International Journal of Industrial Engineering and Management (IJIEM), Vol.1 No 4, 2010, pp. 155 - 161
Available online at www.ftn.uns.ac.rs/ijiem
ISSN 2217-2661

Requirements-Based Testing Process in Practice

(Originally presented as "*Getting it right the first time*")

Predrag Skoković

Test Manager, Execom, Novi Sad, Serbia, pskokovic@execom.eu

Marija Rakić-Skoković

Teaching assistant, Faculty of Technical Sciences, Novi Sad, Serbia, marijars@uns.ac.rs

Received (05.12.2010); Revised (23.12.2010); Accepted (24.12.2010)

Abstract

In many organizations, testing, regarded as quality verification, begins only once code has been completed. However, errors found in requirements are the leading cause of project failures, defects and rework. Even though many companies use some method of requirements management and some form of software quality testing, most of them cannot (or do not) link them together.

While searching for an application lifecycle management (ALM) methodology that might give us an answer to these problems, Requirements-Based Testing (RBT) emerged as a possible solution. RBT provides a set of quality assurance activities and management tools that enable getting requirements right from the outset.

This paper presents lessons learned while introducing requirements-based testing methodology, in order to put the project in control and deliver applications on time.

Key words: *application lifecycle management, process improvement, requirements, testing*

1. INTRODUCTION

Many studies show that majority of software projects fail to achieve schedule and budget goals. The poor quality of software is one of the main reasons laying behind many failures. These often result in great rework of application requirements, design and code. Experience and numerous studies show that: behind poor software quality are defects in requirements specifications and problematic system test coverage. Put simply, low quality of input causes low quality of output no matter how project team is experienced, which methodology is used and which budget and timeline constraints are established.

According to James Martin the root causes of 56 percent of all defects identified in software projects are introduced in the requirements phase (Fig. 1). About 50 percent of requirements defects are result of poorly written, unclear, ambiguous, and incorrect requirements. The other 50 percent of requirements defects are due to incompleteness of specification (i.e. omitted requirements) [1].

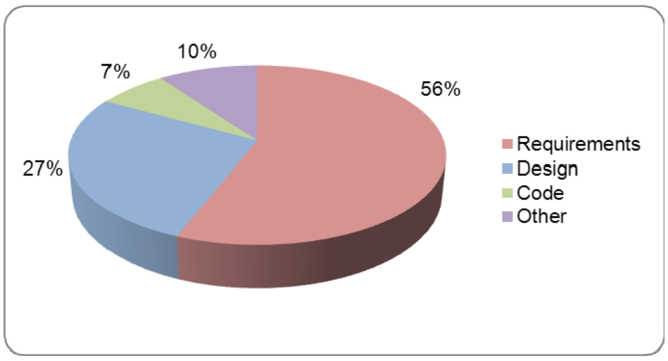


Figure 1. Distribution of defects in software projects

Other statistics point to similar problems [2]:

- 82 percent of application rework is related to errors in requirements.
- Problems in requirements represent 44 percent of the reasons behind project cancellations.
- Only 54 percent of initial project requirements are actually realized.

It is not uncommon that a system, which is thoroughly and successfully tested, makes its users unhappy.

Reason behind this fact is that the development team got the requirements wrong.

Even with all quality techniques practiced today, the lion's share of bugs is found by testing, which is performed typically after code is delivered. This makes testing by far the costliest method of finding bugs. One of the key goals of testing is to achieve optimal test coverage, in order to maximize a chance of finding a defect in testing phase. Reasons for making it very hard to accomplish good test coverage are [2]:

- Tests are often conducted at the end of the development process.
- The complexity of modern applications makes it very hard to cover all of the possible scenarios.
- Application requirements change frequently, but their changes are not properly managed.

During the development of a complex application, with strict time and budget frame, we have noticed some of the abovementioned problems. Major problem was recognized in incomplete and frequently modified requirements. This resulted in constant:

- Rework of design and code.
- Rework of test cases, in order to stay current with requirements, which led to shorter time for test execution than planned.
- Misunderstanding of requirements among members of development team.

