MYKOLAS ROMERIS UNIVERSITY

BUSINESS AND MEDIA SCHOOL

BRIGITA JAZUKEVIČIŪTĖ

(Business Informatics)

EFFECTIVENESS OF SOFTWARE TESTING TECHNIQUES IN ENTERPRISE: A CASE STUDY

Master Thesis

Supervisor –

Assoc. Prof. Andrej Vlasenko

Vilnius, 2016

CONTENTS

THE LIST OF FIGURES

THE LIST OF TABLES

THE LIST OF ANNEXES

ABBREVIATIONS

IT - Information Technology

ICT - Information Communication Technology

IEEE - Institute of Electrical and Electronics Engineers

ISO - International Organization for Standardization

QA - Quality Assurance

SDLC - Software Development Life Cycle

BRS - Business Requirement Specification

SRS - System Requirement Specifications

INTRODUCTION

Relevance of the topic. Since the use of computers increased the significance of software testing has gained more mainstream attention from information technology (hereinafter - IT) professionals. (Gelperin & Hetzel, 1988). From an IT perspective, innovations made software developers to respond to each emerging technology quicker; thus it negatively impacted software quality. Galin (2004) explains that software quality has a direct relationship with software testing; hence testing is an important phase of the software development life cycle. Enterprises expenditure for testing takes a quite big part of all software development budget: 50 % in 1979, 24% in 2006, 18% in 2012, 23% in 2013, 26% in 2014, 35% in 2015, and 31% in 2016 (Hans van Waayenburg & Raffi Margaliot, 2016; Myers, Sandler, & Badgett, 2011; Perry, 2006). From 1979 to 2012, the significant decrease of expenditure towards software testing is observed, whilst from 2013 to 2016 the numbers are quite the opposite. However, this stabilization period may depend on various factors, including the size and the complexity of the software development projects, labor costs, and overall upkeep of information communication technology (hereinafter - ICT) infrastructure.

Software testing is a discipline which by essence is not very complex or difficult to implement. However, it may be costly and demanding in human effort or in technology which multiplies it (Hambling & Morgan, 2011). Software testing is often considered as a routine and low-level task. Despite these unjustified presumptions, it is a critical part of software development process determining the efficiency or even correctness of final product that is tended to be free of serious defects. Indeed, software testing faces a collection of challenges (Bertolino, 2007; Hans van Waayenburg & Raffi Margaliot, 2016): with the complexity, pervasiveness and criticality of software growing; identification of the right areas on which to test, the realization of benefits from automation; the lack of skills in the areas of intelligence-driven testing strategies and newer test automation skills, tight control of budgets, selection of the right testing techniques, types etc. Therefore, ensuring that software behaves according to the desired levels of quality and dependability becomes more crucial, increasingly difficult, and expensive.

The level of investigation. The investigation of the need of software testing and QA varies from the analysis of concepts and definitions (“IEEE Standard Glossary of Software Engineering Terminology,” 1990; McCall, Richards, & Walters, 1977) to efficiency of software testing methods, software testing tools and automation testing.

Researchers analyzed the importance of both terms, software testing and quality assurance (DeVolder, Ghazanshahi, & Zadeh, 2008; Garvin, 1984; Kitchenham & Lawrence, 1996), by giving some background on software testing principles and testing terminology. Other studies focus on the effectiveness and use of following testing techniques: almost all White-box and Black-box techniques

(Jorgensen, 2016; M. E. Khan, 2011a, 2011b; Mohd Ehmer Khan & Khan, 2012; Nidhra & Dondeti, 2012; Saglietti, Oster, & Pinte, 2008), static and dynamic testing and their tools (DeVolder et al., 2008; Emanuelsson & Nilsson, 2008; Ernst, 2003; Fagan, 2001; Hamlet, 1995; Jorgensen, 2016; Nidhra & Dondeti, 2012; Zitser, Lippmann, & Leek, 2004), functional testing (DeVolder et al., 2008; Hamlet, 1995; M. E. Khan, 2011a; Mohd Ehmer Khan & Khan, 2012; Liu & Kuan Tan, 2009), unit testing (Di Tommaso & Roche, 2011; Hamlet, 1995; Williams, Kudrjavets, & Nagappan, 2009), system testing (Hamlet, 1995),

