Software testing, cybernetic process

Authors: *Professor Ion IVAN, PHD.*[1], *Assistant Professor Cristian CIUREA, PHD.*[2], *Alin ZAMFIROIU, PHD. Candidate*[3], *Mihai Liviu DESPA, PHD. Candidate*[4], *Catalin SBORA, PHD. Candidate*[5] *Emanuel HERTELIU, PHD. Candidate*[6]
The Bucharest University of Economics, Bucharest, Romania

Abstract

The objective of this article derives from the need to analyse the importance of the testing process from the perspective of mobile applications, innovative applications, structured entities and collaborative security components. The software testing process is described in the context of software quality assurance. The necessity of software testing is illustrated and objectives and deliverables within the testing process are defined. Elements that determine the quality of testing processes are emphasized. A two-dimensional approach to software testing characterized by method and level is disclosed.

The article is centred on a cybernetic approach, because the process structure depends on the software product's response to changes. Specific aspects of the testing process are identified for mobile applications, for widespread implementation of structured entities, for innovative software development projects and for developing collaborative security. Testing methods are submitted. The correlation between testing level and development stages of software application is illustrated.

Keywords: *testing, mobile applications, innovative projects, collaborative security, structured entities*
JEL Classification: L86

Testing processes

Software application testing is a process, or a series of processes, designed in order to ensure that the written code does what it was meant to do and does not do anything that was not intended [1]. An untested functionality is an incomplete functionality. An untested software application is an application that cannot be launched, delivered or properly used. Therefore the testing process is a critical stage in the software development cycle.

The testing process, for web applications, is particularly important in the quality assurance effort. One of the main goals of testing is to provide an overview of the application's overall quality [5]. Testing an application requires not only searching and reporting malfunctions but also compliance of implemented functionalities with the designed architecture and initial specifications. In software development applications the aim is to test the following parameters:

Compliance is the extent to which the application's features follow the designed architecture, graphic concept and usability flows. Within the testing process the extent to which the application's design complies with the original layout is evaluated and the extent to which the application's functionalities meet the tasks for which they were designed. Compliance is assessed by comparing the features resulting from the application development process with specifications submitted by the project owner and the parameters defined in the planning stage. In practice the project owner is the most qualified tester for assessing the compliance of the application that is being developed.

[1]ionivan@ase.ro

[2] cristian.ciurea@ie.ase.ro

[3] zamfiroiu@ici.ro

[4] mihai.despa@yahoo.com

[5] catalin.sbora@gmail.com

[6] emanuel.herteliu@gmail.com

Usability of an application is given by the ease with which a user accesses and uses its functionalities. Within the testing stage the intuitiveness degree and the ease of reaching the sought functionality should be evaluated. In terms of usability the minimum number of clicks required to access functionality is a relevant indicator. For usability testing, crawler-type instruments are used to map the application.

Reliability is represented by the loading speed of pages, the response times of the various functionalities and by the behaviour when accessed using low-speed Internet connections. In terms of reliability the loading speed of pages is relevant only when the request to the server hosting the application, are launched from different parts of the globe. The reliability is also tested by accessing the applications using devices with different hardware and software configurations. In order to test the reliability of an application, online tools such as performance testing platforms [18] and load testing portals [19] are used.

Repeatability of a software application is given by the degree of predictability when aiming for a specific result. Within the testing stage, evaluation is centred on the characteristic that imposes that for identical input data sets, same output result should be obtained. Repeatability is assessed with the help of test scenarios. All members of the testing team that uses the same testing scenarios should obtain identical results in order to validate the repeatability characteristic of the application.

Availability is represented by the extent to which the application can be accessed. Within the testing process all scenarios that lead to the impossibility of accessing the application must be identified. Availability is tested by simulating a large number of operations that the application must execute without failure.

Security is represented by extent to which private data and information are protected. Within the testing stage security breaches and vulnerabilities, which can be exploited by users with malicious intent, should be identified. Security tests are targeted mainly on forms because they facilitate interactions with the database.

In order to ensure the testing's process quality of a software application, a wide range of possible scenarios must be covered. This is achieved by a two-dimensional approach to the testing process. The two analysed dimensions are method and level of testing.

The method of testing includes all the tools and techniques used in the process. The proposed methods are:

- **black – box** is the method that involves isolating the source code, the database and the development procedures and implementation methods; testing personnel does not know any details about the application's development process but only details about the functional aspects [6]; black-box method focuses on analysing the functionalities of the application and does not require testing personnel to have the knowledge or skills of a programmer; this method positively influence the quality of the application by highlighting opportunities to improve functionality, uncovering malfunctions and emphasizing usability issues;
- **white – box** is the method that is based on providing full access to the source code for all personnel involved in the testing process; the white - box method aims mainly to analyse the internal structure of the application; testing personnel must have extensive programming knowledge and skills in order to work effectively with the white-box method [6]; this method affects positively the quality of the application by discovering opportunities for optimization of the source code, of the architecture, of unnecessary code sequences and of the need for further explicitness of the source code;
- **grey – box** is the method which requires that testing personnel have partial access to the source code, database, configuration elements and architecture of the application; the grey - box approach allows the tester to direct the testing process to the critical elements of the application [6]; testing personnel should have basic programming knowledge and skills; the

grey – box method positively influences the quality of the application by facilitating earlier detection of integration problems that may arise during assembly of different modules.

