TESTING: FIRST STEP TOWARDS SOFTWARE QUALITY

Quality is never an accident; it is always the result of high intention, sincere effort, intelligent direction and skillful execution; it represents the wise choice of many alternatives. (William A. Foster)

Ioan Mihnea IACOB
Managing Partner Qualitance QBS, Quality Consulting and Training Services
Bachelor Degree in Computer Science from Polytechnic University, Bucharest, Romania
CMMI 1.1. (Staged and Continuous) – course from Carnegie Mellon Software Institute

E-mail: ioan.iacob@qualitance.ro

Radu CONSTANTINESCU
PhD Candidate, University Assistant Lecturer, Department of Economic Informatics
University of Economics, Bucharest, Romania

E-mail: radu.constantinescu@ie.ase.ro

Abstract: *This article's purpose is to present the benefits of a mature approach to testing and quality activities in an engineering organization, make a case for more education with respect to software quality and testing, and propose a set of goals for quality and testing education.*

Key words: *software testing; course curricula; quality assurance; testing tools*

1. Introduction

IT managers and professionals may have varied opinions about many software development principles, but most agree on one thing above all – the software you deliver must be accurate and reliable. And successful software development groups have long recognized that effective testing is essential to meeting this goal.

In a recent survey of software development managers, the most often cited top-of-mind issue was software testing and quality assurance [Zeich05][1]. Testing is not quality assurance—a brilliantly tested product that was badly conceived, incompetently designed and indifferently programmed will end up a well-tested, bad product. However, software testing has long been one of the core technical activities that can be used to improve the quality of software. Many organizations invest heavily in testing and many software developers work as testers. For example, most Microsoft projects employ one tester per programmer [Cusum98]; [Gates97]

Despite testing's well-established benefits, however, many IT organizations, especially in the Romanian newly-developed software market, still give the practice short shrift, considering thorough testing impractical; as expected from a fairly new software engineering culture, the approach and mentality are development-centric. The limited testing most of the IT organizations do conduct is often haphazard and painfully cumbersome, an effort with little if any gain. Also, most notable is the small, if any at all, presence of the quality-related education in the engineering / computer science schools and universities.

2. Software management, quality assurance, and testing

Before we dive into the details, let's look at where testing fits in the larger world of software management and why it is a valuable practice.

2.1. Software systems context

Software systems are an increasing part of life, from business applications to consumer products. Most people have had an experience with software that did not work as expected. Software that does not work correctly can lead to many problems. We can mention here loss of money, loss of time or business reputation.

A human being can make an error, which produces a fault in the code, in software or a system, or in a document. If a defect in code is executed, the system will fail to do what it should do or even do something it shouldn't, causing a failure. Defects in software, systems or documents may result in failures, but not all defects do so.

Defects occur because human beings make mistakes and also because of time pressure, complex code, complexity of infrastructure, changed technologies, and many system interactions, including environmental conditions

2.2. Software management

Software management is a set of practices that attempt to achieve the following goals:

- Deliver software products with the expected functionality and quality;
- Deliver in the expected timeframe;
- Deliver for the expected cost;
- Meet expected levels of service during the software's use.

Effective software management is a matter of setting and meeting expectations, which requires a software development process that is ***predictable*** and that produces consistent outcomes. Testing is one of the software management practices that helps improve predictability and consistency.

2.3. Quality assurance

The overall goal of QA is to deliver software that minimizes defects and meets specified levels of function, reliability, and performance. Quality Assurance makes sure the project will be completed based on the previously agreed specifications, standards and functionality required without defects and possible problems. It monitors and tries to improve the development process from the beginning of the project to ensure this. A good, healthy quality assurance process should be overall oriented to "prevention".

2.4. Testing general issues

Testing is a collection of techniques used to measure, and thereby improve, software quality. Testing fits in the broader category of software management practices known as quality assurance (QA), along with other practices, such as defect tracking and design and code inspections. Software testing is oriented to "detection" [KaPe02].

A common perception of testing is that it only consists of running tests, i.e. executing the software. This is part of testing, but not all of the testing activities.

Test activities exist before and after test execution, activities such as planning and control, choosing test conditions, designing test cases and checking results, evaluating completion criteria, reporting on the testing process and system under test, and finalizing or closure (e.g. after a test phase has been completed). Testing also includes reviewing of documents (including source code) and static analysis.

Both dynamic testing and static testing can be used as a means for achieving similar objectives, and will provide information in order to improve both the system to be tested, and the development and testing processes.