All of the abovementioned denoted the need for methodology that could be easily and "cheaply" adopted, with methods that could positively resolve detected issues.

2. WHY REQUIREMENTS-BASED TESTING?

Guided with the definition of software quality (Fig. 2) and one of the general principals of software testing (Fig. 3), Requirements-Based Testing (RBT) methodology emerged as a solution to problems identified in project.

Software quality is:

1. The degree to which a system, component, or process meets specified requirements.
2. The degree to which a system, component, or process meets customer or user needs or expectations.

Figure 2. Definition of software quality suggested by The Institute of Electrical and Electronics Engineers (IEEE, 1991) [3]

Testing activities should start as soon as possible in the software lifecycle and focus on defined goals. This contributes to finding defects early.

Figure 3. Principle of early testing [4]

The focus of RBT is to discover and fix low quality of requirements thus making valid input which contributes greatly in defining clear scope of the project. By combining methods from requirements engineering and software testing, requirements-based testing methodology provides a set of quality assurance activities and management tools that enable getting requirements right from the outset. By using RBT it is possible to discover requirements errors before they become extremely expensive to fix and manage inevitable changes during software lifecycle.

3. OVERVIEW OF REQUIREMENTS-BASED TESTING

The requirements-based testing process addresses two major issues: first, validating that the requirements are correct, complete, unambiguous, and logically consistent; and second, designing a necessary and sufficient (from a black box perspective) set of test cases from those requirements, to ensure that the design and code fully meet the requirements. When designing tests two issues need to be overcome: reducing the enormous number of potential tests down a reasonable size set and ensuring that the tests got the right answer for the right reason. The RBT process does not assume that we will have good requirements specifications. The RBT process will drive out ambiguity and drive down the level of detail.

The overall RBT strategy is to integrate testing throughout the development life cycle and focus on the quality of the requirements specification. This leads to early defect detection which has been shown to be much less expensive than finding defects during integration testing or later. The RBT process also has a focus on defect prevention, not just defect detection [5].

To put the RBT process into perspective, testing can be divided into the following activities [5], [6]:

1. **Define Test Completion Criteria.** The test effort has specific, quantifiable goals. Testing is completed only when goals have been reached
2. **Design Test Cases.** Logical test cases are defined by four characteristics: the initial state of the system prior to executing the test, the data, the inputs, and the expected results.
3. **Build Test Cases.** There are two parts needed to build test cases from logical test cases: creating the necessary data, and building the components to support testing (e.g., build the navigation to get to the portion of the program being tested).
4. **Execute Tests.** Execute test-case steps against the system being tested and document the results.
5. **Verify Test Results.** Testers are responsible for verifying two different types of test results: Are the results as expected? Do the test cases meet the test completion criteria?
6. **Verify Test Coverage.** Track the amount of functional coverage achieved by the successful execution of each test.
7. **Manage and Track Defects**. Any defects detected during the testing process are tracked to resolution. Statistics are maintained concerning the overall defect trends and status.
8. **Manage the Test Library.** The test manager maintains the relationships between the test cases and the programs being tested. The test manager keeps track of what tests have or have not been executed, and whether the executed tests have passed or failed.

RBT addresses activities one, two, and six. The remaining activities are addressed by test management tools that track the status of test executions.