The testing level is closely linked to the development stages of the application and involves the level of detail sought by the testers. The proposed test levels are:

- **unit** – testing is performed at code level; testing at this level is performed to ensure the accuracy of functions or that of sub-assemblies; Unit level testing is done by programmers and takes place during the development stage;
- **integration** – testing is employed when there is integration of several sections of an application that were developed independently, by different programming teams or by different members of the same team; integration testing is performed on a new module or functionality that is being added to an existing application;
- **system** – testing is carried out in order to host the application on the production environment, where it will run after the official release; system testing is carried out on an environment that is identical to the production environment and is aimed at verifying the existence of the necessary prerequisites for the designed architecture.
- **acceptance –** testing is performed by the project's beneficiary or owner on the production environment and serves to validate compliance of the delivered application with the original specifications.

Table 1. Level and method correlation

	Unit	Integration	System	Acceptance
Black - box		*	*	*
White - box	*	*	*	
Grey - box		*	*	

Table 1 presents the proper way to apply testing methods according to the level they are intended for. It is recommended to use a certain method only for the level that is marked with an asterisk.

The particularities of testing process in mobile applications

Mobile applications are code sequences and instructions that run on a mobile device in order to solve a specific problem.

According to [7] testing mobile applications is divided into two stages:

- **emulator testing** - involves testing through certain software modules which simulate operating on a real mobile device;
- **real devices testing** - involves testing of applications on mobile devices for which they were designed and developed.

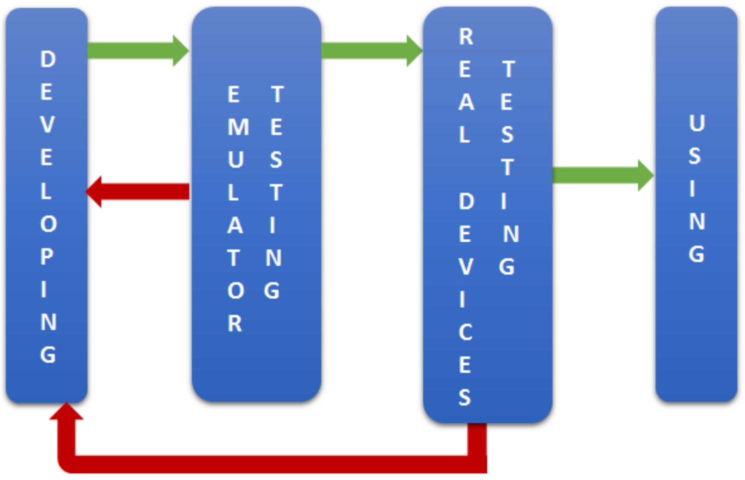


Figure 1. Stages of testing mobile applications

If testing through emulators identifies problems in the application, it is referred back to development module in order to solve all known errors.

When testing the application in the emulator is a success, real environment testing starts, with mobile devices for which the software was designed. If problems are found, while running the application, than the application is referred back to the development module to fix all errors. After malfunctions are solved, the testing circuit resumes by emulator testing and then testing in the real environment. The application is implemented into production only if it has successfully completed all testing cycles and the real environment testing ended with no errors.

The two testing modules are complementary so that scenarios tested in the emulator should not be tested in the real environment. On real devices, only scenarios that cannot be tested in emulator are tested. The following tests should be conducted by using an emulator:

- application interface and how items are displayed on the mobile device screen; this requires the installation of a specific device that emulates the screen size and resolution;
- working with files and data storage mode using internal memory;
- CPU usage and memory when running the application.

The following tests should be performed in the real environment:

- functionality of the application [9];
- network connectivity or other modules such as wireless connectivity, making phone calls or sending SMS;
- storing data using remote database;
- battery consumption while the application is running.