There can be different test objectives:

- finding defects;
- gaining confidence about the level of quality and providing information;
- preventing defects.

The thought process of designing tests early in the life cycle (verifying the test basis via test design) can help preventing defects from being introduced into code. Reviews of documents (e.g. requirements) also help to prevent defects appearing in the code.

Debugging and testing are different. Testing can show failures that are caused by defects. Debugging is the development activity that identifies the cause of a defect, repairs the code and checks that the defect has been fixed correctly. Subsequent confirmation testing by a tester ensures that the fix does indeed resolve the failure. The responsibility for each activity is very different, i.e. testers test and developers debug.

The most visible part of testing is executing tests. But to be effective and efficient, test plans should also include time to be spent on planning the tests, designing test cases, preparing for execution and evaluating status.

The fundamental test process consists of the following main activities:

- planning and control;
- analysis and design;
- implementation and execution;
- evaluating exit criteria and reporting;
- test closure activities.

Although logically sequential, the activities in the process may overlap or take place concurrently.

2.5. Motivation for testing

By understanding the root causes of defects found in other projects, processes can be improved, which in turn should prevent those defects reoccurring and, as a consequence, improve the quality of future systems.

Effective testing before production deployment achieves three major benefits:

- Discovering defects before an application is deployed allows you to fix them before they impact business operations. This reduces business disruptions from software failure or errors and reduces the cost of fixing the defects.
- You can estimate the extent of remaining, undiscovered defects in software and use such estimates to decide when the software meets reliability criteria for production deployment.
- Test results help you identify strengths and deficiencies in your development processes and make process improvements that improve delivered software.

Considering the first benefit, your own experience probably confirms what software researchers have discovered: The later a "bug" or other defect is found, the more troublesome and expensive it is as is depicted in Figure 1.

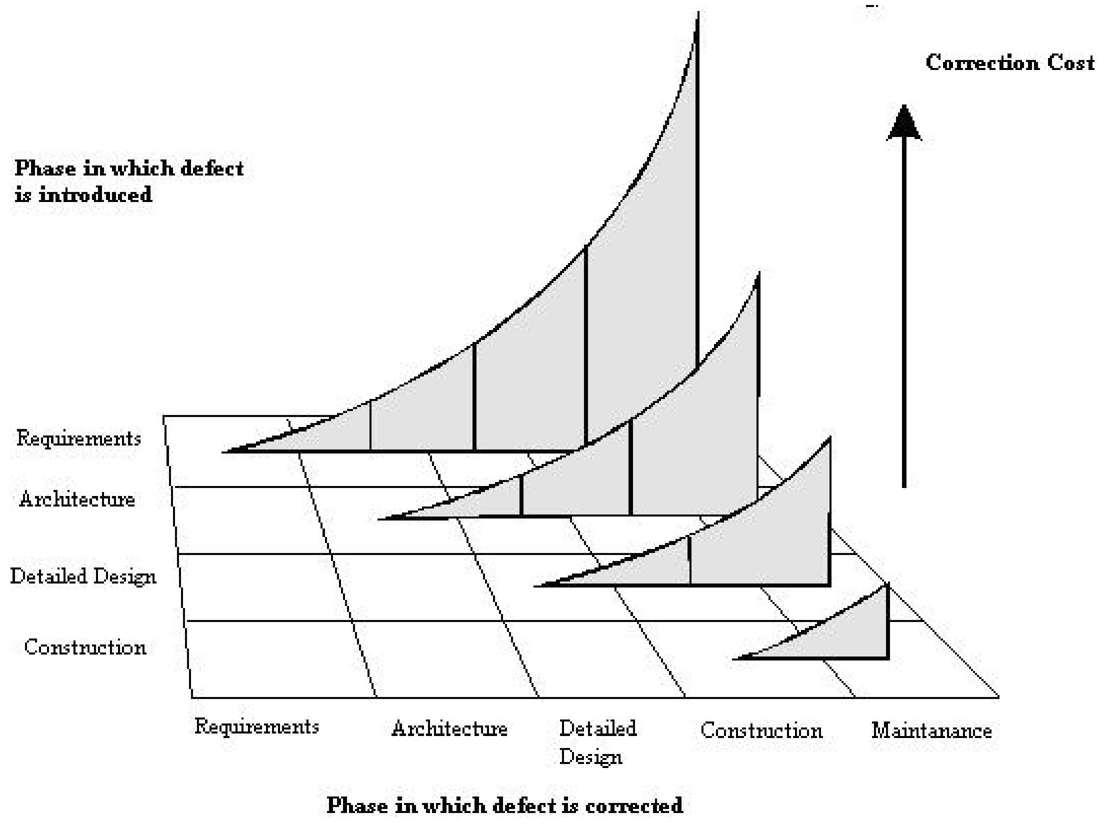

Figure 1. Cost of defects in development phases [McCon97]

