UPPSALA
UNIVERSITET

IT 12 024
Examensarbete 30 hp
Juni 2012

Evaluation of methods for automated testing in large-scale financial systems

Maryna Shtakova

Institutionen för informationsteknologi
Department of Information Technology

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Evaluation of methods for automated testing in large-scale financial systems

Maryna Shtakova

Nowadays automated testing technologies are widely used by companies to make testing of software applications effective and save time and effort applied during a manual testing. Automated testing uses different approaches: from the oldest "record and playback" test method to the modern non-scripting method. All existing testing tools are based on one of these testing methods or their combinations and can be used with varying effectiveness in testing of real software products.
The purpose of this thesis work was to compare testing methods in "real life" – by applying them to the large-scale financial system "Scila Surveillance". To achieve this goal, it was decided to select several testing tools with different testing methods in a basis, evaluate them and select the best one for the automation of functional testing of the graphical user interface in the Scila company. Evaluation was done based on the list of criteria, defined at the beginning of the thesis work and with attention to requirements for a testing tool stated by Scila. Decision matrix method was used to make a final decision about the best testing tool.
It was found that it is complicated to select a testing method which will fulfill all the requirements and expectations completely. The solution taken was to combine a non-scripting testing tool, which includes basic necessary functionality and allows producing test cases quickly with a scripting testing tool, which supports extraordinary test scenarios in case of need.

Handledare: Dennis Persson
Ämnesgranskare: Justin Pearson
Examinator: Anders Berglund
IT 12 024
Tryckt av: Reprocentralen ITC

Acknowledgements

First of all, I would like to thank my supervisor, Dennis Persson, who helped me greatly with his guidance and valuable advices during this thesis work.

Secondly, I would like to express my huge gratitude to Sofia Frisk, Test Manager from Scila AB, whose professional experience in testing and no less important – moral support during the intensive thesis work helped me to achieve my goals.

Also, I would like to thank my reviewer, Justin Pearson for his important comments to this thesis.

Besides, I would like to thank Swedish Institute, who made my studies in Sweden possible with their Visby scholarship. Thank you once again for this not forgettable chance!

Finally, I would like to thank my loved parents and sister, who was always with me during both easy and hard periods of this project.

Contents

1 Introduction

How to automate testing process so that it is effective, testing code is easy and quickly to write and tests are maintainable? Having the right testing method resolves a huge part of problems, which can arise during the test automation. Selection of testing method is not so straightforward task as it can be seen from the beginning. It requires a detailed investigation of available technologies, stating of specific testing requirements actual to this particular application under the test, producing of evaluation strategy and evaluation method, selecting of winning testing approach.

A theoretical description of each testing method gives only estimated impression of their effectiveness and complexity, but real usefulness of them can be discovered only in practise. Even if on the first glance it looks excellent, testing approach can simply lack practical implementation, and therefore it will be useless in a real life. In turn, the average looking testing method can surprisingly be the optimal solution, despite of all disadvantages and weaknesses it may include. So, how to select the best automated testing method for a real large-scale financial system? The thesis work will answer this question in a several steps. It will also use a practical approach, which means that testing methods will be evaluated based on the testing tools that have them in a basis.

1.1 Motivation for research

There are many reasons why it is important to make a good a comparison and evaluation of testing technologies, but the best way to see the motivation for this research is to start thinking about the problem from another side. What are the consequences of randomly selected testing method?

Randomly selected testing method can be:

- not effective
- difficult to use
- take a long time to get used to it
- not applicable to the current software product

As a result of all describes disadvantages, poorly selected testing approach leads to a badly constructed test suite and moreover, it kills a motivation to write tests. One of the bad sides of automated tests is that writing of them after some period of time becomes

a routine monotonic task, and if it is accompanied with complexity of the testing method itself - team will refuse to test the project with a required frequency very quickly. This fact should be especially considered while choosing a testing technology for small teams, where testing tasks handled by developers of a product. These teams have even less time and motivation for testing of their software applications.
Evaluation of testing technologies was selected as a topic for this thesis research to investigate pros and cons of existing testing methods, their application to the real systems and to emphasise that a deep evaluation of testing methods before the work start should be an essential process in all testing teams.

1.2 Thesis aims

This thesis work aims to make evaluation of existing testing methods for large-scale financial systems, based on the application created by Scila AB. To gain the main aim, it is necessary to answer three questions:

How to select testing methods for evaluation? To answer this question, the following steps should be completed:

- To investigate previous work on the problem.
- To list and describe existing automated testing methods.
- To compare testing through the API to a testing through the GUI.
- To finalize list of testing methods for evaluation.

How to select testing tools, which will practically represent testing methods? To answer this question, the following steps should be completed:

- To outline main requirements for the testing tool, stated by Scila.
- To select the initial set of testing tools.
- To make a brief evaluation of them based on their official descriptions and several implemented test cases.
- To remove testing tools which does not fit basic requirements.
- To add new testing tools which better fit basic requirements.
- To finalize and fix the set of testing tools for a detailed evaluation.

How to evaluate testing tools and select the best one? To answer this question, the following steps should be completed:

- To create a list of evaluation criteria based on the requirements stated by Scila and additional characteristics, which will fulfil the testing needs optimally.

- To make a detailed evaluation of testing tools by all criteria.
- To convert evaluation results to brief grades. Evaluation on this step can use different scales.
- To make a pairwise comparison of criteria for each testing tool.
- To calculate weights for criteria.
- To select a common numerical scale for criteria.
- To convert criteria grades to numerical results.
- To calculate a numerical score for each testing tool.
- To select the best one testing tool based on the best result from the previous step.
- To implement set of test cases using the selected testing tool.

1.3 Contribution

The first contribution of this thesis work is a result of research in area of automated testing methods. It was found that among currently existing testing methods, the most powerful and promising is non-scripting method, which is easy and straightforward in its usage and can fulfil the testing needs quite fully and quickly. It is necessary to say, that several limitations of this method were also detected, and the best solution, that can be taken now is to complement non-scripting testing tools with scripting ones, which requires more time to create tests, but which are powerful enough to handle non-standard situations. The found approach can be used by companies in their daily testing tasks.

The second contribution lies in the found fact, that not all existing testing methods are represented properly by currently functioning free testing tools. As for example, behaviour testing tools for Java platform do not have stable supported version yet and record-replay testing tools have limitations in testing of tables.

The second contribution is suggestion of testing method and testing tool for a Scila Surveillance system, evidence of its efficiency in the implementation of test coverage for a Swing GUI based software product.

1.4 Outline of the thesis

This thesis report has eight chapters. Description of every chapter can be seen below. Chapter 1 gives a short introduction to this thesis project, describes motivation for a

research, aims of this thesis and its contribution to the automated testing field of research.

Chapter 2 presents a literature review of previous works in area of test methods for automated testing of software applications.

Chapter 3 outlines theoretical basis for this thesis project. It describes different existing GUI testing methods and also compares advantages and disadvantages of testing through the GUI to a pure API JUnit testing.

Chapter 4 provides description of the evaluation process, which includes requirements definition, selection of testing tool for each testing method, selection of evaluation criteria for testing tools and detailed evaluation itself.

Chapter 5 describes a formalization of results, got in the previous chapter and also includes calculations to simplify selection of the best testing tool, based on the numerical result.

Chapter 6 provides results of evaluation process: testing tool (and corresponding testing method), which gained the best scores and was selected for implementation of test suite.

Chapter 7 presents conclusion of the thesis work, which summarizes results of current research. Moreover, this chapter outlines a future research, which can be done in the GUI testing area.

Chapter 8 gives a discussion about strengths and weaknesses of this work and possible improvements.

Chapter 9 provides a list of references, which were used for this thesis work.

2 Literature review

2.1 Previous research in area of test automation methods

Topic of automated testing methods was appearing before in different scientific papers, books as well as on Internet forums, blogs and companies' slides as one of the most popular topics between test engineers and other quality assurance specialists.

Among others, John Kenth (2009) [17] investigated progress in a test automation of GUI since the early 1990's. He found that one of the problems for testing is maintenance: test suites in huge projects grows quickly and become very difficult to work with. As a result he admits that testing methods should evolve all the time to fit the needs of growing and changeable application under the test. Author also compares several testing approaches: record/replay method, data-driven, scripting methods, their advantages and disadvantages in testing of user interfaces. He finishes with discussion about the test automation frameworks, which he sees as the best way for test automation because frameworks improve tests maintenance, time for development and does not require scripting effort.

Only three years later than Kenth, David Hunt (2012) [18] from the Infuse Consulting, leading provider of testing services, identify already five generations of testing approaches: record and playback, scripting (based one functions reuse), data-driven scripts/functions, action word/ keyword scripts / functions, scriptless. This fact confirms the quick evolution of testing approaches and this list of five test automation generation can be seen as a current list of technologies used for automation of GUI testing

The latest, fifth generation of GUI test automation techniques was studied by Paul Sudipto from Geometric (2011) [19]. Stupido highlights advantages of scriptless testing method: test engineer can create tests visually by constructing them from objects and actions, adding conditions and modifying object properties. The main advantage of scriptless test automation lies in its name – it does not require scripting effort. Stupido underlines that Geometric got increase of productivity on 50% and even more as soon as it started with the scriptless testing tools. Moreover scriptless tools are the same powerful in terms of debugging, running tests, reporting of them, client-server approach and other characteristics.

Jeff Hinz, Martin Gijsen (2012) [20] discuss hybrid approaches, which assume different combinations of previously highlighted methods. They point out that use of hybrid approaches has advantages, like ability to combine technologies by the wish and need of test engineers, as well as disadvantages like complexity of the tool and risks, inherited from the old testing approaches, which are included.