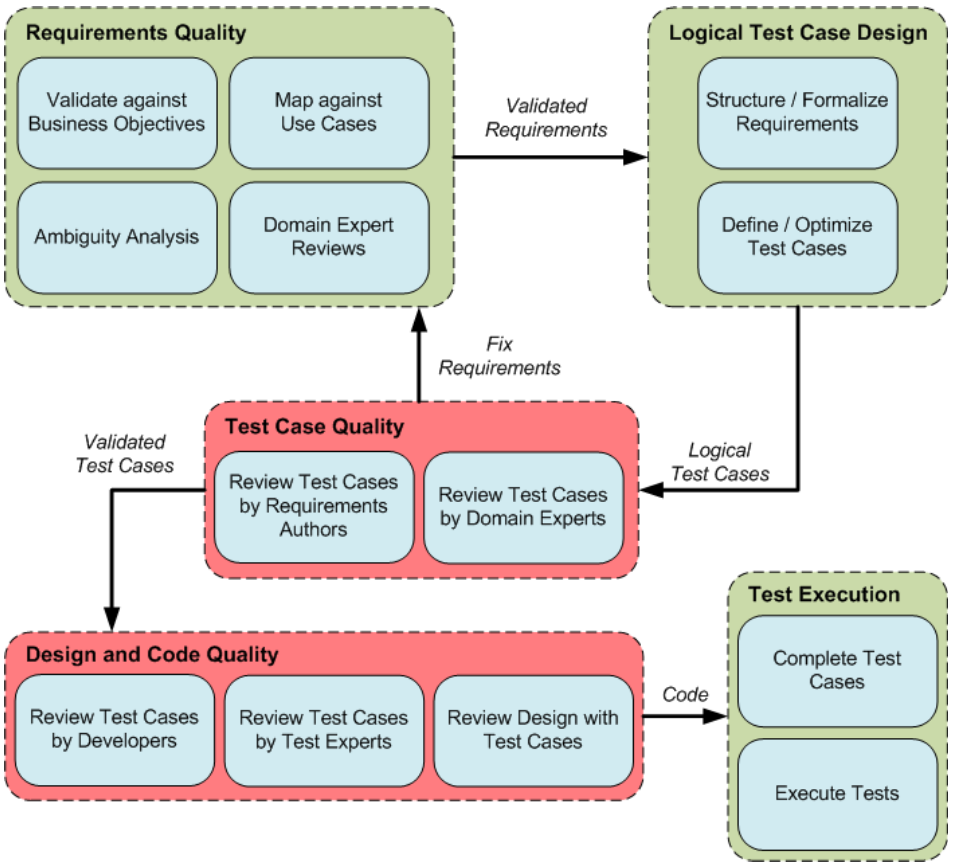


Figure 4. Requirements-based testing process flow

3.1 The RBT methodology

The RBT methodology is a 12-step process (Fig. 4) [6], [7]:

1. **Validate requirements against objectives.** Optimize project scope by ensuring that each requirement satisfies at least one business objective. If there is no match between the requirements and business objectives (if “what” does not match the “why”), refinement is necessary.
2. **Apply use cases against requirements.** Some organizations document their requirements with use cases. Map requirements against a task-

oriented or interaction-oriented view of the system. If one or more use-cases cannot be addressed by the requirements, then the requirements are not complete.

3. **Perform an initial ambiguity review.** An ambiguity review is a technique for identifying and eliminating ambiguous words, phrases, and constructs. It is not a review of the content of the requirements. The ambiguity review produces a higher-quality set of requirements for review by the rest of the project team.

4. **Perform domain expert reviews.** Feedback of users and domain experts should be used to refine the requirements before additional work is done.

5. **Structure and formalize requirements.** To systematically achieve high test coverage formal and structured representations of requirements need to be created.

   Multiple techniques can be used to provide structure and formality to natural language requirements. The purpose of these techniques is to reveal cause-effect relationships embedded within requirements, that is to express requirements as a set of conditions (causes) and resulting actions (effects). Cause-effect charting is one of these techniques. Another way to achieve similar goals is to express requirements as flow charts, since they naturally depict precedence dependency between actions as well as conditional branching of activities.

   Once this is done, it is possible to define “logical” test cases, which will ensure optimal coverage of requirements, while evolving into actual tests that will be run against the system.

6. **Logical consistency checks performed and test cases designed.** A set of logical test cases can be defined (manually or automatically), which is exactly equivalent to the functionality captured in the requirements. However, this set of test cases may include many redundant cases (i.e. overlapping with other test cases).