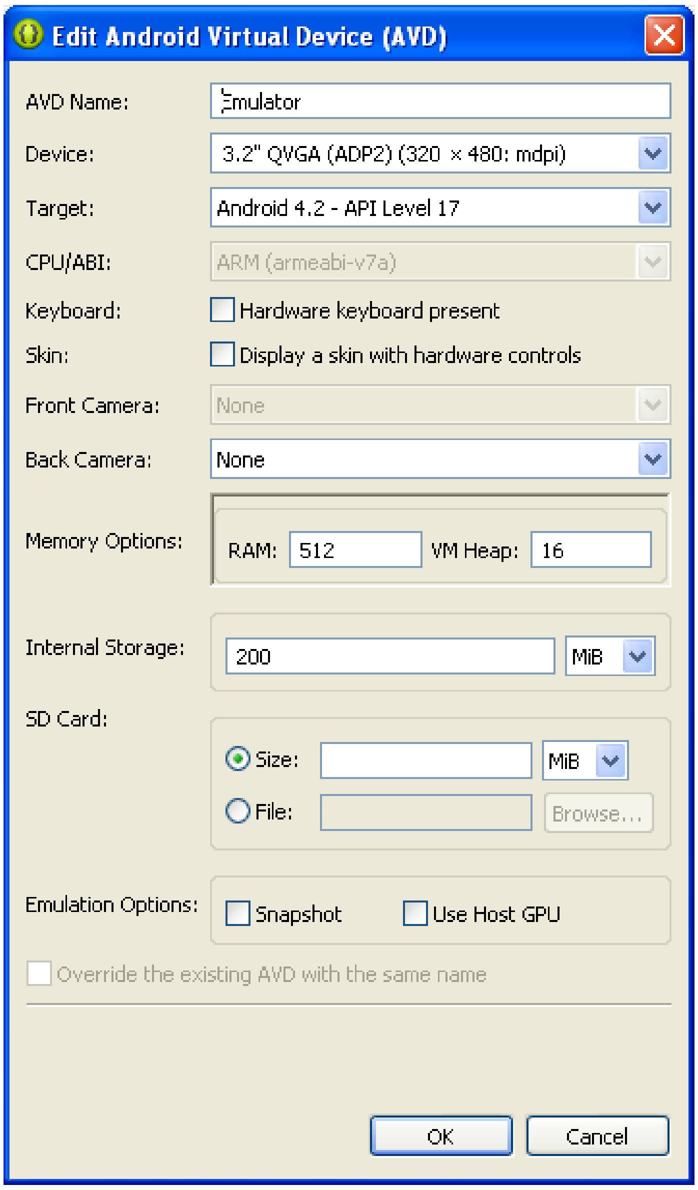
In this way the two test modules cover the entire range of mobile applications testing in a balanced and uniform distribution.

It is necessary to establish certain thresholds in order to decide when application goes to the next stage. Thus defined:

- **threshold P1** – when achieved the application switches from development stage to emulator testing module;
- **threshold P2** - when reached the application switches from emulator testing to testing in the real environment;
- **threshold P3** - when achieved the application is deployed into production.

These tests are carried out by personnel with high qualification in testing mobile applications. Testing should be performed using predefined data sets.

For applications developed for the Android operating system emulator testing is done via emulation provided by the Eclipse programming environment. It can be configured as required for testing. It allows setting parameters such as the type of device, whether or not the keyboard emulator if using the camera as a web camera, the internal memory of the device that is emulated, Figure 2. The developer has the option of Android SDK Manager to install the required SDK version from early versions to the latest version available.


Edit Android Virtual Device (AVD)

AVD Name:	Emulator
Device:	3.2" QVGA (ADP2) (320 × 480: mdpi)
Target:	Android 4.2 - API Level 17
CPU/ABI:	ARM (armeabi-v7a)
Keyboard:	Hardware keyboard present
Skin:	Display a skin with hardware controls
Front Camera:	None
Back Camera:	None
Memory Options:	RAM: 512 VM Heap: 16
Internal Storage:	200 MiB
SD Card:	[✓ Size:] MiB / File: Browse...
Emulation Options:	Snapshot Use Host GPU

Override the existing AVD with the same name

Figure 2. Setting emulator in Eclipse

According to [12] testing in the real environment is made on mobile devices for which the application was designed, and also on other types of devices to in order to observe their behaviour when used outside of their declared range. The description of the application's behaviour when installed on other devices can thus be described for all prospective users. The user is thus informed about what can happen if you install their application, despite warnings that it is not destined for that particular device.

Before the mobile application reaches the above described testing modules it is first tested by programmers during its development stage by using random test data.