Finally, there is necessary to mention behaviour-driven test automation approach, described by Aslak Hellesoy, Matt Wynne (2012) [21]. This method stands separately because very few testing tools implement it. Behaviour-driven testing of GUI can be suitable for teams which work in test-driven development style, writing tests before the actual code exists, and also for teams who want to include product stakeholders in tests construction process.

2.2 Previous research in a selection of testing tools

Comparison of testing tools can be done in a different ways.
Cordell Vail (2005) [24] makes a comparison of testing tools in a tabled form, where he simply describes advantages and disadvantages of selected testing tools and own opinion about each of them, while Terry Horwath (2007) [23] provides a deep and detailed investigation of each testing tool by eighteen evaluation criteria, starting from the ability to recognise GUI objects, looking through the all features of testing tool and finishing with the audience, which is a assumed as a tool users.

At the same time E.Gomonova, I. Anisimov, I. Petrov, D. Kondtratiev, D.Zernov, O. Moroz from Luxoft (2007) [25] provides own comparison of testing tools using a simple scale “poor, good, excellent and by core criteria only: functionality, ease of learning and tests development, documentation and support quality, price”.

T.Illes, A.Herrmann, B. Paech, J. Rückert (2005) [22] add matrix to a testing tools evaluation, where they state a score for each tool using scale from 1 to 3. This allows to make a comparison of tools in a more formal way and to calculate the total result. Numerical results are easier to percept than descriptions made in words.

An interesting suggestion was proposed by Ray Robinson (2001) [26] – compare tools by quality criteria using ISO/IEC 9126 standard: functionality, reliability, usability, efficiency and others.

3 Theoretical basis

3.1 GUI testing methods

Nowadays there are several different methods for automated testing of software applications with graphical user interface. Each method has its advantages and disadvantages. Main testing methods are listed below:

1. **Record and Playback method**

   Pros [16]

   - Very quick to produce scripts. In the Record and Playback method scripts are recorded automatically with a minimal human effort. It requires performing of some set of actions on the user interface, which are recorded and can be reproduces as many times as necessary.
   - Easy to understand. Record and Replay method is quite straightforward and can be understood even by non-technical person.
   - Very quick to maintain. Tests can be removed and recorded again very quickly.

   Cons

   - Limited in functionality. This method often faces difficulties with recording of custom objects; it is mostly oriented on the standard GUI objects recording.
   - Encourages script proliferation. Number of recorded tests can increase significantly in a short period of time and it can be very difficult to organize them without redundancy.
   - Often very fragile. Even small change to the GUI layout can cause crash of many test cases, which depend on the particular GUI elements positions, names, or other characteristics.

2. **Scripting method** [16]

   Pros

   - Powerful. Scripting method can handle almost all possible test scenarios with custom GUI objects included, because scripting method is based on some programming language, which expands the borders of possible implementations.
   - Encourages reuse. Scripting method is based on the objects, procedures and functions, which are reused in different test cases.

- Can be very robust. Test cases, created using scripting method, can be very stable, because of powerful programming methods used.

Cons

- Complicated. Scripting method complexity increases with increase of programming language complexity in its basis. It usually requires technical knowledge and cannot be used by tester without programming skills.
- Parallel development effort. Creation of scripted test cases can be seen as a second project in addition to a project, which is under the testing. Sometimes development of test cases takes even more time than development of software product itself. This puts a double effort on the involved team.
- Not suitable for all testers. Scripting method assumes obligatory knowledge of programming language, which it uses and cannot be used by testers without programming background.

3. **Behaviour method**

Pros

- Test scenarios. Behaviour testing method assumes writing of test scenarios, which describes some set of actions with a given background and aimed result. Test scenarios can also act like documentation to a software product and are easy to read and understand.
- Aims to include stakeholders in product testing. Writing of tests scenarios can be done by product owner and product users, which guarantees that most import product usage scenarios will be included and described without errors.
- Tests can be written before the code exists. This means that code, written after the tests will fulfil requirements strictly and will not be redundant.
- Easy to understand. Behaviour testing scenarios are often written in a simple language, close to the human language, which makes its understanding natural.

Cons

- Double effort. In practise, it is quite difficult to get the test scenarios from the product owner in that form, which is acceptable for their future implementation. So, scenarios are modified and rewritten by test engineers and developers teams that increase their effort level.
- Too exotic. Behaviour testing is not widely used approach and some test engineers can be simply not familiar with this method.
- Limited number of testing tools. As behaviour testing is not widely used approach, there is a short list of testing tools, which implement this approach.

4. **Non-scripting method**

Pros

- Easy to use. Non-scripting testing method assumes that tester does not see a product code and does not interact with it. This method usually uses predefined objects to build a test case. Objects represent GUI elements and simple actions. Tester is only required to order the objects and actions and to fill their properties for a particular test case. As predefined objects do not require time to build them, time and effort for test construction reduce significantly.
- Can be used by non-technicians. As scripting method does not involve tests programming, it can be used by all testers: with development background and without it.
- Test cases can be structured easily. Test cases can be structured by reordering of their parts. Also test cases can be reused partly or entirely in other test cases.
- The last generation technology. Non-scripting testing method belongs to the last generation of test methods, which aims to minimize time and effort for test creation while increase effectiveness.

Cons

- Limited in functionality. Non-scripting methods are limited in functionality if to say about the custom GUI objects. They allow writing of extensions to overcome this problem, but these extensions can be quite complicated and require knowledge of programming languages as well as investigation of testing tool code.

3.2 Evaluation of testing through the GUI in comparison to a pure API JUnit testing

JUnit API testing:

Advantages

- Relatively low complexity. API testing is based on the well-developed functions, procedures and types, so its complexity is lower than GUI testing, where for example, custom components increase the effort applied to a testing significantly.

- Stability of test code. Tests remain stable regardless the changes made to the code. They do not fail unexpectedly; failed API test usually means that underlying application's code was significantly changed.
- Focused on testing of business logic of application. API testing is based on the check of internal functionality of application under test.
- Reduced cost of testing in total. API testing reduces the cost of manual testing of application.
- Low maintenance costs. API tests require minor changes in the majority of situations, huge changes to the tests should be done only if significant changes to the big part of underlying code were done.

Disadvantages

- Does not detect errors which are displayed on GUI layer. API testing check the functionality of application, but as GUI layer is not involved in API testing, errors on this layer cannot be found during the API testing process.
- Testers are required to be familiar with programming. As API testing is white-box testing, testers should understand the code of application under test. Bad understanding of the code can finish in the not effective or not full API testsuite.
- Test scenarios, which are unreal in practise, while using the application through the GUI, can be implemented. Unnecessary tests can introduce wrong dependences in testsuite as well as increase a total testing time of application.

Testing through the GUI:

Advantages

- Focused on testing of functionality visible to end-user. Information, which is displayed to end-user, can differ from the information taken from the database because of some bug in the code. API testing will not find it, as it tests information taken from the database, but GUI testing will do, as it tests exactly that what can be seen on the screen by anyone.
- Detect errors, which are actually displayed on the GUI layer. Testing through the GUI allows finding problems, which can be not straightforward, while testing the API but can occur on the GUI layer.
- Reduced cost of testing. GUI testing reduces the cost of manual testing of application.
- Testers can be people without programming background. GUI testing benefits in most cases if it is done in a black-box way. This allows looking on the GUI

testing from the position of application's end-user. So, testers without the programming background can participate in GUI testing process.

Disadvantages

- Relatively high complexity. GUI testing often faces problems like recognising of custom GUI components, reusing of GUI components (for example, reusing of main application window to eliminate a long and costly login procedure for every test).
- Low stability of test code. GUI testing is affected by necessity to wait for components to be displayed or available. Waiting time usually varies every test execution. Long waiting time waste the testing resources and increase the total testing time, while too short testing time makes testsuite unstable - tests fail just because some GUI component was not displayed in the time limit.
- Not possible to start with testing on the early stage. GUI testing is dependable from the physical exist of graphical user interface. It also should not only exist but work, to make possible test actions go through the test paths.
- High maintenance costs. GUI tests reuse the same elements of interface, so the small change to one element can lead to fail of significant part of test suite. Maintenance of broken tests takes a time and a lot of effort.

Keeping in mind comparison, which is made above, it is easy to conclude that there is not possible to choose the best testing method in favour of testing though the GUI or testing through the API. Both of them plays important role and should be used in combination to gain the best level of test coverage.

Scila application includes twelve packages with integration API tests, written using the JUnit testing library. They verify functionality of application, mainly comparing data, which was selected from the database with expected etalon. This promise correctness of information requested from the database, but does not checking the actual displaying of it on the Scila Surveillance interface. Verification of tables on the GUI is necessary to be done via the GUI tesing tool. Combination of two testing strategies will provide a correct and the fullest testing of Scila application.

4 Evaluation process

4.1 Requirements definition

At the beginning of the thesis work list of requirements to a Swing testing tool was stated by Scila:

Obligatory characteristics of a testing tool:

1. Compatible with Scila Surveillance application.
   GUI testing tool must be compatible with a Scila application as testing of it is the Scila's core goal.
2. Open-source tool.
   Scila system is completely written in Java using industry-standard open source frameworks. Therefore testing tool must be also open-source.
3. Able to test the Swing GUI components with a focus on the testing of tables.
   Scila system is a financial application with data, stored and presented in a table form.
4. Test cases can be created quickly and easily, should be maintainable.
   Scila has a small development team; tests must be easily created and managed by the current team.
5. Effective debugging provided.
   It must be easy to identify tests failures and their reasons for effective tests creation.