Defects in production software can severely disrupt business operations by causing downtime, customer complaints, or errors. Software researchers have found that defects discovered in production can be as much as one hundred times more expensive to fix than those found early in the design and implementation process. Excessive numbers of defects also disrupt the software development process, because the development group "thrashes" as they try to manage the cycle of fixing defects in production code while adding new features or making other planned changes required by the business.

Test execution is just one way to discover defects earlier in the development process. Various studies indicate that well-conducted testing by itself may identify somewhere around 50 or 60 percent of the defects present in an application. Design and code inspections are another technique for discovering defects; and, interestingly, research indicates inspections can achieve error detection rates as high as 90 percent. [Kaner04]; [KaNg93]

With the help of testing, it is possible to measure the quality of software in terms of defects found, for both functional and non-functional software requirements and characteristics (e.g. reliability, usability, efficiency and maintainability).

Thorough testing of systems and documentation can help to reduce the risk of problems occurring in an operational environment and contribute to the quality of the software system, if defects found are corrected before the system is released for operational use.

Another benefit of testing (which applies to code inspections, as well) is that you gain some measures of how "buggy" a piece of software is before you decide to deploy it. For example, if your experience with testing reveals that, on average, about as many defects are found after deployment as are found during testing, then you can project a similar relationship for future projects.

Software testing may also be required to meet contractual or legal requirements, or industry-specific standards.

2.6. Testing principles

A number of testing principles have been suggested over the past 40 years and offer general guidelines common for all testing. These should be the fundament for a healthy testing organization approach. [GrVe06]

a) Principle 1 – Testing shows only presence of defects

Testing can show that defects are present, but cannot prove that there are no defects. Testing reduces the probability of undiscovered defects remaining in the software but, even if no defects are found, it is not a proof of correctness.

b) Principle 2 – Exhaustive testing is impossible

Testing everything (all combinations of inputs and preconditions) is not feasible except for trivial cases. Instead of exhaustive testing, risks and priorities are used to focus testing efforts.

c) Principle 3 – Early testing

Testing activities should start as early as possible in the software or system development life cycle, and should be focused on defined objectives.

d) Principle 4 – Defect clustering

A small number of modules contain most of the defects discovered during pre-release testing, or show the most operational failures.

e) Principle 5 – Pesticide paradox

If the same tests are repeated over and over again, eventually the same set of test cases will no longer find any new bugs. To overcome this "pesticide paradox", the test cases need to be regularly reviewed and revised, and new and different tests need to be written to exercise different parts of the software or system to potentially find more defects.

f) Principle 6 – Testing is context dependent

Testing is done differently in different contexts. For example, safety-critical software is tested differently from an e-commerce site.

g) Principle 7 – Absence-of-errors fallacy

Finding and fixing defects does not help if the system built is unusable and does not fulfill the users' needs and expectations.

3. High-level issues for building an effective quality/testing organization

3.1. Test organization and independence

A critical issue with testing is independence. [ISTQB05] The effectiveness of finding defects by testing and reviews can be improved by using independent testers. Options for independence are:

- Independent testers within the development teams.
- Independent test team or group within the organization, reporting to project management or executive management.
- Independent testers from the business organization, user community and IT.
- Independent test specialists for specific test targets such as usability testers, security testers or certification testers (who certify a software product against standards and regulations).
- Independent testers outsourced or external to the organization.

For large, complex or safety critical projects, it is usually best to have multiple levels of testing, with some or all of the levels done by independent testers. Development staff may participate in testing, especially at the lower levels, but their lack of objectivity often limits their effectiveness. The independent testers may have the authority to require and define test processes and rules, but testers should take on such process-related roles only in the presence of a clear management mandate to do so.

The benefits of independence include:

- Independent testers see other and different defects, and are unbiased.
- An independent tester can verify assumptions people made during specification and implementation of the system.

Drawbacks include:

- Isolation from the development team (if treated as totally independent).
- Independent testers may be the bottleneck as the last checkpoint.
- If proper communication is not insured, developers may lose a sense of responsibility for quality.