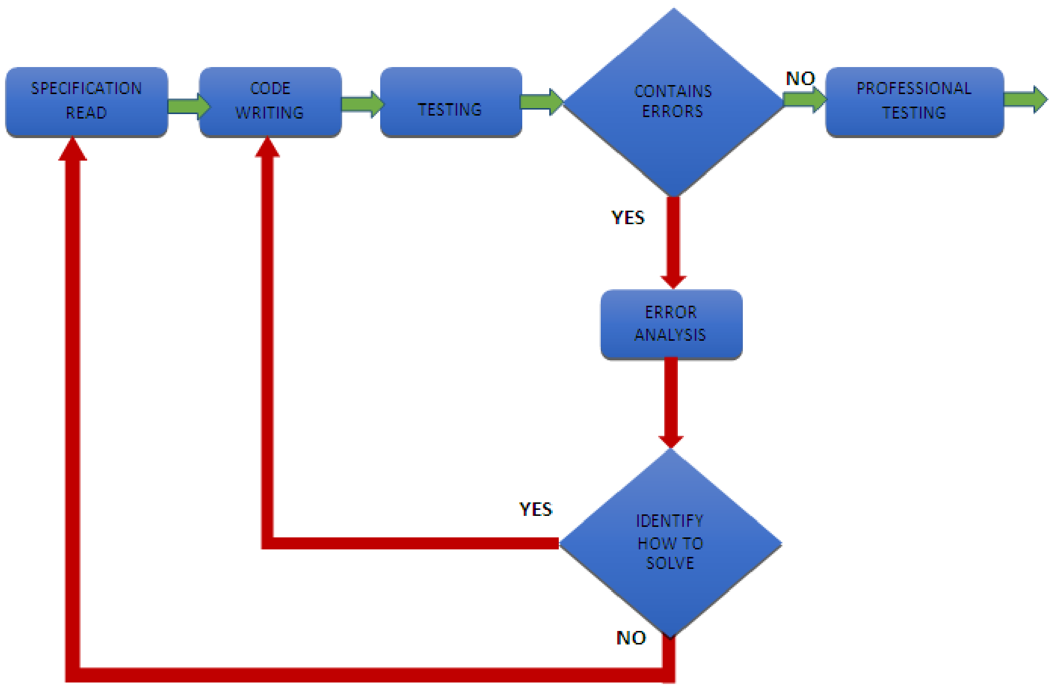


Figure 3. The development with integrated testing

Figure 3 displays the diagram of the compatibility of the application development before it reaches the emulator and the real environment tests. The stringency with which tests are carried out is reflected in the final quality of the mobile application. By reaching thresholds defined, we obtain a quality level QL Application optimal for implementation to end users.

Characteristics of innovative web applications testing

An innovative web application is a software product, available online, whose features and functionality gives the user a high degree of satisfaction by meeting its needs at higher standard. The testing process of innovative applications involves meeting all standards and following all procedures which apply to classical applications but requires a specific set of practices proprietary to this category.

In the process of testing innovative applications the aim is to achieve a balance between terms of the criteria being evaluated. Development of innovative applications that meet all the criteria for compliance, usability, reliability, repeatability, availability and security is unrealistic. The above stated criteria are antagonistic and maximizing one will automatically lead to a decrease in another. Increased security will lead to decreased usability and reliability. Increasing compliance will affect the reliability of the application. Increasing availability will negatively affect the security. A balance must be achieved among the above mentioned parameters. To ensure this balance and allow the release of a high quality application the *Te* indicator is defined.

$$\mathrm{Te}=\frac{\frac{1}{n}\left(\sum_{k=1}^{n}\mathrm{Mc}_k+\sum_{k=1}^{n}\mathrm{Mu}_k\sum_{k=1}^{n}\mathrm{Mf}_k+\sum_{k=1}^{n}\mathrm{Mr}_k+\sum_{k=1}^{n}\mathrm{Md}_k+\sum_{k=1}^{n}\mathrm{Ms}_k\right)}{6} \qquad (1)$$

Where:

n – the number of testers who tested the application; for relevant results requires that $n \geq 3$

Mc – average rating given to the application by testers for the Compliance criteria

Mu – average rating awarded to the application by testers for the Usability criteria

Mf – average rating granted to the application by testers for the Reliability criteria

Mr – average rating given to the application by testers for the Repeatability criteria

Md – average rating awarded to the application by testers for the Availability criteria

Ms - average rating given to the application by testers for the Security criteria

The *Te* indicator ranges in the [1,100] interval, where an application that after the testing process achieves only 1 point for the *Te* indicator is totally inadequate in terms of the analysed criteria and an applications that achieves 100 points for the *Te* indicator is perfect in terms of the analysed criteria. The threshold for which an application is considered suitable to be released into the production environment is obtaining in the testing process achieving an 85 point value for the *Te* indicator. In order to illustrate how to properly use the *Te* indicator, the Alpha application will be submitted for testing. The Alpha application was actually tested using the *Te* indicator and is currently operational. The data obtained in the first testing cycle is illustrated in Table 2. Name and details that could lead to identification of the application have been anonymized. It is recommended to use the model shown in Table 2 for data recording.

Table 2. **Data collection process required for *Te* the indicator**

	Compliance	**Usability**	**Reliability**	**Repeatability**	**Availability**	**Security**	**Observations**
T1	82	98	92	95	67	80	The application cannot be accessed from mobile devices
T2	51	77	88	82	80	90	Does not allow event deletion
T 3	93	82	89	76	82	91	-
T4	86	53	76	89	87	60	The registration form is difficult to find
T5	94	91	82	95	60	82	The application cannot be accessed from mobile devices
	81,2	80,2	85,4	87,5	75,2	80,6	**82,6**

According to data recorded in Table 2, the *Te* indicator’s value for Alpha applications is 82,6. The value is below the 85 point threshold responsible for determining whether an application is ready for release into the production environment. The application should be improved in accordance with the testers’ observations and reported bugs. After necessary adjustments and changes will be made, the Alpha application will be tested again using as a benchmark and tool the *Te* indicator. After performing three independent test cycles and improving the application with each iteration, the 85 point threshold is not reached than lunching should be abandoned.

Development of innovative software applications involves a high degree of uncertainty. Innovation is characterized by novelty and innovative software applications often come with original functionalities or exploit ideas that have not been implemented so far in the online environment. This scenario creates difficulties for test team because traditional test scenarios and tools no longer apply. In innovative software applications the testing team should expect new challenges because specifications are elusive and the image of the final product is often unclear.

The innovativeness attribute of a software application is largely a matter of subjective assessment. The users are the ones who ultimately determine whether or not an application is innovative. It is important to consider the fact that different people perceive the same software application as having different levels of quality [5]. This aspect makes the testing process for innovative applications to include intuitive methods in addition to other industry established methods.