Preferable characteristics of a testing tool:

1. Includes visual reproducing of test fall.
   It is preferably to have visual reproducing of tests execution process to have a possibility see what is going on.
2. Easy configurable.
   It is preferably to have quick and simple installation and configuration procedure of a testing tool.

4.2 Evaluation criteria definition

Evaluation criteria were defined based on the list of initial requirements, stated by Scila at the very beginning. The main aim was to cover all the obligatory requirements for a desirable Swing testing tool. Furthermore, criteria, which will cover preferable requirements, were added as well as criteria which can be useful for the testing of Swing GUI based applications.

Below is a list of evaluation criteria for Swing GUI testing tools:

1. Configuration
    - supported ways
    - complexity
2. Use
    - complexity of tool
    - time to get familiar with tool
3. Swing GUI Object Recognition
    - recognition of standard Swing objects
    - recognition of custom objects
4. Creation method:
    - visual recording
    - programming
    - constructing from ready objects
5. Playback
6. Debugging Support
    - debugging tools provided
    - effectiveness
7. Recovery System
    - failure screenshots
    - failure messages
    - visual indication
8. Documentation
    - introduction guide
    - Javadoc API specification
    - examples provided
    - user guide
9. Technical Support

- supported forum
- feedback from development team

10. Reports
11. User Audience
    - developers
    - test engineers
    - both
12. Weaknesses
13. Extension options

4.3 List and general overview of testing tools

It was decided to choose testing tools, which use different testing methods. This approach will give a possibility to compare existing testing methods in practical use. Initially, three testing tools were selected for evaluation:

1. **Window Tester Pro** - testing tool, which uses record/replay testing method.
   Advantages of Window Tester pro, stated by its developers [7]:

- Eliminates need to create tests manually. This increases a productivity of testing process dramatically.
- Tests generated by Window Tester Pro are pure JUnit tests. Tests have all the advantages of JUnit (for example, they can be automated using Ant or executed within the development environment like Eclipse).
- Tests are created in a record/replay way. This characteristics means that tests can be written quickly, without significant effort.

2. **UISpec4J** - testing tool, which uses scripting approach.
   UISpec4J development team describes its main goals in testing of Swing GUI based applications like [3]:

- Ease the testing of Swing based GUIs. This is reached through the fact, that actual Swing components are wrapped in a set of UISpec4J wrappers. They hide the complexity of Swing components and allow writing simple, easily readable functional tests.
- Provide a code-oriented tool. UISpec4J is a Java library, provided as a .jar file. Tests are created as a peace of code in a programming way.

- Promote the use of functional testing as advocated by extreme programming. UISpec4J is developed to be simple, to provide a possibility to write functional tests as a specifications for a project.

The advantages, highlighted by UISpec4J developers are [4]:

- API designed for readability. Created tests are human-readable, complicated constructions are avoided and data-structures are made to reproduce the actual structure of the components (for example, contents of a table is represented by a two-dimensional array).
- Smart component search. GUI components can be found without the name specified.
- Rely on displayed information only. UISpec4J library finds component by their displayed labels, which simulate the human way for finding objects.
- Invisible execution only. GUI does not require to be displayed during the process of tests execution.
- No modifications needed in production code. Modifications to the existing product's code are not required to be able to test it.
- Extensibility. UISpec4J library can be extended with wrappers for custom Swing components.

3. **Cucumber** - testing tool, which uses behaviour driven testing method.

Advantages of Cucumber testing tool, highlighted by its developers [8]:

- Allows creating of automated tests in a plain text. Plain text descriptions works as documentation to the projects, as well as specifications to the functional tests and future development.
- It has been translated to more than 40 languages. Tests can be written on the native language, which generally simplify the tests construction process.
- Supports behaviour driven development. Tests can be written before the code exists.

After the basic evaluation of shortlisted tools, was found that Window Tester Pro tool has limited resources for testing Swing components. While having nine packages for testing SWT based applications, it includes only three packages with classes and interfaces, able to test Swing components. Classes, which are developed specifically for testing Swing JTable components, are limited to a single class - JTableItemLocator, which does not provide enough functions to test tables. As testing of tables is critical characteristic for Scila Surveillance application, it was decided to skip Window Tester Pro tool from the list for the further evaluation process.

There was found unexpected problem with Cucumber testing tool as well. Cucumber does not have official supported version for a Java VM currently [5]. The previously supported version Cuke4Duke was considered by development team as a hard to install, slow, hard to use and test and unstable. It is no longer maintained. Nowadays developing version is a Cucumber-JVM, which is in the process of continuous changes. Though, there is a possibility to build the Cucumber-JVM from the provided sources, it was considered as a not acceptable solution. Therefore, as previous version is not supported, system under development is not stable and officially released; it was decided to not follow with the Cucumber testing tool.

Two additional testing tools were chosen to replace the Window Tester Pro and Cucumber, to have options for tools evaluation. The best seemed solution was to search through the most popular tools used for Swing GUI testing purposes. “Google Insights for Search” [6] provides a possibility to compare volume of search requests for patterns. The diagram below [1] displays the most searched Swing testing tools for the 2011 year. IBM Rational Functional Tester (FT) was found to be a commercial testing tool, which is not allowed due to the initial requirements, stated by Scila. TOSCA is rather rear used. Therefore, the remaining two testing tools were added to the list: FEST (its popularity is relatively stable according to the graph) and Jubula (it has raises and fall down in popularity, but it significantly differs from other tools, what make it possibly a promising tool).

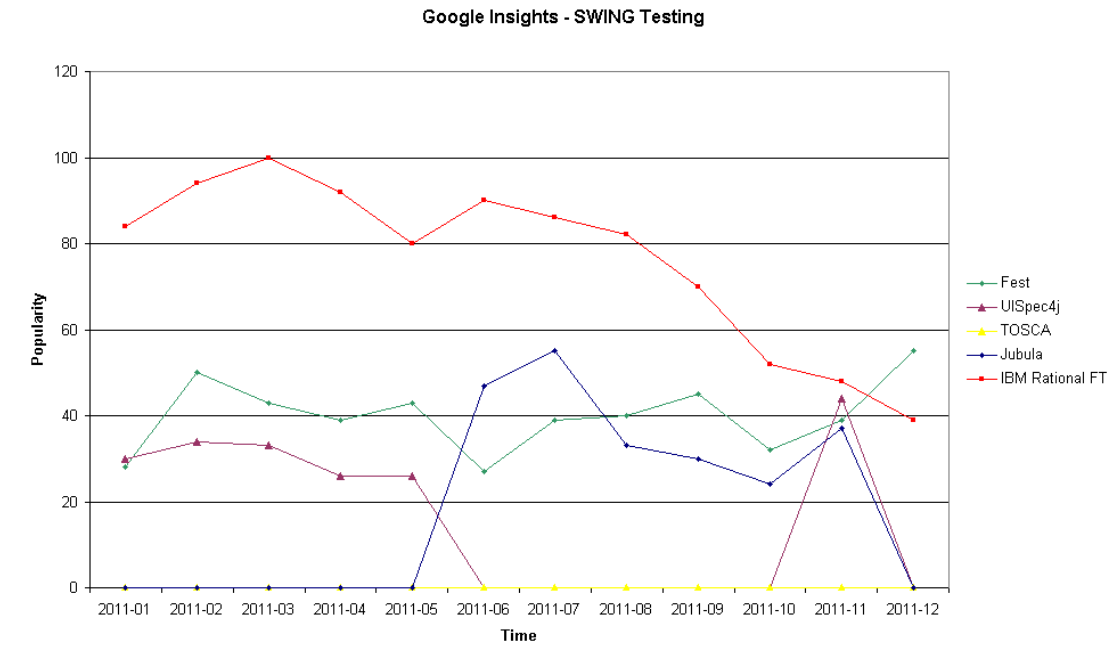

Figure 4.1 - Popularity of SWING testing tools [1]

1. **Jubula** - testing tool, which uses scriptless approach + record/playback testing method.

   Advantages of Jubula testing tool, highlighted by its developers [10]:

- Treats the software as a black-box. This helps to create tests quickly, because process is not based on programming skills. Tests can be written by testers from the user-perspective rather than from developer.
- Projects are stored in multi-user database. Saved data is well-organized and formatted.
- Automatic screenshot on failure. The cause of failure can be easily found based on the screenshot taken at the moment of the failure occurrence.
- Support for redundancy removal. Tests are constructed from the small predefined blocks, which can be reused in multiple different tests to eliminate redundancy and achieve a good modularity.

2. **FEST (Fixtures for Easy Software Testing)** - testing tool, which uses scripting approach.

   FEST team describes its main advantages as [9]:

- Provides a simple and intuitive API for functional testing of Swing user interfaces. Tests created with FEST are easily readable.
- Tests written using FEST-Swing are robust. Components of GUI can be found in multiple ways, search is reliable, which ensure tests stability in case of some changes to GUI was introduced. Tests reproduce actual user's actions, which ensure their correct behaviour.
- Makes troubleshooting failures a lot easier. It saves screenshots of a screen in case of tests fail, which helps significantly to investigate the source of problem, forced tests to fail. It provides nicely formatted failure messages.
- Provides a fluent interface for assertions. Assertions are good readable, formatted close to the plain English text, which allows understanding their sense even to non-technicians.
- Provides an intuitive, compact and type-safe fluent API for Java reflection. Java reflections become easily to use.
- Provides mock template. Expectations and real tests can be clearly separated from each other.