3.2. Testing staff education

Even though erratic testing could provide "some" results, delivering quality software cannot be achieved without the testing personnel having the appropriate knowledge for approaching testing in a more scientific way. We should address this topic further in the 3rd section of this paper.

3.3. Tools for testing activities

There are tools available to support many activities in the test processes. Some of them are purely the basis for a well-established testing organization (such as defect tracking tools), others provide improvements in testing efficiency. Irrespective of that, providing the correct set of tools to the testing organization can be critical to its success.

Tools can be classified in according to the testing activities that they support [GrVe06]. Some tools clearly support one activity; others may support more than one activity, but are classified under the activity with which they are most closely associated.

Management tools apply to all test activities over the entire software life cycle:

- Test management tools.
- Requirements management tools.
- Incident management tools.
- Configuration management tools.

The major benefit of **static testing tools** is the cost effectiveness of finding more defects at an earlier time in the development process. As a result, the development process may accelerate and improve by having less rework. Tools to support static testing are:

- Review process support tools.
- Static analysis tools.
- Modeling tools.

Test specification tools support testing activities prior to test execution:

- Test design tools.
- Test data preparation tools.

Probably the broadest set of tools is the category that provides support for the actual testing activities: test execution, monitoring and logging:

- Test execution tools (functional).
- Test harness/unit test framework tools.
- Test comparators.
- Coverage measurement tools.
- Security tools.
- Dynamic analysis tools.
- Performance testing/load testing/stress testing tools.
- Monitoring tools.

Other tools might appear to provide support for **specific application areas**.

4. Proposed education goals

The intent of this section is to pinpoint some very specific information that should be part of any tester's knowledge, information that could be provided, for example, as part of a university curriculum or professional training for testing specialists. The main topics of the course would be:

- Testing fundamentals.
- Testing within the software life cycle context.
- Static testing techniques.
- Test design techniques.
- Test management.
- Tools support for testing.

4.1. Testing fundamentals

A good tester should understand his role in the organization and the responsibility this role assumes. A trained tester should:

- Understand the way in which a defect in software can cause harm to a person, to the environment or to a company.
- Distinguish between the root cause of a defect and its effects.
- Be aware of his role as part of quality assurance.
- Understand how testing contributes to higher quality.
- Understand the terms mistake, defect, failure and corresponding terms error and bug.
- Understand the purpose of testing in software development, maintenance and operations as a means to find defects, provide confidence and information, and ultimately prevent defects.
- Live his professional life by the principles of testing
- Know the fundamental test activities from planning to test closure activities and the main tasks of each test activity:
  - test planning;
  - test control;
  - test analysis and design;
  - test implementation and execution;
  - evaluate exit criteria;
  - reporting;
  - test closure activities.
- Be aware that the success of his job is influenced by psychological factors:
  - clear objectives;
  - a balance of self-testing and independent testing;
  - recognition of courteous communication and feedback on defects.

4.2. Testing within the software life cycle context

Testing does not exist in isolation. Testing activities are related to software development activities. Different approaches to testing are needed for different development life cycle models. In this context, a tester should:

- Understand the relationship between development, test activities and work products in the development life cycle model of the environment, product characteristics and context.
- Know and use test levels:
  - Component testing.
  - Integration testing.
  - Systems testing.
  - Acceptance testing (user, operational, contract, field).
- Know and use the four testing types:
  - functional,
  - non-functional,
  - structural,
  - change-related
- Apply functional and structural tests occur at any test level.

- Identify non-functional test types based on non-functional requirements.
- Identify test types based on the analysis of a software system's structure or architecture.
- Understand the purpose of confirmation testing and regression testing
- Make the difference between maintenance testing (testing an existing system) and testing a new application with respect to test types, triggers for testing and amount of testing.
- Identify reasons for maintenance testing (modification, migration and retirement) and apply this in his job.
- Know the role of regression testing and impact analysis in maintenance.

4.3. Static testing techniques

Static testing techniques do not execute the software that is being tested; they are manual (reviews) or automated (static analysis). A test engineer should:

- Apply the different static techniques to examine software work products, using specific phases, roles and responsibilities, and success factors for each type of technique:
  - informal review;
  - technical review;
  - walkthrough;
  - inspection.
- Be aware of the importance and value of considering static techniques for the assessment of software work products.
- Understand the objective of static analysis.
- Be familiar with typical defects and errors identified by static analysis
- Acknowledge both typical and organization specific benefits of static analysis.
- Know typical code and design defects that may be identified by static analysis tools.