Negative testing is the process of verifying how the application behaves in scenarios it was designed for. Given that innovative applications introduce a significant degree of novelty, users tend to experiment with new ways of using the software. This scenario may lead to using the application for a purpose that it was not designed for. Testing team must ensure that in such cases the application responds appropriately: either by displaying custom errors or by restricting access to certain functionalities, modules and sections.

Collaborative testing is a defining part of innovative software applications. In traditional software development projects, testing occurs after the development has ceased and the interaction between developers and testers is minimal. In the process of testing innovative projects testing and development are tightly linked and run in parallel. Innovative software testing involves effective collaboration between developers and testers. In the Agile development methodology, the testing process is integrated into the development process [2].

User performed testing also known as acceptance testing is a very important component in innovative projects. In innovative applications the user is represented by the project owner. In innovative applications, the project owner is involved in the testing process over the course of the entire application development stage [3].

General testing is essential in innovative software solutions it creates prerequisites for various test scenarios to take place. General testing should not be confused with overall testing. General testing implies that in innovative projects all parties act as testers. The programmers, project manager, graphic designer and owner of the project are responsible for developing and covering various test scenarios. In innovative software development projects each member of the project team is directly responsible for testing the functionalities he has implemented. This does not mean that no one else will test that functionality.

In innovative software applications the testing process is not focused on identifying defects but instead is directed towards generating improvement proposals. The testing process in innovative applications is not performed exclusively at the end of the development phase but rather throughout the entire programming cycle.

Critical to the success of the testing stage is the fact that the infrastructure on which the applications is being tested must be identical to the infrastructure on which the application will run after the official launch [4].

Particularities for implementing structured entities

Structured entities are entities with common properties which allow their representation using a structure. The properties are conditions for membership to the structure and each entity has a distinct form verified by measuring the orthogonality level. Formally, the structures are sets whose member elements meet all conditions of membership. The structures of entities are homogeneous for sets of entities of the same type and inhomogeneous for sets of entities of different types.

Computer applications meant for implementing structured entities are web applications providing modules for: building, populating and generating structures, users' interaction with the structure, processing, centralizing and interpreting the results, interaction's results displaying. There are two user categories: personnel who administrate the structure and personnel who interact with the structure. The names of the modules and their codification are presented in Table 3.

Table 3. Module names/Codification

Module	**Codification**
Building	*MC*
Populating and generating	*MPG*
Interaction	*MI*
Processing, centralizing and interpreting the results	*MPCI*
Results displaying	*MD*

The modules are linked together and the processing done at the level of one module generates input data for the next module. Each user category has access to specific modules as specified in Table 4.

Table 4. Application's modules and users access

User\Access	***MC***	***MPG***	***MI***	***MPCI***	***MD***
Administrator	yes	yes	no	yes	yes
Interaction User	no	no	yes	no	yes

Testing is done by building test cases for every module, applied using samples of users from the corresponding category, according to Table 4. The process implies controlling the normal functioning and it is accomplished at the application level as well as the module level. The activity of usability testing implies monitoring the extent to which the given results are the expected ones and the time stamp necessary for obtaining them is subject to the specified limits [14].

The user interface handles input data and provides the necessary functionalities for accessing the features of each module. For web applications user interface is displayed through web browsers thus testing is done by running the application in different browsers verifying the correctness of content displaying and checking the functionality of input/output operations. In order to do this the set of test cases is built as well as the test data sets and they are applied either by automated testing using specialized computer programs for simulating user's actions or by using real users. [15].

Correctness of data is important for obtaining the expected results because the output obtained by processing done at one module level represent the input for the next module. Figure 4 illustrates the data flow and the links between the modules.

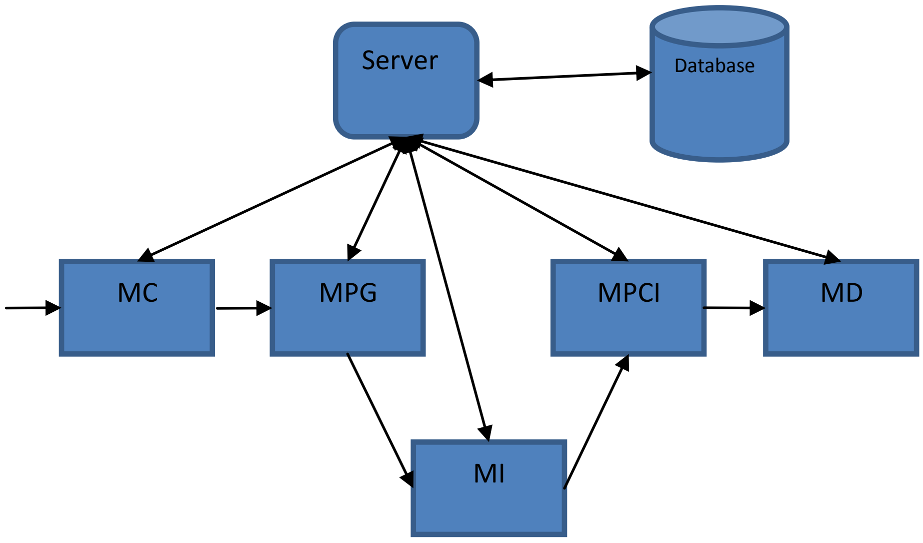


Figure 4. Data flow and links between modules

Graphical user interface's test scenarios are built for each module and depend on the set of operations provided by the module [13]. Coverage ratio increases with the number and diversity of the test scenarios. Distinct test data sets are built for each test scenario. They include both valid and invalid input data in order to test the validation mechanisms. Each set of test data has a corresponding result. For interface evaluation, results for all test data are compared with the results in application's specifications and the level of conformity is determined [16].