Finally, the modified list of Swing testing tools, which were chosen for further evaluation, is stated below:

1. UISpec4J – uses scripting method.
2. Jubula – uses scriptless method + record/playback
3. FEST – uses scripting method.

4.4 Evaluation of testing tools

Several tests scenarios were developed to check the tools' advantages and disadvantages in testing of Scila Surveillance application. Tests scenarios aimed to cover all Swing objects, used in Scila application interface. The same tests were implemented using all three testing tools to have equal base for their evaluation. Implemented tests were connected in a testsuite, to ensure their fully automatical execution.
Evaluation of each tool, based on the previously selected evaluation criteria, is presented below.

1. **Configuration**

- **supported ways**

UISpec4J	There are 2 possible ways to configure UISpec4 with the project under the test: ● include uispec4j-xxx.jar to a project's classpath ● get it from the Maven repository using Maven dependencies
Jubula	There are 5 different distributions of Jubula, including standalone application and Jubula plugin for Eclipse.
FEST	There are 2 possible ways to configure FEST with the project under the test: ● include several .jar files (can be found in fest-assert-1.4.zip, fest-swing-1.2.zip, fest-reflect-1.2.zip, fest-mocks-1.0.zip) to a project's classpath ● get it from the Maven repository using Maven dependencies

- **complexity**

UISpec4J	Complexity of configuration of UISpec4J with Scila application was low. It was configured using Maven dependencies.
Jubula	Complexity of configuration of Jubula with Scila application was relatively high. Jubula Eclipse plugin was used with Scila application. After

	installation, plugin requires to assign properties, which are specific for Scila application start (Virtual Machine variables, path to the executable file of application under test (AUT), AUT name, host and id). Assigning of these properties was not always straightforward and clear task.
FEST	Complexity of configuration of FEST with Scila application was intermidiate. Despite of FEST developers advise to add only 4 jar files from the download page of FEST, there was necessary to attach additional jar files, for utils module and JUnit module (fest-util-1.1.6.jar, fest-swing-junit-1.2.jar) to get FEST working.

2. Use

- **complexity of tool**

UISpec4J	At first glance, tool looks simple, as it is based on JUnit and the structure of code is familiar. In reality, it took a long time, around several weeks, to create a testsuite for the Scila application. The main appeared problems were reuse of application window through the different test (login to the Scila application is a long process and cannot be done every test, because it is not efficient), as well as reuse of components through the tests within one class and between different classes. Reuse of variables violated the JUnit principles in this case, but it was the only one effective way found to implement the effective testsuite. One of the advantages of UISpec4J is very intuitive call of functions. All methods calls start with ".get*", all variables can be set with ".set*" (for example, window.getButton(), button.setText()). This characteristic helps a lot during the code creation.
Jubula	Tool is simple in use. The initial difficulty is to understand how to use different windows of Jubula, which have different purposes as well as to study how to form optimal reusable blocks and build tests based on these blocks. Nevertheless, usage of Jubula is significantly simpler than usage of two other tools, as the test is built from the ready blocks and time for their hand-scripting is saved. It took 3 days to create testsuite using Jubula testing tool.
FEST	Tool is similar in usage to UISpec4J that is why complexity, advantages

	and disadvantages are similar too. It is the same difficult to reuse application window and components variables.
	Main advantage of FEST is the biggest number of built-in libraries and classes. That is why it is the most powerful tool among the three. It has libraries for drag-and-drop functionality, reproducing of mouse movements and others. However, these libraries are not used in Scila application, so their pluses are discussible.
	Disadvantage of FEST is its call of functions, which is less intuitive (in comparison to UISpec4J). The usual call is : window.button().
	Also disadvantage is the way in which FEST implements a search of component.
	For example:
	`frame.button(withName("cancel").andText("Cancel")).click();`
	Use of "`andText`" requires typing by hands of static import:
	`import static org.fest.swing.core.matcher.JButtonMatcher.withText;`

- **time to get familiar with tool**

UISpec4J	It took quite a long time to get familiar with a tool, 2 weeks. This time includes time for understanding how to use the tool, investigating its main libraries and creating set of 3 tests within the testsuite.
Jubula	It took 3 days to get familiar with a tool including the time it took to create 3 tests within the testsuite.
FEST	It took around 1 week to get familiar with a tool, its libraries, and to create tests. Time could be longer, but as FEST has many similarities with a UISpec4J, knowledge, received previously, helped to get familiar with FEST quicker.

3. Swing GUI Objects Recognition

- **recognition of standard Swing objects**

UISpec4J	Recognises all standard Swing components.
Jubula	Recognises all standard Swing components.

FEST	Recognises all standard Swing components.

- **recognition of custom objects**

UISpec4J	Requires extension, written by tester, to be able to recognise custom objects.
Jubula	Requires extension, written by tester, to be able to recognise custom objects.
FEST	Requires extension, written by tester, to be able to recognise custom objects.

4. Creation method:

UISpec4J	Programming. Code is similar to JUnit test, but it is extended with UISpec4J functions, used specifically for Swing elements.
Jubula	It has 2 possible methods: • constructing from ready objects Test is constructed in drag-and-drop mood from the predefined objects and actions. Tester just needs to assign them properties, which are specific for the application under test. • visual recording Visual recording is used to record user manipulations, but it creates objects, which are usually redundant.
FEST	Programming. Code is similar to JUnit test, but it is extended with FEST functions, used specifically for Swing elements.

5. Playback

UISpec4J	Playback in UISpec4J tool is an execution of testsuite without the visual reproducing of actions, which are currently happen. “Blind” playback speedup a reproducing of testsuite, which can be guessed as an advantage for the Scila application testing, but blind playback makes debugging of application harder.

Jubula	Visual reproducing during a playback is supported, which helps significantly to understand what is going on with application under test.
FEST	Visual reproducing during a playback is supported, which helps significantly to understand what is going on with application under test.

6. Debugging Support

- **debugging tools provided**

UISpec4J	Test code can be debugged using built-in Eclipse debugging system as well as some debugging feedback can be received through the printing variables using System.out.println Java command.
Jubula	Debugging step-by-step is not supported but test execution can be paused on error, so that error's reason can be analysed.
FEST	Test code can be debugged using built-in Eclipse debugging system as well as some debugging feedback can be received through the printing variables using System.out.println Java command.

- **effectiveness**

UISpec4J	Debugging tools are effective in the identifying of failures reasons.
Jubula	Debugging tools are effective in the identifying of failures reasons.
FEST	Debugging tools are effective in the identifying of failures reasons.

7. Recovery System

- **failure screenshots**

UISpec4J	Failure screenshots are not supported in the UISpec4J testing tool.
Jubula	Failure screenshots are taken automatically when any failure occurs in the testsuite reproducing.
FEST	Failure screenshots can be taken when failure occurs, but this feature should be configured manually with special FEST annotations, written

	before the test methods/ test classes, which should be monitored for failure occurrence. Screenshots of failed tests are saved in a special folder “failed-gui-tests”, placed relatively to the folder where are tests executed.

- **failure messages**

UISpec4J	Failure messages are well-descriptive. Especially helpful for Scila testing were messages which suggest alternatives in case of several Swing object with the same name or type were found.
Jubula	Failure messages are well-descriptive and give opportunity to identify the failure cause easily.
FEST	Despite of FEST team writes that FEST provides good messages on failures, during the Scila application testing; it was found that messages are often very confusing on context. Failures due to the logical errors in a test code, assertion failures, failures appeared because of insufficient timeout for long events should preferably have different well-descriptive messages. They were not found in the FEST testing tool.

- **visual indication**

UISpec4J	Visual indication of failed UISpec4J tests is the same as JUnit tests. Failed test is displayed with a red cross before the test class name and failed testsuite is also marked with red color.
Jubula	Testsuite flow is represented with a tree. Tree nodes are simple actions of tests within the testsuite. Failed tree nodes are marked with a red cross before them. Advantage of this representation is that it is possible to see a specific failed step in a test, but not only the failed test in total like in 2 others testing tools.
FEST	Visual indication of failed FEST tests is the same as JUnit tests. Failed test is displayed with a red cross before the test class name and failed testsuite is also marked with red color.

8. Documentation

- **introduction guide**

UISpec4J	Introduction guide is provided on the official site of UISpec4J. It is quite short and does not have a complete peace of testing code, which makes starting with UISpec4J not as easy as it is desired.
Jubula	Introduction guide is provided in a form of “cheat sheets” – small practical examples, which are built in a last Eclipse release together with Jubula itself. Also good introduction can be found on the next link [14]. Introduction guide is good and make it is to start with Jubula testing.
FEST	Introduction guide is provided on the official site of FEST. It is quite short, but it includes two complete tests, which help to start with test writing significantly.

- **Javadoc API specification**

UISpec4J	Javadoc API documentation is provided and can be integrated in the Eclipse project. This makes code writing easy and intuitively.
Jubula	API documentation is provided in a form of Jubula Reference Manual. It is integrated in the last Eclipse version and can be found in the Eclipse help section.
FEST	Javadoc API documentation is provided and can be integrated in the Eclipse project. This makes code writing easy and intuitively.

- **examples provided**

UISpec4J	Examples are included in a tutorial, provided on the official site of UISpec4J. Also two simple projects can be found in the download section of the site. These projects are excellent start point to get familiar with UISpec4J.
Jubula	Examples are included in a Jubula User Manual, as well as in Jubula “cheat sheets” – short examples, which guide user through the simple test

	constructing step by step.
FEST	Examples are included in documentation, provided on the official site of FEST. Also some nice examples can be found on the slides, provided by the FEST developer [13].