There are more studies conducted to determine how improve software testing effectively and efficiently, such as the one by Bertolino (2007), Glass, Collard, Bertolino, Bach, & Kaner (2006), Vegas, Juristo, & Basili (2002) and Juristo, Moreno, & Strigel (2006). Additionally, similar empirical studies are prepared by Ng, Murnane, Reed, Grant, & Chen, (2004), Causevic, Sundmark, & Punnekkat (2010) and Lee, Kang, & Lee (2012), Ng et al. (2004). Other authors distinguish particular testing techniques, such as unit testing (Di Tommaso & Roche, 2011; Williams et al., 2009), regression testing (Elbaum, Malishevsky, & Rothermel, 2002; Li, Harman, & Hierons, 2007; Org, 2012; Rothermel, Untch, Chu, & Harrold, 1999; Srivastava, 2008; Wong, Horgan, London, & Agrawal, 1997), functional testing (Popescu, 2010) and analyze their use and effectiveness. Moreover, software testing methods can be automated fully or partially in order to shorten the period of software development. There are considerations on what processes should be automated or be aware of automation (Garousi & Mäntylä, 2016, D. M. Rafi, Moses, Petersen, & Mäntylä, 2016; Mulder & Whyte, 2013). Although, other authors present different approaches to automate: detection of infeasible paths in software testing to the extent of test coverage (Gong & Yao, 2010), some testing techniques (unit testing, functional testing, regression testing and performance testing (Williams, Kudrjavets, & Nagappan, 2009; Cem Kaner, n.d.; Last, Friedman, & Kandel, 2004). And Kasurinen, Taipale, & Smolander (2010) observe the practices in software test automation and identified factors that affect software test automation.

However, most of the mentioned studies are based on theoretical aspects, except some surveys (Causevic et al., 2010; Kasurinen et al., 2010; Lee et al., 2012; Ng et al., 2004) that focus on empirical data collected via a case study of industrial software development companies in different business sectors. Only Causevic et al. (2010), Kasurinen et al. (2010) and Lee et al. (2012) of all mentioned researchers provide both, qualitative and quantitative, data analysis from an industrial questionnaire survey, with a focus on current practices and preferences on contemporary aspects of software testing.

Novelty of the topic. The current software testing practices are far from satisfactory (Bertolino, 2007; Glass et al., 2006; Juristo et al., 2006). The researches (Hans van Waayenburg & Raffi Margaliot, 2016; Juristo et al., 2006; Vegas et al., 2002) argue that there are still needs for sophisticated tools and there are gaps between testing research and industry practices. The barriers to adoption of software testing methods and tools in terms of capabilities, limitations, improvements and needs are not revealed clearly in practice. Identification of what capabilities should be enhanced is essential in order to ensure the efficiency and effectiveness of testing (Lee et al., 2012). In our opinion, those barriers may lead to inappropriate use of some techniques, it means that some techniques could be misused, others - never used or only a few of them could be applied again and again. Thus, our research will try to answer similar research questions collecting data on the way of an industrial case study. Moreover, according to our level of investigation, we are tend not to analyze the discrepancies observed between the current practices and the perceptions (including beliefs and attitudes) of respondents (it is examined in the research (Causevic et al., 2010)). Our main focus is on observation of software testing techniques practices in a one specified sector (not like the researches (Causevic et al., 2010; Lee et al., 2012; Ng et al., 2004) that examine a variety of different sectors and different enterprises) and to look into the important organizational aspects (e.g. when some techniques are used only by other teams, affiliates or certain team members) as well.