An important factor to measure during testing is response time. Because the applications for structured entities automation are web applications, in order to calculate the response time, the following are considered: the time required for input validation, the time required for sending input data from client to server, the time required for processing on the server, the time required for querying the database, the time required to send the response to the client. The time required for processing and validating depends on the configuration of the system the application is running on. Thus, the total response time coefficient *CT* indicator is given by:

$$CT = \frac{\sum_{i=1}^{NM} (\mathrm{TPC}_i + \mathrm{TCS}_i + \mathrm{TPS}_i + \mathrm{TBD}_i + \mathrm{TSC}_i)}{CTS_i} \qquad (2)$$

Where:

CT – application's total response time coefficient;

NM – total number of application's modules;

TPCi – the time required for processing on client for the i module;

TCSi – the time required for sending data from client to server for i module;

TPSi – the time required for processing on server for the i module;

TBDi – the time required for querying the database by the server for the operations corresponding to i module;

TSCi – the time required for sending the response from server to client;

CTS – application's total standard response time coefficient, calculated for different system configurations.

The response time is considered for evaluating application's usability level and it is compared to the standard values. The *CT* indicator takes values in the [0,1] range and for values under 0,8 the application is to be resent to the developing team.

Testing process for collaborative security components

Considering collaborative security we distinguish two categories:

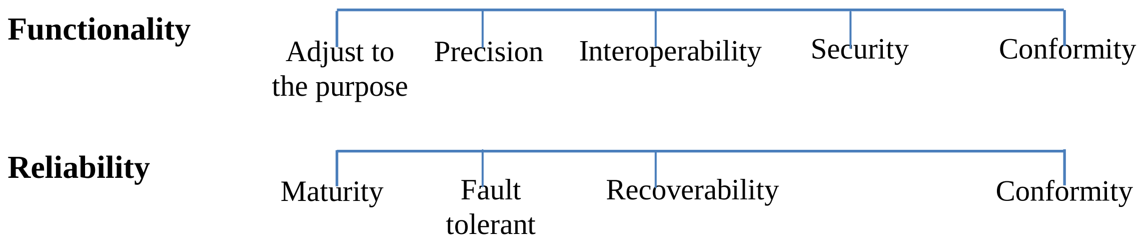
- **Automated collaborative security systems** – collaboration process takes place independent of human factor between security active components, like Firewall, Intrusion Prevention Systems by keeping a continuous communication that allows adjusting the configurations in dynamic and automated manner.
- **User dependent collaborative security systems** – these systems are configured as a consequence of human factor action; the security system must implement communication mechanisms that allow collaboration between users, so that the final configurations will represent the result of the collaboration between multiple specialists in information security.

Regarding the testing process for collaborative security components, when testing, the quality characteristics that are being targeted must be known. According to the ISO 9126-1 standard for software quality, the characteristics in Table 5 are considered to be relevant.

Table 5. Software quality characteristics according to the ISO9126 quality model [17]

Characteristic	Characteristic Description
Functionality	Ability for the software components to provide the needed functions that are needed for maintaining an appropriate level of security.
Reliability	Ability for the software components to provide a high level of trust in identifying and managing the threats and also to provide redundant communication mechanisms.
Usability	Ability for the software component to be easy to understand, easy to use, attractive for the user and to facilitate collaboration at operational level.
Efficiency	Ability for the software component to provide an optimal level of performance balancing the resources that are being used in the given conditions.
Maintenance	Ability for the collaborative components to be modified. Modifications include improvements or changes in the working environment, changes for the functional requirements.
Portability	Ability for the collaborative components to be used in heterogeneous environments, ensuring that they are platform independent

Starting from these generic characteristics the sub characteristics are identified according to Figure 5.