- **user guide**

UISpec4J	User guide is provided in a form of several different tutorials. It can be accessed through the UISpec4J official site. User guide is quite limited, mainly describes only the most basic functions and actions, though it includes code examples.
Jubula	User guide is provided in a form of Jubula User Manual. It is integrated in the last Eclipse version and can be found in the Eclipse help section. User guide is very well structured and detailed.
FEST	User guide is provided and a form of Wiki and HTML document. It can be accessed on the FEST official site. User guide is well-structured but it is not enough detailed. Some descriptions preferably should be more precise also.

9. Technical Support

- **supported forum**

UISpec4J	Forum exists; response on the forum is rather slow. Messages are displayed only after forum administrators' approval, which slows down response time.
Jubula	Forum exists, response on the forum is very quick, but the current number of messages on the forum is limited. Messages on the forum are displayed at the time of publishing, which increases a response time.
FEST	Forum exists; it has around 3000 messages, which promise a quick response or a possibility to find answer on the similar problem.

- **feedback from development team**

UISpec4J	Feedback from the development team took several days. It can be a problem in case of response is critical for a project. Feedback was helpful, explained how to share window of the application under test between different tests.
Jubula	Feedback from the development team was very quick, around 1 hour. It is impressive and helpful in a strict deadline conditions. Feedback was helpful.
FEST	Feedback from the development team is available. It was not used, because there was no necessity for a help during the tests development.

10. Reports

UISpec4J	Tests reports can be created using Ant build tool.
Jubula	Tests reports are not provided in Jubula. Tests reports are included only in commercial version of Jubula – GUIDancer.
FEST	Tests reports can be created using Ant build tool. Failure screenshots are built-in in reports (link to the failure screenshot is placed under the failure description in html file). This makes reports very helpful while identifying the failure's reason.

11. User Audience

UISpec4J	Testing tool can be used by developers or test engineers with a background in the development, because it requires knowledge in programming and code of the application under test.
Jubula	Testing tool can be used by test engineers without knowledge in programming as tool doesn't require to program tests. They can be done from the ready components.
FEST	Testing tool can be used by developers or test engineers with a background in the development, because it requires knowledge in

	programming and code of the application under test.

12. Weaknesses

UISpec4J	1. Timeout for heavy elements loading – makes test longer in total, because timeout should be enough to wait for an element. Sometimes, timeout is too long – then system is wasting testing time, sometimes timeout is too short – then test fails, because necessary element was not found. 2. Swing components preferably should be named. Unnamed components also can be found using other characteristics, but named components makes testing code well-readable, simple and searching process – more effective. 3. There is no way to test window closing. Closing of application force test termination. 4. Cannot test icons. 5. Cannot test focus on the component if component is not visible on the screen.
Jubula	1. Timeout for heavy elements loading – makes test longer in total, because timeout should be enough to wait for an element. Sometimes, timeout is too long – then system is wasting testing time, sometimes timeout is too short – then test fails, because necessary element was not found. 2. Bad logical objects separation leads to testing code redundancy. 3. Interface with many windows takes a time to get familiar with. 4. Extending of Jubula with custom components is a quite complicated task.
FEST	1. Timeout for heavy elements loading – makes test longer in total, because timeout should be enough to wait for an element. Sometimes, timeout is too long – then system is wasting testing time, sometimes timeout is too short – then test fails, because necessary element was not found. 2. Swing components preferably should be named. Unnamed components also can be found using other characteristics, but named components makes testing code well-readable, simple and

	searching process – more effective. 3. Lookup for components without names significantly reduce readability of testing code. 4. Throws errors without any descriptions. 5. Look first for internal names of components, but not visible ones (labels), which is unnatural for visual GUI testing and makes testing harder (Swing components almost never have internal names, but often have labels). 6. Tests run is time consuming. 7. Mouse cannot be touched while tests execution. 8. JUnit setUpBeforeClass and tesrDownAfterClass methods are not recognised.

13. Extension options

UISpec4J	Tool can be extended to recognise custom components. There is necessary to add asm-[version].jar and asm-utils-[version].jar libraries to the classpath to make extension work. Extension of standard set of Swing components with the custom one assume a wrapper for the custom component and *.jar file, which was generated for the new component and was added to the classpath.
Jubula	Tool can be extended to recognise custom components. Extension for the custom component requires completing a set of actions, described on the site [11]. Extending of Jubula requires knowledge of programming, because there is necessary to write a new class for a custom component and plugin it to existing code of Jubula testing tool. So, Jubula loose the benefit of “tool, which can be used by testers without development background”, when it comes to its extension.
FEST	Tool can be extended to recognise custom components. There is necessary to extend ComponentFixture class or implement interface, which is related to a new component (for example, MouseInputSimulationFixture for implementation of mouse input in a custom component, KeyboardInputSimulationFixture for implementation of keyboard input).

5 Decision matrix construction

5.1 Brief evaluation by criteria

After detailed evaluation of testing tools by criteria, table, which contains evaluations by different scales, was constructed. This table is a basis for future calculations, which will identify the best GUI testing tool for a Scila application.

Table 5.1 – Brief evaluation of testing tools by criteria

	UISpec4J	Jubula	FEST
1. Configuration			
supported ways	2	1	2
complexity	low	medium+	medium
2. Use			
complexity of tool	medium	low	medium+
time to get familiar with tool	long	short	long
3. Swing GUI Objects Recognition			
recognition of standard Swing objects	good	good	good
recognition of custom objects	require extension	require extension	require extension
4. Creation method	programming	constructing from blocks, visual recording	programming
5. Playback	has no visual reproducing	has visual reproducing	has visual reproducing

6. Debugging Support			
debugging tools provided	2	1	2
effectiveness	good	good	good
7. Recovery System			
failure screenshots	not provided	created automatically	require manual configuration
failure messages	well-descriptive	well-descriptive	poor-descriptive
visual indication	exist	exist	exist
8. Documentation			
introduction guide	intermediate	excellent	good
Javadoc API specification	good	excellent	good
examples provided	good	excellent	intermediate
user guide	intermediate	excellent	intermediate
9. Technical Support			
supported forum	helpful	helpful	helpful
feedback from development team	good	good	good
10. Reports	supported	not supported	supported
11. User Audience	developers	developers, test engineers	developers
12. Weaknesses	5	4	8
13. Extension options	medium complexity	high complexity	medium complexity

5.2 Ranking and weighting criteria

The next step was ranking and weighting criteria. This step was necessary to understand which criteria are more important and which are less important. Assigning weights to criteria will then influence the final evaluation of testing tools.

To get the percentage estimations of criteria, pairwise comparison [15] of criteria was done first. Results of this comparison are in table 2. The sense of pairwise comparison is next: criteria A is compared to criteria B. If A is more important – A is written in an intersection cell, if B is more important – then B is written in the cell, if both – AB.

Pairwise comparison of criteria was done taking in consideration requirements to a testing tool, stated by Scila (paragraph 3.1). Stated requirements were assigned more importance than other criteria.

Table 5.2 – Pairwise comparison of criteria

		A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**
supported configuration ways	**A**	-	B	C	D	E	F	AG	H	AI	J	K	L	M	N	O	P	Q	AR	AS	AT	AU	V	AW
complexity (configuartion)	**B**	-	-	C	BD	E	BF	BG	B	BI	BJ	K	L	BM	B	O	P	Q	BR	B	B	B	BV	B
complexity of tool	**C**	-	-	-	CD	E	C	CG	H	I	J	K	L	C	C	CO	CP	CQ	C	C	CT	CU	C	C
time to get familiar	**D**	-	-	-	-	E	D	D	H	I	J	DK	DL	M	N	O	P	Q	DR	DS	T	DU	V	DW
recognition of standard Swing objects	**E**	-	-	-	-	-	E	E	E	EI	EJ	E	EL	E	E	EO	E	EQ	E	E	E	E	E	E
recognition of custom objects	**F**	-	-	-	-	-	-	G	H	I	J	FK	FL	FM	FN	O	P	Q	F	F	F	FU	V	FW
creation method	**G**	-	-	-	-	-	-	-	GH	I	J	GK	GL	GM	G	O	P	Q	G	G	G	GU	G	G
playback	**H**	-	-	-	-	-	-	-	-	HI	HJ	H	HL	H	H	HO	HP	HQ	HR	H	H	H	HV	HW
debugging tools provided	**I**	-	-	-	-	-	-	-	-	-	IJ	IK	IL	I	I	IO	IP	IQ	I	I	IT	I	I	IW
effectiveness	**J**	-	-	-	-	-	-	-	-	-	-	JK	JL	J	J	JO	JP	JQ	J	J	J	J	J	JW
failure screenshots	**K**	-	-	-	-	-	-	-	-	-	-	-	KL	KM	K	O	P	Q	K	K	K	K	K	W

failure messages	**L**	-	-	-	-	-	-	-	-	-	-	-	-	L	L	LO	LP	LQ	L	L	L	L	L	L
visual indication	**M**	-	-	-	-	-	-	-	-	-	-	-	-	-	N	O	P	Q	MR	M	MT	M	M	W
introduction guide	**N**	-	-	-	-	-	-	-	-	-	-	-	-	-	-	NO	P	Q	NR	N	T	U	V	W
Javadoc API spec.	**O**	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	OP	OQ	OR	O	O	O	O	O
examples provided	**P**	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	PQ	PR	P	P	P	P	PW
user guide	**Q**	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	RQ	Q	Q	Q	Q	QW
supported forum	**R**	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	R	RT	R	R	W
feedback from development team	**S**	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	S	S	V	W
reports	**T**	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	U	V	W
user audience	**U**	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	U	W
weaknesses	**V**	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	V
extension options	**W**	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