   To optimize the number of test cases but still provide full coverage, techniques such as decision (truth) tables can be applied if cause-effects charts were used to structure the requirements. If flow charts were used for that purpose, then generation of optimal set of test cases means finding all unique paths on the flow chart, for which there are known techniques.

7. **Review of test cases by requirements authors.** The designed test cases are reviewed by the requirements authors. If there is a problem with a test case, the requirements associated with the test case can be corrected and the test cases redesigned.

8. **Validate test cases with the users/domain experts.** If there is a problem with the test case, the requirements associated with it can be corrected and the test case redesigned. Users/domain experts obtain a better understanding of what the deliverable system will be like.

9. **Review of test cases by developers.** The test cases are also reviewed by the developers. By doing so, the developers understand what they are going to be tested on, and obtain a better understanding of what they are to deliver so they can deliver for success.

10. **Use test cases in design review.** The test cases restate the requirements as a series of causes and effects. As a result, the test cases can be used to validate that the design is robust enough to satisfy the requirements. If the design cannot meet the requirements, then either the requirements are infeasible or the design needs rework.

11. **Use test cases in code review.** Each code module must deliver a portion of the requirements. The test cases can be used to validate that each code module delivers what is expected.

12. **Verify code against the test cases derived from requirements.** The final step is to build test cases from the logical test cases that have been designed by adding data and navigation to them, and executing them against the code to compare the actual behaviour to the expected behaviour. Once all of the test cases execute successfully against the code, then it can be said that 100 percent of the functionality has been verified and the code is ready to be delivered into production.

3.2 Measurement in RBT process

Throughout the RBT process, multiple measures can be used to quantify the status of deliverables and activities. This helps managers and process experts oversee quality initiatives across the IT application portfolio.

Examples of information that could be measured include [7]:

- Percent of requirements reviewed by domain experts, designers and coders.
- Percent of requirements that contain ambiguous terms.
- Percent of requirements with formal representation.

- Percent of formal requirements covered by formal test cases.
- Logical and actual code coverage.

3.3 The role of traceability in RBT

Traceability also plays a critical role if using RBT since maintaining traceability information between requirements and logical test-cases and tests is crucial. This information is required for monitoring progress and coverage, as well as properly managing the impact of changes in requirements. Without it, it is more difficult to determine which test cases or tests should be changed if a specific requirement changes.

4. CASE STUDY

4.1 Overview

This case study covers an application, which consists of two web portals (Portal_1 and Portal_2) that share the same database. Complexity of Portal_1 is three times greater than of Portal_2. Being a part of a large system, which was already developed and in use, technology and domain knowledge should not have significant impact on implementation.

After requirements have been written, and accepted (without deep analysis), budget was determined with aspect to estimates and proved formula. With acknowledged time frame limit and budget a team was gathered. Since this was a shared outsourced project, the team was divided into two (a thousand miles separated) groups:

- Team group 1 – 3 members (project manager, requirements manager and architect).
- Team group 2 – 5 full time members (3 developers and 2 testers), 2 part time members (project manager and additional tester).

As soon as careful planning was done, implementation of Portal_1 started. To gain better control over implementation phase a set of milestones was determined. After the milestone was reached the release was deployed and tested to verify that implementation is on the right track.

However, with the first milestones it was obvious that implementation is running behind a schedule. Number of misunderstandings of requirements between team members and requirements changes led to a bunch of rework of code and test cases.

As the end of implementation of Portal_1 was approaching, (but far beyond deadline) change of methodology emerged as a must in order to finish portals on time, with set budget. In order to retake control over project, the decision to introduce parts of requirements based testing methodology was made. Because of the lack of resources, time and knowledge of RBT, only the next steps were introduced in process of software lifecycle (Fig. 4, green boxes):

- Requirements quality steps.
- Logical test case design steps.
- Test execution step (which presents a standard testing activity).

With introduction of RBT methods implementation phase of Portal_2 ended with minor problems, much before estimated time, and the project was finished on time.