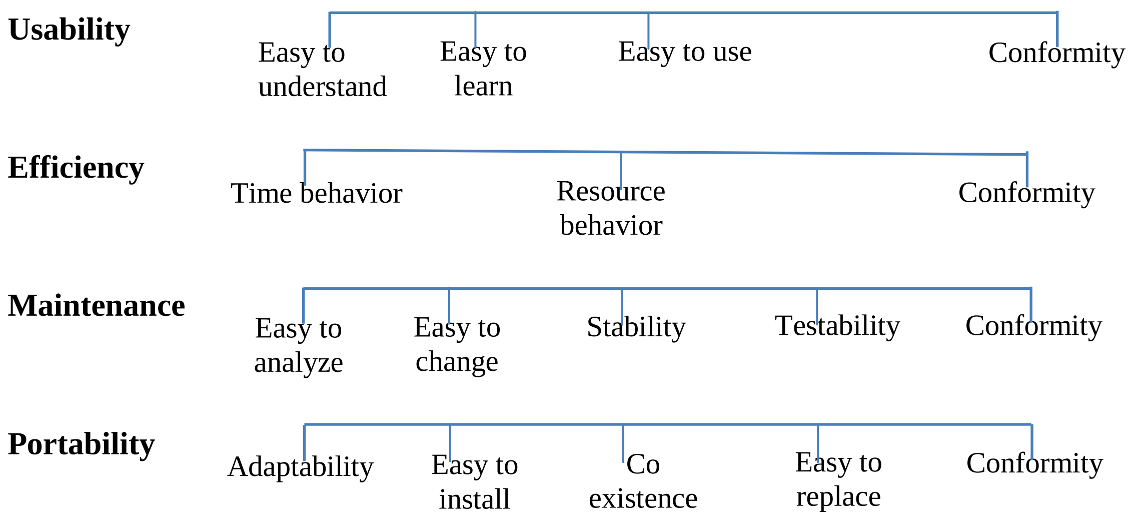


Figure 5. Software quality sub-characteristics according to the ISO 9126 model [17]

The testing procedure considered to be appropriate for verifying and improving the quality of the collaborative security components is Plan – Do – Check – Act adapted for the software quality insurance process, according to Figure 6

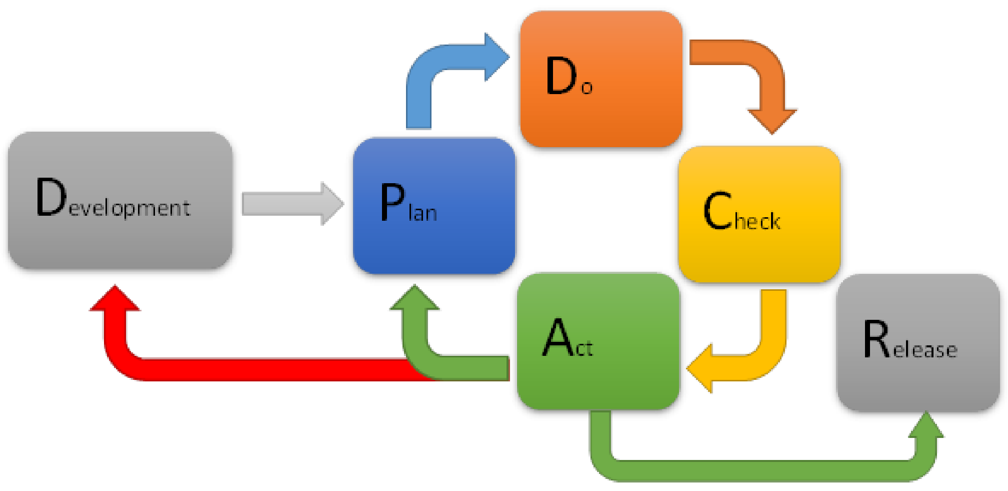


Figure 6. Software quality verification and improvement diagram for collaborative security systems

The planning stage, *Plan* from Figure 6, involves elaborating testing plans considering:

- Identification of the base functionality for the component that needs to be tested and establish acceptance criteria according to the interoperability necessities, precision, conformity
- Pinpointing user interaction components and establishing validation criteria for user input
- Highlighting the critical operability nodes and establish the level of redundancy for those nodes
- Establish a test dataset for which the final result is known, for evaluating component's efficiency
- Identification for the additional components that are interacting with the component that needs to be tested and establish the test cases for verifying an appropriate integration
- Allocation for the resources needed for running the testing process

The *Do* stage, from figure 6, involves:

- Running the testing procedures according to the testing plans
- Executing the testing procedures for ensuring that the base functionality was not affected

Base functionality testing are not necessarily included in the test plans and the purpose for this kind of tests is to ensure a large coverage over the tested functionality. It is recommended to apply *Smoke Testing* and *Stress Testing* no matter what the initial test plans include.

The verification stage, called *Check* in figure 6, involves:

- Verifying the correctness of the way that the tests were made;
- Verifying the results obtained after the tests were completed and establishing which test cases have failed.

The *Act* stage, involves:

- Forwarding the results of the failed test cases in order to have the bugs fixed
- Establish whether the software component it's viable for being released in production

For the collaborative systems, more attention needs to be paid on the communications and to ensure redundancy in cases where the main communication channel becomes unavailable. For these components a Continuous Testing approach is the most indicated. Also, considering that for the collaboration processes, mobility is a huge advantage, a mobile implementation has to be targeted.