After pairwise comparison of criteria was finished, number of letters for criteria was calculated to get ranks for them:

1. Configuration
    - supported ways (A) - 7
    - complexity (B) - 15
2. Use
    - complexity of tool (C) - 16
    - time to get familiar with tool (D) - 11
3. Swing GUI Object Recognition
    - recognition of standard Swing objects (E) - 22
    - recognition of custom objects (F) - 11
4. Creation method (G) - 15
5. Playback (H) - 20

6. Debugging Support
    - debugging tools provided (I) - 22
    - effectiveness (J) - 22
7. Recovery System
    - failure screenshots (K) - 16
    - failure messages (L) - 22
    - visual indication (M) - 11
8. Documentation
    - introduction guide (N) - 7
    - Javadoc API specification (O) - 22
    - examples provided (P) - 21
    - user guide (Q) - 22
9. Technical Support
    - supported forum (R) - 13
    - feedback from development team (S) - 4
10. Reports (T) - 7
11. User Audience (U) - 8
12. Weaknesses (V) - 9
13. Extension options (W) – 15

The next step was associating weights with criteria that were received. Linear proportion formula allows resolving this task:

$$100 = 7x + 15x + 16x + 11x + 22x + 11x + 15x + 20x + 22x + 22x + 16x + 22x + 11x + 7x + 22x + 21x + 22x + 13x + 4x + 7x + 8x + 9x + 15x$$

Approximate calculated value of $x = 0.3$.
Coefficients in the equation correspond to number of letters for criteria calculated above.
This leads to weights (in percentages):

1. Configuration
    - supported ways (A) – 2.1%
    - complexity (B) – 4.5%
2. Use
    - complexity of tool (C) – 4.8%
    - time to get familiar with tool (D) – 3.3%

3. Swing GUI Object Recognition
    - recognition of standard Swing objects (E) – 6.6%
    - recognition of custom objects (F) – 3.3%
4. Creation method (G) – 4.5%
5. Playback (H) – 6%
6. Debugging Support
    - debugging tools provided (I) – 6.6%
    - effectiveness (J) – 6.6%
7. Recovery System
    - failure screenshots (K) – 4.8%
    - failure messages (L) – 6.6%
    - visual indication (M) – 3.3%
8. Documentation
    - introduction guide (N) – 2.1%
    - Javadoc API specification (O) – 6.6%
    - examples provided (P) – 6.3%
    - user guide (Q) – 6.6%
9. Technical Support
    - supported forum (R) – 3.9%
    - feedback from development team (S) – 1.2%
10. Reports (T) – 2.1%
11. User Audience (U) – 2.4%
12. Weaknesses (V) – 2.7%
13. Extension options (W) – 4.5%

5.3 Choosing ranking scale

Criteria for evaluation of the GUI testing tools are very different, and as a result they have different scales. To present evaluation process in numerical values, it was necessary to convert all criteria scores to a single ranking scale. Common scale for criteria is underlined below in a table 3:

Table 5.3 – Symmetrical ranking scale [12]

Rating	Meaning
-2	Greatly inferior compared to the alternatives
-1	Somewhat inferior
0	Satisfactory, but nothing to write about
1	Good; somewhat superior
2	Excellent; greatly superior

The converted values are displayed in table 4:

Table 5.4 – Converting to a common ranking scale

	UISpec4J	Jubula	FEST
1. Configuration			
supported ways	2	1	2
complexity	2	-1	0
2. Use			
complexity of tool	0	1	-1
time to get familiar with tool	0	1	-1
3. Swing GUI Objects Recognition			
recognition of standard Swing objects	2	2	2
recognition of custom objects	0	-1	0
4. Creation method	0	2	0
5. Playback	0	2	2
6. Debugging Support			
debugging tools provided	2	1	2
effectiveness	1	1	1
7. Recovery System			

failure screenshots	-1	2	0
failure messages	1	1	-1
visual indication	2	2	2
8. Documentation			
introduction guide	0	2	1
Javadoc API specification	1	2	1
examples provided	1	2	0
user guide	0	2	0
9. Technical Support			
supported forum	1	1	1
feedback from development team	1	1	1
10. Reports	1	-1	1
11. User Audience	1	2	1
12. Weaknesses	-1	1	-2
13. Extension options	-1	-2	-1

5.4 Calculating decision matrix

Decision matrix for evaluation of testing tools was calculated based on the next algorithm:

1. Multiply each weight by each rating for UISpec4J and write result to the Score column.
2. Sum scores for the UISpec4J and write result in the TOTAL.
3. Repeat actions 1-4 for Jubula and FEST testing tools.
4. Considering numbers in TOTAL row, assign ranks to the testing tools (1, 2 or 3).
5. Write ranks to the RANK row.

The decision matrix is presented below:

Table 5.5 – Decision matrix

		Concepts					
		UISpec4J		Jubula		FEST	
Criteria	**Weight**	**Rating**	**Score**	**Rating**	**Score**	**Rating**	**Score**
1. Configuration							
supported ways	0.021	2	0.042	1	0.021	2	0.042
complexity	0.045	2	0.09	-1	-0.045	0	0
2. Use							
complexity of tool	0.048	0	0	1	0.048	-1	-0.048
time to get familiar with tool	0.033	0	0	1	0.033	-1	-0.033
3. Swing GUI Objects Recognition							
recognition of standard Swing objects	0.066	2	0.132	2	0.132	2	0.132
recognition of custom objects	0.033	0	0	-1	-0.033	0	0
4. Creation method	0.045	0	0	2	0.09	0	0
5. Playback	0.06	0	0	2	0.12	2	0.12
6. Debugging Support							
debugging tools provided	0.066	2	0.132	1	0.066	2	0.132
effectiveness	0.066	1	0.066	1	0.066	1	0.066

7. Recovery System							
failure screenshots	0.048	-1	-0.048	2	0.096	0	0
failure messages	0.066	1	0.066	1	0.066	-1	-0.066
visual indication	0.033	2	0.066	2	0.066	2	0.066
8. Documentation							
introduction guide	0.021	0	0	2	0.042	1	0.021
Javadoc API specification	0.066	1	0.066	2	0.132	1	0.066
examples provided	0.063	1	0.063	2	0.126	0	0
user guide	0.066	0	0	2	0.132	0	0
9. Technical Support							
supported forum	0.039	1	0.039	1	0.039	1	0.039
feedback from development team	0.012	1	0.012	1	0.012	1	0.012
10. Reports	0.021	1	0.021	-1	-0.021	1	0.021
11. User Audience	0.024	1	0.024	2	0.048	1	0.024
12. Weaknesses	0.027	-1	-0.027	1	0.027	-2	-0.054
13. Extension options	0.045	-1	-0.045	-2	-0.09	-1	-0.045
TOTAL			0.699		1.173		0.495
RANK			2		1		3

6 Results

6.1 Testing tool chosen for Scila application

Evaluation of testing tools was finished with ranking of them based on the scores gained. Testing tool, which received the highest score was Jubula. The lowest score gained FEST and UISpec4J was in the middle between Jubula and FEST.
It was decided to choose Jubula for testing of the Scila application as this testing tool is the best according to the evaluation and also it optimally fits the needs of Scila:

1. It is fully compatible with Scila Surveillance application.
2. It is an open-source tool.
3. Jubula is able to test all the standard Swing GUI components and can be extended to test custom components.
4. Jubula provides good techniques for testing of tables.
5. Test cases can be created quickly and easily without any scripting effort.
6. Effective debugging is provided. Screenshots are taken automatically in case of test failure.

Comparison of total scores for each testing tool can be seen from the diagram 6.1:

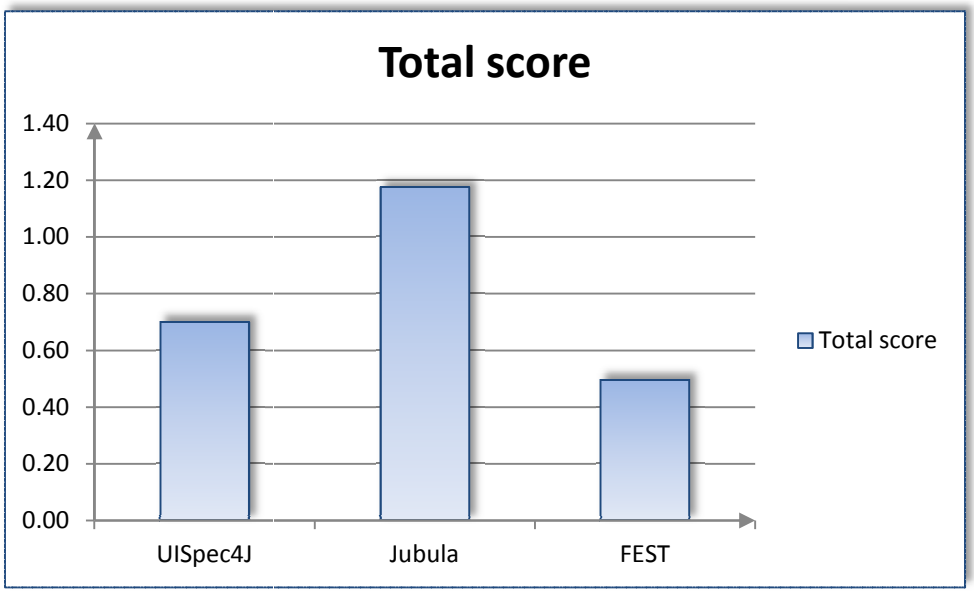

Figure 6.1 – Total score for each testing tool

6.2 Implemented test suite

Jubula testing tool was used to make a test suite for a Statistics module of the Scila Surveillance application. Statistics module includes Statistics Overview, Global Statistics and Participant Statistics parts. Implemented test cases cover main test paths from the every part.