The research problem. The effectiveness of software testing techniques. There are various software testing techniques, including test design techniques and testing tools used in enterprises, but the advantages of using one testing technique as opposed to another in a given situation are unclear. Finding a valuable way to perform more effective testing is a key challenge in software testing because of few aspects: a) the information about testing techniques are distributed across many sources; b) the vast array of programming languages, operating systems, and hardware platforms make software testing more difficult; c) digital transformation in every day influences the complexity of software as well. We believe that there are still gaps on effective software testing in general because of the listed reasons before.

The object of the research - Software testing in quality assurance process at specific enterprise.

Subject - The techniques of software testing.

The purpose of the research. To investigate the use of software testing techniques in terms of limitations and improvements in software quality assurance process at specific enterprise.

The objectives are enumerated in order to achieve the purpose:

1. To explore quality assurance process and identify the relationship between software testing and quality assurance by generalizing scientific literature analysis.

2. To provide a comprehensive view on the main features of software testing techniques by examining theoretical studies and empirical studies of the best practices.

3. To prepare a theoretical framework for conducting a case study for software testing techniques within a specific enterprise.

4. To explore and define the most problematic areas and potential improvements in software testing process by generalizing results of case study and enterprise statistical documents.

Methods of the research: Theoretical methods: comparison and contrast, generalization, abstraction, analogy, modeling, scientific literature review. Empirical methods: case study based on expert interviews and quantitative statistical document analysis.

Structure of research. The research consists of four chapters each of them analyses the objectives provided above. The first chapter discusses software testing fundamentals and quality assurance activities, analyze how these concepts are related with each other. Some background is given on concepts analysis, quality assurance activities, the differences between validation and verification, brief history of software testing, the types of error and defect and main principles of testing. Second chapter presents the software testing techniques, including static and dynamic testing as a code analysis, test design based methods to create test cases, software testing levels that are analyzed as a stage of software development. Test execution types, including manual and automated testing, are analyzed as well as their benefits. Further, the practical use of software testing techniques in enterprises is examined distinguishing the benefits and limitations they are facing during testing. In third chapter, the methodology of case study research is presented. Forth chapter identifies the main software testing techniques used in the selected organization by analyzing qualitative research results conducted by interviewing experts. It also provides the problematic issues for an effective software testing by generalizing qualitative research results and the statistical document analysis of the selected enterprise.

1. THE RELATIONSHIP BETWEEN SOFTWARE TESTING AND SOFTWARE QUALITY ASSURANCE

The first chapter discusses software testing fundamentals and quality assurance activities, analyze how these concepts are related with each other. The purposes of them are discussed by giving some background on concepts analysis, quality assurance activities, the differences between validation and verification, brief history of software testing and main principles of testing. The types of error and defect in the SDLC are illustrated in a Figure as well.

1.1. Introduction to Software Quality Assurance

In order to understand software testing activities and later identify software testing techniques, it is essential to determine the relationship between software quality and software testing. Regarding the activities that comprise software quality, quality assurance activities, quality factors and criteria are analyzed in this subchapter. Additionally, an introduction to the concepts of software quality and the separation between defect types are made as well.

The concept and definitions of software quality in terms of quality factors and quality criteria and metrics was first introduced by McCall et al. (1977) and later developed by others authors (Crosby, 1979; Garvin, 1984; "IEEE Standard Glossary of Software Engineering Terminology," 1990; Juran, 1988; Pressman, 2000). This approach is presented as a foundational guide to answer the following questions: "How do you ensure that all of the software you produce does what it was designed to do and, just as important, does not do what it isn't supposed to do?" (Myers et al., 2011, p. 3). Indeed, software quality is a complex concept which can be constricted or broadened. The concept can be interpreted in different ways by different people, and it is highly context dependent; thus, the deep analysis of concepts should be made in order to understand the relationship between software quality and software testing. Garvin (1984) identified five different views of quality that are presented in a table below (see Table 1, page 11) with other concepts of software quality defined by different authors.