4.2 Results

After completing both portals, certain analyses were conducted in order to determine the impact that implementation of RBT methods had on performance and quality. Results of those analyses are displayed in the following figures.

As mentioned above, time estimation for implementation of Portal_1 was breached. After applying some of the RBT methods during the development of Portal_2, number of issues (related to erroneous requirements) was significantly reduced. This led to shorter time of implementation. As a final result the project was finished within the estimated time frame (Fig. 5).

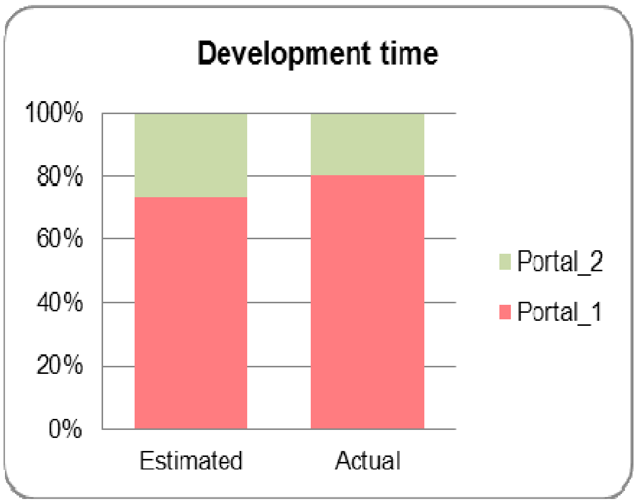
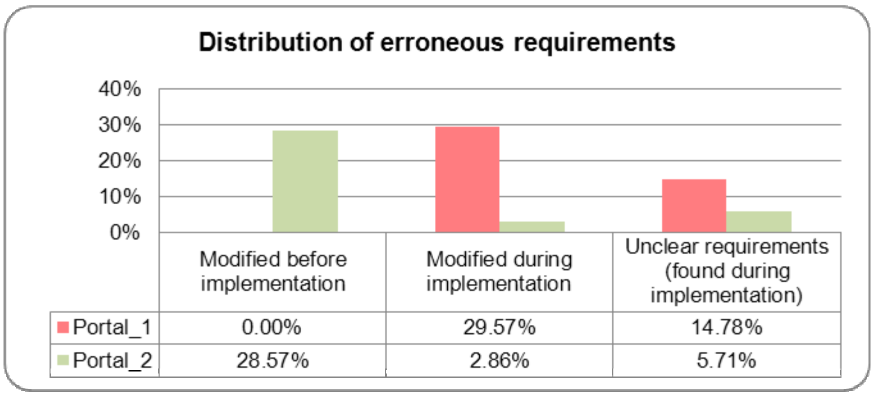


Figure 5. Estimated vs. Actual time of development of Portal_1 and Portal_2

Another statistics consider distribution of erroneous requirements before and during implementation of web portals (Fig. 6):

- Before implementation of Portal_1 requirements specification was not thoroughly tested, so all the problems related to requirements appeared in implementation phase. The data shows that, at the end, there were more than 44% of defected requirements.

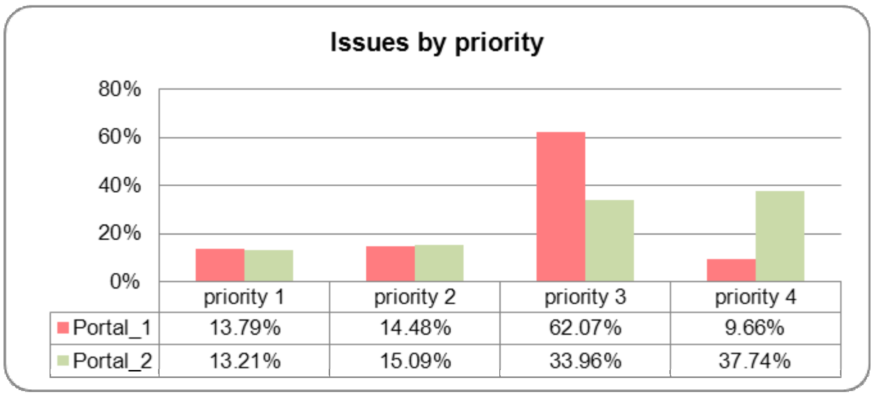
- Regarding detailed analysis of requirements specification, before the implementation of Portal_2, the most of problematic requirements were revealed and fixed. Only small percent of requirements were detected in implementation phase as troublesome.