Three test suits were created for Scila application testing:

1. FULLTEST – includes all the working test cases.
2. FULLTEST_BROKEN – includes currently not working test cases.
3. WORK_(INITIALS) – includes test cases, which are under the construction.

FULLTEST tests suite consists of six test cases:

1. <Start Application> - starts application and performs login.
2. <Check a row in Participant Statistics table> - check that all values in a first row are correct.
3. <Check a row in Global Statistics table> - check that all values in a first row are correct.
4. <Check sorting for Today's Winners table> - check sorting in descending order.
5. <Check sorting for Today's Losers table> - check sorting in ascending order.
6. <Check sorting in Global Statistics table> - check sorting performed by different sorting criteria.

Created test suite can be seen below:

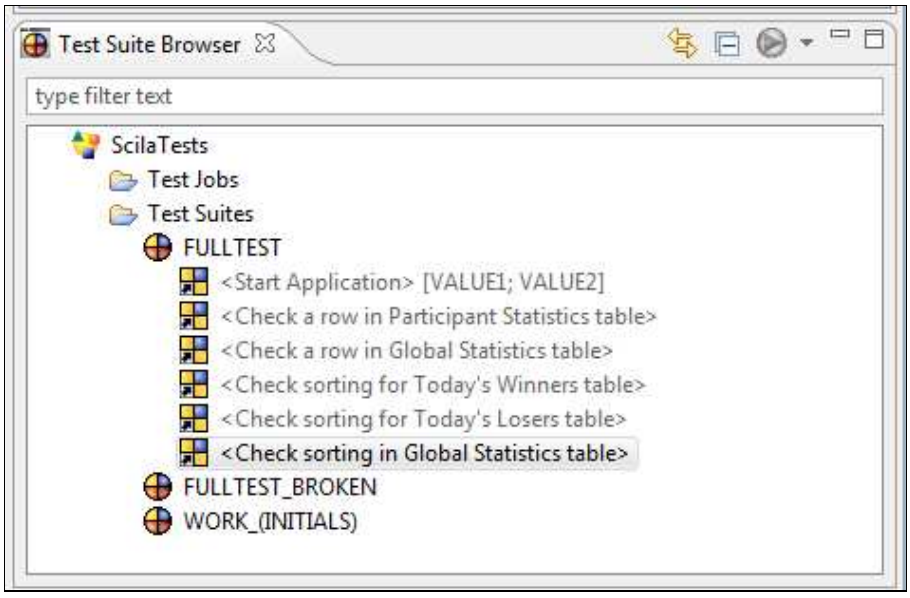

Figure 6.2 - Test suite

Test Case Browser (Figure 6.3) displays test cases organised in the following order:

1. EXECUTABLE TESTS – includes complete test cases, organised by application tabs, which they test.
    - Statistics tab – includes tests for a Statistics module.
        - Global Statistics – includes tests for tables on the Global Statistics tab.
            - ✓ Check a row in Global Statistics table – test case, which check each value in the first row of the Global Statistics table.
            - ✓ Check sorting in Global Statistics table – test case which check that Global Statistics table can be sorted right by all the available sorting criteria.
        - Participant Statistics – includes tests for tables on the Participant Statistics tab.
            - ✓ Check a row in Participant Statistics table – test case, which check each value in the first row of the Participant Statistics table.
        - Statistics Overview – includes tests for tables on the Statistics Overview tab.
            - ✓ Check sorting for Today's Losers table – test case, which check that Today's Losers table is sording according to the Losers column.
            - ✓ Check sorting for Today's Winners table – test case, which check that Today's Winners table is sording according to the Winners column.
2. MODULES – includes reusable test parts, which can be used in many different test cases.
    - Select a date in Global Statistics – reusable part of testcase, which select a fixed date in the Global Statistics table. Fixed date allows to get values in a table for a specific day, month and year.
    - Select a date in Participant Statistics – reusable part of testcase, which select a fixed date in the Participant Statistics table. ytFixed date allows to get values in a table for a specific day, month and year.
    - Select a date in Statistics Overview – reusable part of testcase, which select a fixed date in the Statistics Overview table. Fixed date allows to get values in a table for a specific day, month and year.
    - Sort by criteria [TEXT] – reusable part of test case, which accepts sorting criteria (TEXT), and performs sorting of the table by the selected criteria.
    - Start Application [VALUE1; VALUE2] – reusable part of test case, which accepts login (VALUE1) and password (VALUE2), perform login process to

the Scila Surveillance application and check if the main application window appeared.

Reusing of modules is a core principle of the Jubula testing tool. This makes test cases structured and eliminate redundancy in a test suite.

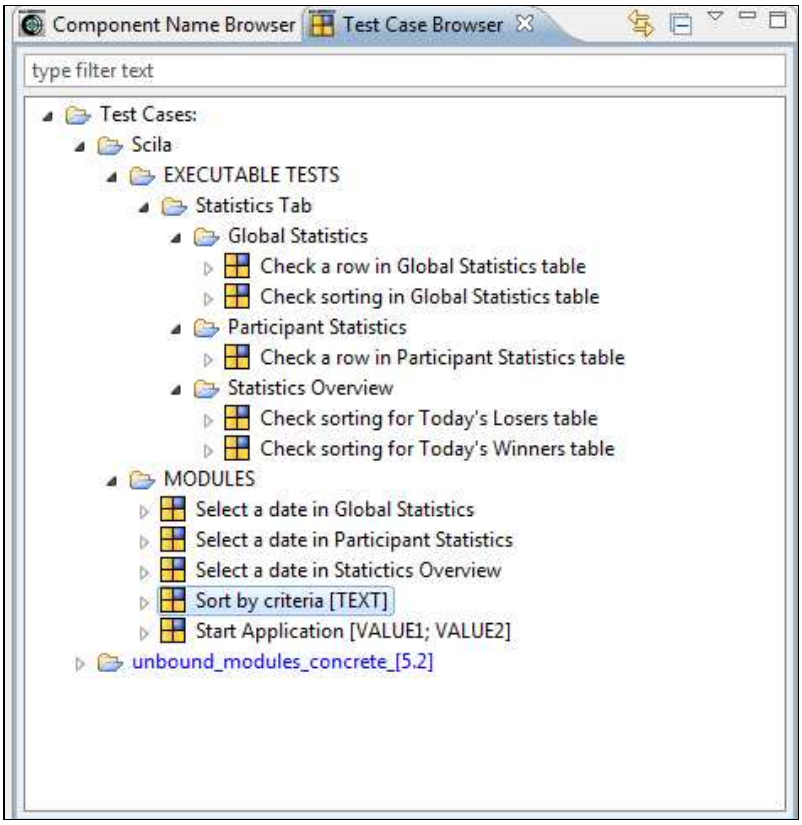

Figure 6.3 - Tests by packages

Figure 6.4 displays an example of a test case ‘Check sorting in Global Statistics table’. The aim of the test case is to check sorting for Global Statistics table by all possible sorting criteria. Test case consists of fifteen steps:

1. Select Statistics/Global Statistics submenu – selects submenu by the text path Statistics/Global Statistics.
2. <Select a date in Global Statistics> - reusable part, choose a fixed date for a data in Global Statistics table.
3. Sort by Turnover (Sort by criteria) [TEXT] – perform sorting by Turnover column – sorting criteria.
4. Check if numbers are sorted – check if values in Global Statistics table are sorted by the Turnover criteria.
5. Sort by Traded Volume (Sort by criteria) [TEXT] – perform sorting by Traded Volume column – sorting criteria.
6. Check if numbers are sorted – check if values in Global Statistics table are sorted by the Traded Volume criteria.

7. Sort by Gain (Sort by criteria) [TEXT] – perform sorting by Gain column – sorting criteria.
8. Check if numbers are sorted – check if values in Global Statistics table are sorted by the Gain criteria.
9. Sort by Loss (Sort by criteria) [TEXT] – perform sorting by Loss column – sorting criteria.
10. Check if numbers are sorted – check if values in Global Statistics table are sorted by the Loss criteria.
11. Sort by Swing (Sort by criteria) [TEXT] – perform sorting by Swing column – sorting criteria.
12. Check if numbers are sorted – check if values in Global Statistics table are sorted by the Swing criteria.
13. Sort by VWAP (Sort by criteria) [TEXT] – perform sorting by VWAP column – sorting criteria.
14. Check if numbers are sorted – check if values in Global Statistics table are sorted by the VWAP criteria.
15. Close the Global Statistics tab – close the tab by clicking the cross button.

