure 5. Testing methods efficiency

This figure presents the dependency of the testing process efficiency from the number of iteration cycles. The number of cycles for MA normally resides within [10..30] values. The process efficiency is calculated as the ratio of the sum of

K cycles time of MM and the given processes. The following features can be distinguished in this figure:

- The PR method is the most efficient even with K values between $[5..10]$
- Efficiency calculation is based on the MM method, therefore this method has constant efficiency on the plot

The significance of the proposed partial criterion is in the capability to measure different test generation techniques based on the graph traversal approach. This criterion abstracts the measurement procedure from the concrete design under test.

7. Conclusions

The paper comprises an analysis of mobile applications testing processes, mobile applications testing metrics, along with the full test coverage criterion. It also contains an integral optimization criterion of testing processes which leads to an assumption of a tests generation approach efficiency. Therefore a partial criterion of the tests generation process is proposed. The numerical results section includes processes comparison and proves the taken assumptions.

The significance of the proposed integral criterion is in the capability to measure different testing methods integrated into the iterative development process.

The significance of the proposed partial criterion is in the capability to measure different tests generation techniques and algorithms based on the graph traversal approach.

REFERENCES

[1] Современные проблемы фундаментальных и прикладных наук. Хатько, Е. Е. Москва, Долгопрудный : МФТИ, 2009. Один из подходов к анализу системы тестирования сложных программных комплексов. Vol. 1, pp. 104-107. 52.

[2] Heiskanen, H., Maunumaa, M. and Katara, M.Test Process Improvement for Automated Test Generation. Tampere : Tampere University of Technology, Department of Software Systems, 2010.

[3] Xijiang, L, Pomeranz, I and R., Sudhakar M.Techniques for Improving the Efficiency of Sequential Circuit Test Generation. Iowa : University of Iowa.

[4] Ural, H. Formal methods for test sequence generation. Computercommunications. 1992, Vol. 15.

[5] Кулямин, В. В.Методы верификации программного обеспечения. Москва : Институт системного программирования РАН.

[6] Томас, Д. and Х., Хэнссон Д.Гибкая разработка веб-приложений в среде Rails. Москва. Санкт-Петербург : Питер, 2008.

[7] Степанченко, И. В. Эквивалентное разбиение. Методы тестирования программного обеспечения. Волгоград : РПК "Политехник", 2006.

[8] Хатько, Е. Е. and Филиппов, В. А. Проблемы качества тестирования программного обеспечения для мультизадачных пользовательских комплексов. Качество. Инновации. Образование. 3 2011, Vol. 3, pp. 32-35.

[9] Хатько, Е. Е. Об одном методе тестирования «мобильных» приложений. Труды МФТИ. 3, 2012, Vol. 4, pp. 132-140.

[10] Карпов, Ю. Г.Теория автоматов. Санк-Петербург : Питер, 2003.