4.4. Test design techniques

The purpose of a test design technique is to identify test conditions and test cases. Accumulated knowledge should allow a test engineer to:

- Identifying test conditions and design test cases.
- Know and apply concepts like: test design specification, test case specification and test procedure specification.
- Write test cases:
  - showing a clear traceability to the requirements;
  - containing an expected result.
- Translate test cases into a well-structured test procedure specification at a level of detail relevant to the knowledge of the testers.
- Write a test execution schedule for a given set of test cases, considering prioritization, and technical and logical dependencies.
- Be familiar with categories of test design techniques.
- Recall reasons that both specification-based (black-box) and structure-based (white-box) approaches to test case design are useful, and be aware of the common techniques for each.

- Know the characteristics and differences between specification-based testing, structure based testing and experience-based testing.
- Be familiar with and apply specification-based or black-box techniques
- Write test cases from given software models using test design techniques like:
    - equivalence partitioning;
    - boundary value analysis;
    - decision tables;
    - state transition diagrams.
- Understand the main purpose of each of the four techniques, what level and type of testing could use the technique, and how coverage may be measured.
- Understand the concept of use case testing and its benefits.
- Be familiar with and apply structure-based or white-box techniques
- Understand the concept and importance of code coverage.
- Understand the concepts of statement and decision coverage, and understand that these concepts can also be used at other test levels than component testing (e.g. on business procedures at system level).
- Write test cases from given control flows using test design techniques like:
    - statement testing;
    - decision testing.
- Assess statement and decision coverage for completeness.
- Apply experience-based techniques
- Be able to write test cases based on intuition, experience and knowledge about common defects, and specific areas where defects are prone to appear.
- Be able to make a decision choosing test techniques
- Identify the factors that influence the selection of the appropriate test design technique for a particular kind of problem, such as the type of system, risk, customer requirements, models for use case modeling, requirements models or tester knowledge.

4.5. Test management

Due to the more independent nature of the testing activities, issues of test management should be well known to a member of the testing team [Black02]; [Black07], who should:

- Show proficiency in test organization:
    - Understand the importance of, and advocate for independent testing.
    - Understand the benefits and drawbacks of independent testing within an organization.
    - Recognize the different team members to be considered for the creation of a test team.
    - Recall the tasks of typical test leader and tester.
- Display good skills in test planning and estimation
    - Use the different levels and objectives of test planning.
    - Be familiar with the purpose and content of the test plan, test design specification and test procedure documents according to the 'Standard for Software Test Documentation' (IEEE829).
    - Be aware of the typical factors that influence the testing-related effort.

  - Be familiar with the two conceptually different estimation approaches: the metrics-based approach and the expert-based approach.
  - Be able to devise, or at least be familiar with various documents types, such as test planning for a project, for individual test levels (e.g. system test) or specific test targets (e.g. usability test), and for test execution.
  - Know the test preparation and execution tasks that need planning.
  - Use and justify adequate exit criteria for specific test levels and groups of test cases.
- Be able to run test progress monitoring and control activities:
  - Use common metrics for monitoring test preparation and execution.
  - Understand and interpret test metrics for test reporting and test control (e.g. defects found and fixed, and tests passed and failed).
  - Be familiar with the purpose and content of the test summary report document according to the 'Standard for Software Test Documentation' (IEEE 829) or organization specific standard.
- Understand how configuration management interacts with and supports testing.
- Be able to take a risk-based approach to testing
  - Understand risks as a possible problem that would threaten the achievement of one or more stakeholders' project objectives.
  - Prioritize risks by likelihood (of happening) and impact (harm resulting if it does happen).
  - Distinguish between the project and product risks.
  - Identify typical product and project risks.
  - Use risk analysis and risk management for test planning.
- Write incident reports covering the observation of a failure during testing.
- Use the 'Standard for Software Test Documentation' (IEEE 829) incident report or organization specific standard.

4.6. Tools support for testing

There are a number of tools that support different aspects of testing. A proficient member of the testing organization should be able to:

- Be familiar with the different types of test tools according to the test process activities.
- Know the tools that may help developers in their testing.
- Understand the potential benefits and risks of test automation and tool support for testing.
- Recognize that test execution tools can have different scripting techniques, including data driven and keyword driven.
- Know and understand the main principles of introducing a tool into an organization.
  - Use the goals of a proof-of-concept/piloting phase for tool evaluation.
  - Identify factors (other than simply acquiring a tool) that are required for good tool support.