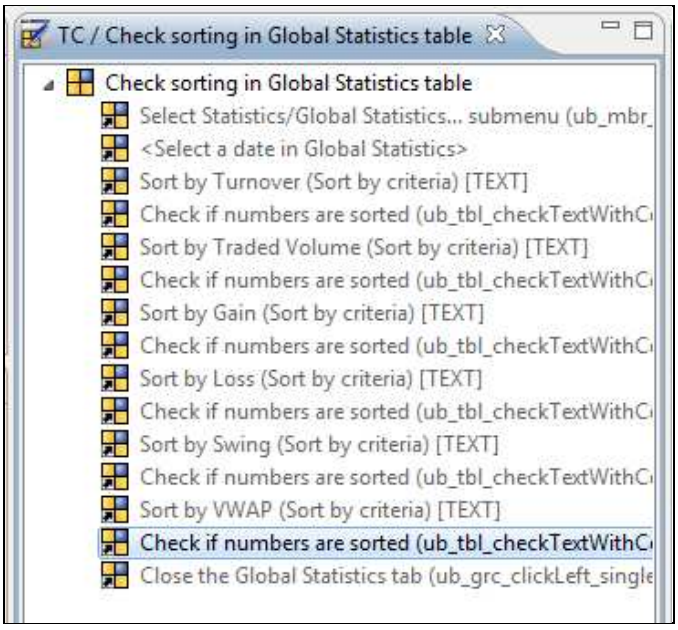

Figure 6.4 - Test case example

7 Conclusion

This thesis work aimed to investigate methods for automated testing in large-scale financial systems. Study of previous research papers in this area showed that automated methods, used in GUI testing, vary significantly in their characteristics. The same huge is difference between different comparison strategies, selected for comparison process. Almost every researcher, every company, who conducts study in test automation area, creates its own comparison method, which can be just plain text description of advantages and disadvantages of concrete testing method, or it can be score-based marks accompanied by well-descriptive diagrams and list of calculations done.

After detailed studies of previous works on selection of testing methods it was decided, first of all, to make their practical comparison. As this thesis research results were expected to be implemented in a testing of Scila Surveillance financial application, pure theoretical comparison would not be able to display the actual situation.

Therefore, pure scripting method, scripting method with elements of record-replay and non-scripting method were tried and compared in this thesis research. Non-scripting method was found as the best one, as it requires short time for configuration of testing tool, and even shorter time for test development. This saves testing time dramatically. Moreover, non-scripting method appeared to be suitable for needs of test engineers as well as manual testers, because it does not require any programming skills from a tester. Also, non-scripting testing method resolved a key problem, which is well-known in all rapidly growing projects – tests maintenance. Tests maintenance is much simpler, while tester does changes in non-scripted blocks positions and their properties, rather than in programmed classes and functions. It is worth to say, that disadvantages of non-scripting testing method also exist. Non-scripting testing tool assumes usage of objects, predefined by its developer. This found to be a complicated side in case testers need to automate testing of custom objects. Extension of non-scripting testing tools is a complicated non-trivial task, which requires involvement of developers. As Scila Surveillance uses mostly standard GUI objects, this disadvantage is currently minor, but in case of need non-scripting testing method can be always followed by scripting one to use a full power of programming languages for automation of out-of-box objects and testing tasks.

Finally, there is necessary to emphasise that a future research in test automation methods can and should be done. Testing methods should grow with the evolution of software product and technologies they are testing. Currently, researches distinguish five generations of testing methods [17], [18]: record-replay, data-driven, functions reuse, scripting and non-scripting testing approaches. Non-scripting technology is modern and should be developed and used further. As about comparison methodology, future research can focus on developing of common unified approach. It will simplify future selection of testing method for companies, as testing criteria will be the same and problem of different comparison criteria will not affect the final results.

And of course, free-based testing tools should be investigated in details all the time as they are quite powerful and promising in test automation, though sometimes are not the same wide-spread as commercial ones.

8 Discussion

At the beginning of this thesis work it was decided to compare several different testing methods for implementation of GUI testing in large-scale financial system Scila Surveillance. Testing methods selected were expected to be tried in practise rather than investigated only in their theoretical advantages and disadvantages. In reality it was found that some of the methods do not have suitable fully functioning testing tools, which uses these methods in a basis. Furthermore, problem was even more complicated because of requirement to compare only free open-source testing tools, so the number of them was rather limited. For example, it was found that Cucumber – testing tool, which uses behaviour-driven approach, does not have stable current version. Previous version is no more supported and new version is still in the process of development.

Another problem was that record-replay testing tool appeared to be useful for GUI testing in general, but was found unable to test heavy Swing GUI objects like JTables. As application under the test is constructed mostly from tables and testing of them was a core requirement, it was not possible to continue with WindowTester Pro, and as a result, record-replay method was not investigated to a proper level.

There is also necessary to mention once again, that current list of free testing tools is huge only on the first glance. Actually, only some of them are implemented to that extent that can be competitors to commercial testing tools. This fact narrowed the possibility to select several strong testing tools for each existing testing method. Mostly, each testing approach appeared to be represented only by one really powerful free testing tool. Finally, it was not found any unified comparison methodology neither for testing approaches, nor for testing tools, which uses these approaches. It was decided to study all comparison strategies, tried before by researches in their scientific papers, but especially accent was made on the comparison methodologies, used by quality assurance companies. This thesis starts with a detailed comparison at the beginning, to describe all the sides of testing tools, and it finishes with formalization of found results by assigning weights and scores to candidates, calculating of the total numerical result. This approach seems very clear and understandable, as information, presented in numbers is the easiest for perception and comparison, but of course, this method can be arguable from the point of view of other researches in a test automation area.

9 References

[1]. Google Insights. Swing Testing (2011), viewed January 9 2012. http://laurii.info/wp-content/uploads/2012/01/swing-00-google-insights.png

[2]. WindowTester Pro forum (2011), viewed January 10 2012. http://forums.instantiations.com/topic-5-6064.html

[3]. UISpec4J. Project Goals, viewed January 17 2012. http://www.uispec4j.org/goals

[4]. UISpec4J. Features (2012), viewed January 20 2012. http://www.uispec4j.org/features

[5]. Aslak Hellesoy (2011). Transitioning from Cuke4Duke to Cucumber-JVM, viewed January 20 2012. http://groups.google.com/group/cukes/browse_thread/thread/299d94d38500e8c3

[6]. Google Insights for Search (2012), viewed January 25 2012. http://www.google.com/insights/search/

[7]. WindowTester Pro User Guide (2012), viewed February 1 2012. http://code.google.com/intl/sv-SE/javadevtools/wintester/html/index.html

[8]. Cucumber. Behaviour driven development with elegance and joy (2012), viewed February 7 2012. http://cukes.info/

[9]. Fixtures for Easy Software Testing (2012), viewed February 12 2012. http://fest.easytesting.org/

[10]. Jubula. Automated functional testing (2012), viewed February 17 2012. http://www.eclipse.org/jubula/

[11]. Torsten Witte (2012). Extending Jubula for custom RCP Applications, viewed February 23 2012. http://www.subshell.com/en/subshell/blog/Extending-Jubula-For-Custom-SWT-Controls-In-RCP-Applications100.html

[12]. Decision matrix (2004-2011), viewed February 27 2012. http://deseng.ryerson.ca/xiki/Learning/Main:Decision_matrix

[13]. Alex Ruiz, Yvonne W.Prince (2010). Easy GUI testing with FEST, Oracle corporation, viewed March 2 2012. http://www.slideshare.net/Softwarecentral/easy-gui-testing-with-fest

[14]. Joachim Hofer (2011). UI Testing with Eclipse Jubula: Preparing the Test Object, viewed March 5 2012. http://jmhofer.johoop.de/?p=97

[15]. Pairwise Comparison (2004-2011), viewed March 11, 2012. http://deseng.ryerson.ca/xiki/Learning/Main:Pairwise_comparison

[16]. Tim Bower, George Wilson (2012). How TestDrive Works, Original Software, viewed March 15 2012. http://www.origsoft.com/webinars/test_automation_debate/test_automation_debate.pdf

[17] John Kenth (2009). Test Automation: From Record/Playback to Frameworks, viewed April 20 2012.
http://www.simplytesting.com/Downloads/Kent%20-%20From%20Rec-Playback%20To%20FrameworksV1.0.pdf
[18] David Hunt (2012). Software Test-on-Demand - A Better Way Forward, Infuse Consulting, viewed April 20 2012.http://www.infuse.it/infuse-com/_img/pdfs/testondemand_mktg/WP_001_Test-on-Demand_V1b.pdf
[19] Paul Sudipto (2011). Scriptless Test Automation. Next Generation Technique for Productivity Improvement in Software Testing, Geometric, viewed April 20 2012.
http://products.geometricglobal.com/solutions/downloads/Geometric_Whitepaper_scriptless_test_automation.pdf
[20] Jeff Hinz, Martin Gijsen (2012). Fifth Generation Scriptless and Advanced Test Automation Technologies, viewed April 20 2012.
http://www.testars.com/docs/5GTA.pdf
[21] Aslak Hellesoy, Matt Wynne (2012). The Cucumber Book: Behaviour-Driven Development for Testers and Developers, Pragmatic Bookshelf.
[22] T.Illes, A.Herrmann, B. Paech, J. Rückert (2005). Criteria for Software Testing Tool Evaluation – A Task Oriented View. Institute for Computer Science, University of Heidelberg.
[23] Terry Horwath (2007). Automated Test Tools. Evaluation Criteria, viewed April 27 2012.
http://www.slideshare.net/basma_iti_1984/testing-tool-evaluation-criteria-presentation
[24] Cordell Vail (2005). Stress, Load, Volume, Performance, Benchmark and Base Line Testing Tool Evaluation and Comparison, viewed April 27 2012.
http://www.vcaa.com/tools/loadtesttoolevaluationchart-023.pdf
[25] E.Gomonova, I. Anisimov, I. Petrov, D. Kondtratiev, D.Zernov, O. Moroz (2007). Automated Testing Tools Comparison, Luxoft, viewed April 27 2012.
http://www.luxoft.com/downloads/white_papers/Automated%20Testing%20Tools.pdf
[26] Ray Robinson (2001). Automation Test Tools Comparison, viewed April 27 2012.
http://www.automatedtestinginstitute.com/home/index.php?option=com_content&view=article&catid=19:techniquestoolsmithing&id=133:automation-test-tools-comparison