Table 1. The collection of concepts of Software Quality

Definition	Provided by author
A general term applicable to any trait or characteristic, whether individual or generic, a distinguishing attribute which indicates a degree of excellence or identifies the basic nature of something.	(McCall et al., 1977)
Quality means conformance to requirements.	(Crosby, 1979)
Transcendental View: quality is something that can be recognized through	(Garvin, 1984)

experience is not defined in some tractable form.	
User View: quality is as fitness for purpose; the evaluation to which a product satisfy user needs and expectations.	
Manufacturing View: quality is seen as conformance to requirements.	
Product View: quality is seen as tied to the inherent characteristics of the product.	
Value-Based View: quality depends on the amount a customer is willing to pay for it.	
Quality consists of those product features which meet the needs of customers and thereby provide product satisfaction.	(Juran, 1988)
Quality consists of freedom from deficiencies.	
The degree to which a system, component, or process meets specified requirements.	(“IEEE Standard Glossary of Software Engineering Terminology,” 1990)
The degree to which a system, component, or process meets customer or user needs or expectations.	
Software quality is defined as: Conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software.	(Pressman, 2000)

Prepared by author.

The research by Kitchenham & Lawrence (1996) showed that User and Manufacturing views are more important comparing with other Garvin's views. Indeed, Manufacturing view is inherited from Crosby (1979) and later developed as well as User view which defines the main idea of software quality. On the other hand, the more extended term we intend to use. Juran (1988) suggested term fits to describe user satisfactions, but the important part, requirements, is missing. Pressman (2000) notes software as "professionally developed software", nevertheless it is not clear what criteria determine professionally of software in this particular case. Regarding the arguments discussed before, we choose to adopt the term by “IEEE Standard Glossary of Software Engineering Terminology” (1990) as "Software quality - the degree to which a software meets specified requirements and user needs or expectations".

Furthermore, quality issues as a part of quality assurance process have been analyzed by Kitchenham & Lawrence (1996) as well. Quality assurance (hereinafter - QA) is defined as a set of planned activities with the purpose of providing an adequate confidence that a software conforms to established technical requirements (“IEEE Standard Glossary of Software Engineering Terminology,” 1990). Therefore, the provided quality issues that stand for QA have been ranked by respondents in terms of importance respectively:

1. specifying quality requirements objectively;
2. setting up a quality-management system;

3. achieving operational quality that meets requirements;
4. measuring quality achievements;
5. agreeing with the customer on what quality means.

In fact, these quality issues depend on context of software complexity, user needs and expectations, and they can vary in a different order. This aspect was investigated by McCall et al. (1977) as a guideline in how to objectively specify the desired amount of quality at the system requirements specification phase and reduce the cost of software development. The research determines the need of quality factors which jointly comprise software quality, identification of a set of criteria for each factor and application of required metrics for each criterion. Quality factors have been introduced and grouped into few categories by different authors: the classic model of software quality factors, suggested by McCall et al. (1977), consists of 11 factors, very similar models, consisting of 12 to 15 factors, were suggested by Deutsch & Willis (1988) and Evans & Marciniak (1987). Quality factors by McCall et al. (1977) are grouped into three broad categories, such as product operation, product revision, product transition in order to distinguish the relationship between quality factors and software development activities. The elements and their definitions ("IEEE Standard Glossary of Software Engineering Terminology," 1990) of each quality category are illustrated in a table below (see Table 2, page 13).

Table 2. Categorization of Quality factors

Quality Categories	**Quality Factors**	**Definition of Quality Factor**
Product Operation	Correctness	The degree to which a software meets its specifications and fulfills the user's needs and expectations
	Reliability	The ability of a system to perform its required functions under stated conditions for a specified period of time
	Efficiency	The degree to which a system performs its designated functions with minimum consumption of resources
	Integrity	The degree to which a system prevents unauthorized access to, or modification of, computer programs or data
	Usability	The ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system
Product revision	Maintainability	The ease with which a software system can be modified to correct faults, improve performance or adapt to a changed environment.
	Testability	The degree to which a system facilitates the establishment of test criteria and the performance of tests
	Flexibility	The ease with which a system can be modified for use in applications

		or environments other than those for which it was specifically designed
Product transition	Portability	The ease with which a system can be transferred from one hardware or software environment to another
	Reusability	The degree to which a software module or other work product can be used in more than one computer program or software system
	Interoperability	The ability of two or more systems to exchange information and to use the information that has been exchanged