	Modified before implementation	Modified during implementation	Unclear requirements (found during implementation)
🟥 Portal_1	0.00%	29.57%	14.78%
🟩 Portal_2	28.57%	2.86%	5.71%

Figure 6. Distribution of erroneous requirements

The following analysis addresses the distribution of issues by priority (Fig. 7.).



	priority 1	priority 2	priority 3	priority 4
🟥 Portal_1	13.79%	14.48%	62.07%	9.66%
🟩 Portal_2	13.21%	15.09%	33.96%	37.74%

Figure 7. Distribution of issues by priority

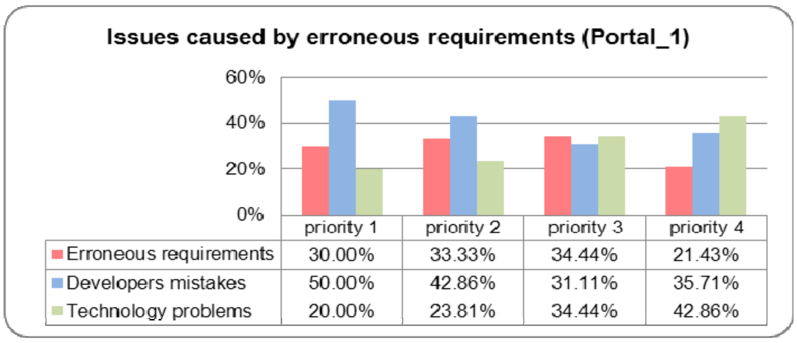
Issues in the project were categorized by priority, as follows:

- Priority 1 – must fix, will be fixed as soon as possible (blocking).
- Priority 2 – should fix, will be fixed before product release (non-blocking).
- Priority 3 – could fix, fix as time and resources allow.
- Priority 4 – would like to fix, but low priority (nice to have).

It is noticed that root cause of priority 3 issues is in the requirements. Figure 7 shows that number of priority 3 issues is significantly reduced.

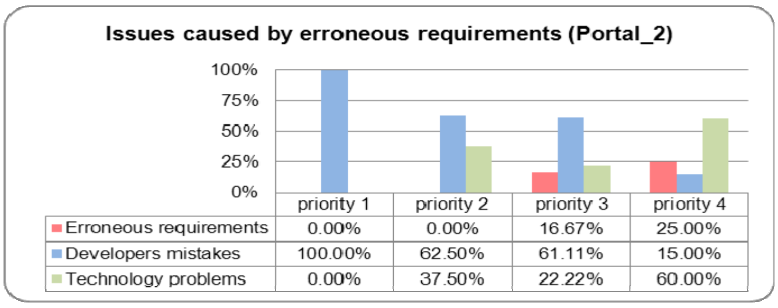
Priority 4 issues relate mostly to usability problems. These were the most registered issues in Portal_2.

Distribution of issues caused by erroneous requirements of Portal_1 and Portal_2 are displayed in Figure 8 and Figure 9, respectively.



	priority 1	priority 2	priority 3	priority 4
🟥 Erroneous requirements	30.00%	33.33%	34.44%	21.43%
🟦 Developers mistakes	50.00%	42.86%	31.11%	35.71%
🟩 Technology problems	20.00%	23.81%	34.44%	42.86%

Figure 8. Distribution of issues caused by erroneous requirements of Portal_1



	priority 1	priority 2	priority 3	priority 4
🟥 Erroneous requirements	0.00%	0.00%	16.67%	25.00%
🟦 Developers mistakes	100.00%	62.50%	61.11%	15.00%
🟩 Technology problems	0.00%	37.50%	22.22%	60.00%

Figure 9. Distribution of issues caused by erroneous requirements of Portal_2

The data shows that there were no priority 1 and priority 2 issues caused by erroneous requirements in Portal_2. Also, number of priority 3 and priority 4 issues related to problematic requirements is significantly lower.