5. Conclusions

Efficiency and quality are best served by approaching testing activities in a structured and scientific way, instead of the, unfortunately, usual 'monkey-testing'. The effectiveness of testing effort can be maximized by selection of appropriate testing strategy, good management of testing process, and appropriate use of tools to support the testing process. The net result would be an increase in the produced software quality and a decrease in costs, both of which can only be beneficial to a software development organization.

However, in order to be able to put quality processes into place, the appropriate knowledge is needed. The quality engineering should be recognized as a standalone area of study and treated as such in the computer science universities and faculties curricula in the emergent software development market that Romania is. Taking the right path can never be too early; it can only be too late.

References

1. Black, R. **Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing**, Wiley, 2002
2. Black, R. **Critical Testing Processes: Plan, Prepare, Perform, Perfect**, Addison-Wesley, 2003
3. Black, R. **Pragmatic Software Testing: Becoming an Effective and Efficient Test Professional**, Wiley, 2007
4. Cusumano, M. A. and Selby, R. W. **Microsoft Secrets**, Touchstone, New York, Free Press, 1998.
5. Gates, B. **Remarks by Bill Gates of Microsoft Corporation**, Gartner Symposium, January 16, 2006, www.oasis-open.org/cover/gates-gartnerXML.html
6. Graham, D., Veenendaal, E., Evans, I. and Black, R. **Foundations of Software Testing: ISTQB Certification**, CENGAGE Lrng Business Press, 2006
7. Kaner, C., Bach, J. and Pettichord, B. **Lessons Learned in Software Testing**, John Wiley & Sons, 2002
8. Kaner, C., Falk, J. and Nguyen, H.O. **Testing Computer Software (2nd Ed.)**, International Thomson Computer Press, 1993
9. Kaner, C. **The Ongoing Revolution in Software Testing**, Software Test & Performance Conference, Baltimore, MD, December 7-9, 2004
10. Kaner, C. **Curricular support for software testing**, Proposal to the National Science Foundation
11. McConnell, S. **Upstream Decisions, Downstream Costs**, Windows Tech Journal, November 1997
12. Zeichick, A. **Quality is Hot, Hi-B Visas are Not**, SD Times: The Industry Newspaper for Software Development Managers, 5., 2005, http://www.sdtimes.com/article/story-20050401-04.html
13. * * * **Certified Tester - Foundation Level Syllabus**, International Software Testing Qualifications Board, Version 2005

[1] Codification of references:

[Black02]	Black, R. **Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing**, Wiley, 2002
[Black03]	Black, R. **Critical Testing Processes: Plan, Prepare, Perform, Perfect**, Addison-Wesley, 2003
[Black07]	Black, R. **Pragmatic Software Testing: Becoming an Effective and Efficient Test Professional**, Wiley, 2007
[Cusum98]	Cusumano, M. A. and Selby, R. W. **Microsoft Secrets**, Touchstone, New York, Free Press, 1998

[Gates97]	Gates, B. **Remarks by Bill Gates of Microsoft Corporation**, Gartner Symposium, January 16, 2006, www.oasis-open.org/cover/gates-gartnerXML.html
[GrVe06]	Graham, D., Veenendaal, E., Evans, I. and Black, R. **Foundations of Software Testing: ISTQB Certification**, CENGAGE Lrng Business Press, 2006
[KaPe02]	Kaner, C., Bach, J. and Pettichord, B. **Lessons Learned in Software Testing**, John Wiley & Sons, 2002
[KaNg93]	Kaner, C., Falk, J. and Nguyen, H.O. **Testing Computer Software (2nd Ed.)**, International Thomson Computer Press, 1993
[Kaner04]	Kaner, C. **The Ongoing Revolution in Software Testing**, Software Test & Performance Conference, Baltimore, MD, December 7-9, 2004
[Kaner06]	Kaner, C. **Curricular support for software testing**, Proposal to the National Science Foundation
[McCon97]	McConnell, S. **Upstream Decisions, Downstream Costs**, Windows Tech Journal, November 1997
[Zeich05]	Zeichick, A. **Quality is Hot, Hi-B Visas are Not**, SD Times: The Industry Newspaper for Software Development Managers, 5., 2005, http://www.sdtimes.com/article/story-20050401-04.html
[ISTQB05]	* * * **Certified Tester - Foundation Level Syllabus**, International Software Testing Qualifications Board, Version 2005