Conclusions

The testing process is a critical component of the software development cycle. Testing an application requires not only the identification and reporting of operational errors but also checking the compliance of written code with the original architecture and design specifications. The software testing process is particularly important in ensuring the quality of an application. To ensure the quality of the testing process of a software application, the widest possible range of scenarios has to be covered. To select test tools needed for a specific type of software development project a two-dimensional approach based on method and level is used. Parameters that have to be evaluated in the testing process are: compliance, usability, reliability, repeatability, availability and security. The *Te* indicator is used to determine a balance between the levels of these parameters. Mobile application testing is done using emulators and directly in real environment. In testing both methods should be used as they are complementary and functionalities tested in emulator are not to be tested in the real environment. Testing structured entities is oriented towards evaluating the response times of the application. For determining and assessing the response times of an application the *CT* indicator is recommended. In collaborative security components mainly the following parameters are tested: functionality, reliability, usability, efficiency, maintenance and portability. The research direction that is being outlined in the field of software testing is the automation of the by using software tools to evaluate the correctness of code syntax, effectiveness of implemented algorithms, response times and usability level.

References

[1] G. J. MYERS, C. SANDLER, T. BADGETT - *The Art of Software Testing,* 3rd edition, Publisher: John Wiley & Sons, 2011, pg. 240, ISBN 978-1118031964

[2] D. TALBY, A. KEREN, O. HAZZAN, Y. DUBINSKY - *Agile software testing in a large-scale project*, IEEE Software, vol. 23, no.4, pg. 30-37, 2006, doi: 10.1109/MS.2006.93

[3] M. LINDVALL, D. MUTHIG, A. DAGNINO, C. WALLIN, M. STUPPERICH, D. KIEFER, J. MAY, T. KAHKONEN - *Agile software development in large organizations*, IEE Computer , vol. 37, no. 12, pg. 26-34, 2004, doi: 10.1109/MC.2004.231

[4] M. L. DESPA, A. ZAMFIROIU - *Reasons, Circumstances and Innovative Trends in Mobile Environments*, Informatica Economica, vol. 17, nr. 2, pg 109-118, 2013, ISSN: 1453-1305;

[5] C. KANER - *Exploratory Testing*, Keynote address at the Quality Assurance Institute Worldwide Annual Software Testing Conference, 17 Nov. 2006, Orlando, Florida, U.S.A.

[6] A. MORGAN - *Production Methodologies, Techniques and Guidelines for Modern Computer Game Testing*, 2012, Available at: https://itunes.apple.com/us/book/production-methodologies-techniques/id528733045?mt=11 (requires iBooks 1.3.1 or later and iOS 4.3.3 or later)

[7] I. IVAN, C. BOJA, A. ZAMFIROIU - *Procese de emulare pentru testarea aplicatiilor mobile*, Revista Romàna de Informatică și Automatică, vol. 22, nr. 1, 2012, pg. 5-16, ISSN:1220-1758

[8] http://www.computerblog.ro/software/ericsson-solutie-testare-aplicatiilor-mobile.html, Accesat la 17.08.2013

[9] P. POCATILU - *Testarea functionala a m-aplicatiei Dictionar*, Revista Informatica Economica, nr. 4(28), 2003, pg. 55-58.

[10] C. V. EREMIA, M. TERTIŞCO - *Testarea experimentală a unor algoritmi de compresie a datelor*, Revista Română de Informatică şi Automatică, vol. 20, nr. 1, 2010.

[11] I. IVAN, C. CIUREA, S. VÎNTURIŞ - *Autotestarea aplicaţiilor de scrolling*, Revista Română de Informatică şi Automatică, vol. 21, nr. 1, 2011.

[12] A. KAIKKONEN, T. KALLIO, A. KEKALAINEN, A. KANKAINEN, M. CANKAR - *Usability testing of mobile applications: a comparison between laboratory and field testing*, Journal of usability studies, vol. 1, nr. 1, 2005.

[13] I. IVAN, C. TOMA – Testarea interfetelor om-calculator, Revista Romana de Informatică și Automatică, vol. 13, nr. 2, 2004, pg. 22 – 29, ISSN 1220 – 1758

[14] E. E. HERȚELIU – *Software Quality Management in Developing Citizen Oriented Informatics Applications*, Materialele primei Teleconferințe Internaționale a Tinerilor Cercetători – Crearea Societății Conștiinței, 7-8 Apr 2012, Chişinău, Republica Moldova, Editura ASEM, Chişinău, Republica Moldova, 2012, pg. 55-57, ISBN: 978-9975-75-611-2

[15] S. VILLKOMIR, A. A. KARAHROUDY, N. TABRIZI – *Interface Testing Using a Subgraph Splitting Algorithm: A Case Study*, Proceedings of the 23rd International Conference on Software Engineering & Knowledge Engineering, SEKE 2011, 7-9 July, 2011, Miami Beach, USA, 2011, pg. 219-224, ISSSN: 2325-9000

[16] J. ITKONEN, M. MÄNTYLÄ – *Are test cases needed? Replicated comparison between exploratory and test-case-based software testing*, Empirical Software Engineering Journal, July 2013, ISSN: 1573 -7616

[17] I. PADAYACHEE, P. KOTZE, A. van der MERWE - *ISO 9126 External Systems Quality Characteristics, SubCharacteristics And Domain Specific Criteria For Evaluating E-Learning Systems*, Proceedings of SACLA Conference, 7-9 June 2010, Mountain Lodge, South Africa, ISBN: 978-0-620-47173-2

[18] http://gtmetrix.com/

[19] http://loadimpact.com/