Figure 10 shows that number of faulty test cases in Portal_2 has significantly decreased. Also, it was noticed that almost 20 percent of faulty test cases were fixed several times, as related requirements were changed.

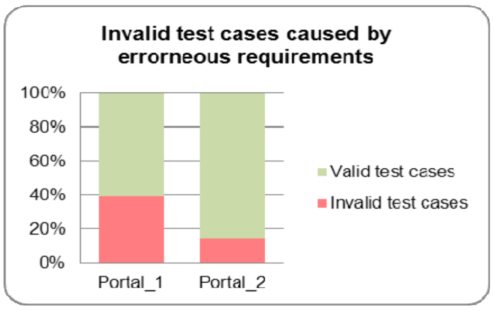


Figure 10. Percent of invalid test cases caused by erroneous requirements

5. CONCLUSION

After analyzing gathered data we have come to next conclusions:

- By introducing RBT methods of testing requirements, before the implementation phase, number of problematic requirements found during the implementation was significantly lower, which resulted in improved stability of code and test cases (less rework) and time saving.
- Most of the issues found in Portal_2 had low priority, and were less effort-demanding and time consuming to be fixed.
- High priority issues (bugs) in Portal_2 were mostly results of developers' negligence. This shows that peer review techniques that developers were using do not produce expected results. By improving this part of the process (peer reviews) it should be possible to increase quality and save time.
- Applying of full RBT process could significantly lower the number of invalid test cases and make developers more aware of testing activities.
- Exhaustive requirements testing demanded constant communication between team groups, located in different countries, which resulted in better understanding and overcoming of cultural differences.

Partial applying of RBT methodology did make improvement in software development process, and made it possible to get it right the first time. Some steps of the RBT process have not been used and statistics shows that there can be positive impact if introduced in process of software development.

6. REFERENCES

[1] Martin, J.,(1984), "*An Information Systems Manifesto"*. http://www.amazon.com/Information-Systems-Manifesto-James-Martin/dp/0134647696/ref=sr_1_1?ie=UTF8&s=books&qid=1233001157&sr=8-1 [15.02.2009]

[2] Borland, (2006), "*Eliminate the Testing Bottleneck"*. http://www.borland.com/us/solutions/lifecycle-quality-management/requirements-based-testing.html [15.02.2009]

[3] Galin D., (2004), "Software Quality Assurance". http://www.amazon.com/Software-Quality-Assurance-Theory-Implementation/dp/0201709457/ref=sr_1_1?ie=UTF8&s=books&qid=1232900529&sr=1-1 [15.02.2009]

[4] Spillner, A., Linz, T., Schaefer, H., (2006), "*Software Testing Foundations"*. http://www.amazon.de/Software-Testing-Foundations-Certified-Foundation/dp/3898643638/ref=sr_1_7?ie=UTF8&s=books-intl-de&qid=1232900648&sr=1-7 [15.02.2009]

[5] Bender RBT Inc., (2003), "*Requirements Based Testing*, Process Overview". http://benderrbt.com/Bender-Requirements Based Testing Process Overview.pdf [15.02.2009]

[6] Mogyorodi, G., (2003), "*What Is Requirements-Based Testing?"*. http://www.stsc.hill.af.mil/crosstalk/2003/03/Mogyorodi.html [15.02.2009]

[7] Aharonovitz, M., (2008), "*Three Tips to Improve Your Requirements-Based Testing* (RBT)". http://www.borland.com/us/company/newsletter/issue5/3tips_improve_rbt.html [15.02.2009]