Prepared by author according to Sources: ("IEEE Standard Glossary of Software Engineering Terminology," 1990; McCall et al., 1977)

According to Naik & Tripathy (2008), the main focus of quality factors categorization is on expectations of software post-development activities compared with in-development activities. However, we think that quality factors have the same importance in both, expectations of software post-development activities and in-development activities. Concerning the fact that "The quality assurance activities must start early and become an integrated part of the entire development project" (Hass, 2008), we presume that quality factors are conformed in development phase and it leads to expectations of post-production activities as well. Conformation of quality factors are ensured by the following QA activities:

- **validation** - the process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements ("IEEE Standard Glossary of Software Engineering Terminology," 1990). In contrast, Hass (2008) emphasizes the validation as the evaluation of correctness of the system which involves user's satisfaction as well. It other words, when the requirements are agreed and approved by a contract between two interested parties, an enterprise of software development and a user (as a client the mentioned enterprise), the following issues should be ensured during the entire development life cycle: 1) all elements are implemented that were asked by user; 2) none of undefined elements in the contract have been implemented.
- **verification** - the process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase ("IEEE Standard Glossary of Software Engineering Terminology," 1990).

Thus, based on the terms defined above, it may be concluded that quality factors are validated and verified in-development activities as well.

The differences between validation and verification (see Figure 1, page 15) can be distinguished like this: validation confirms that all elements of software have been designed and implemented by user needs (defined requirements specification), and verification confirms that all elements of software are being developed in the right way and the software works as it supposed to work.

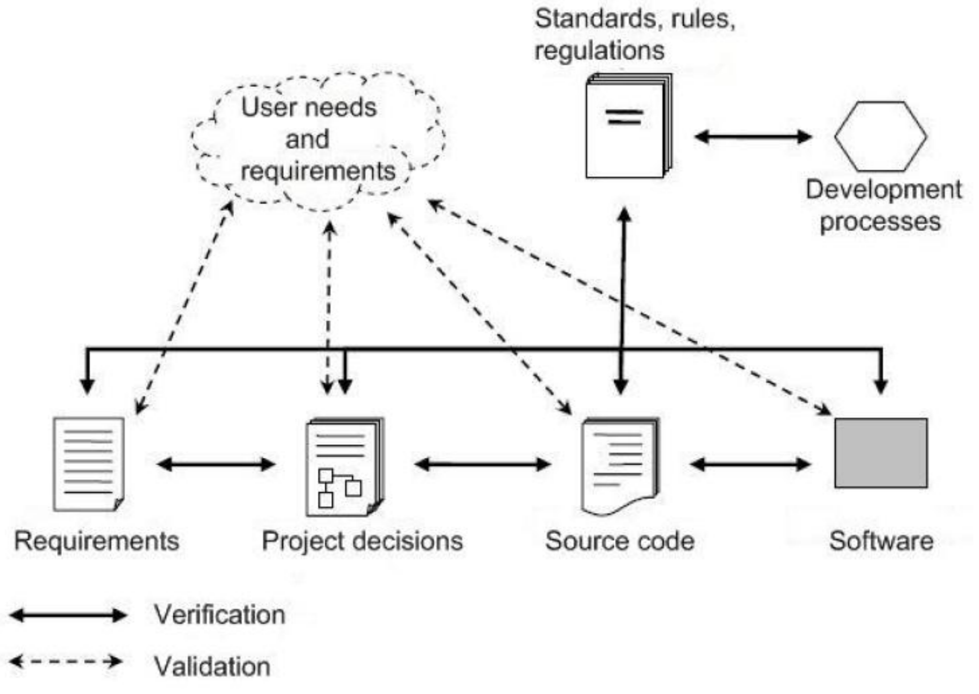


Figure 1. Correspondence between verification and validation

Source: (Uspenskiy, 2010)

According to Hass (2008), verification should be performed after validation, but in some cases such software QA activities can be done in parallel. In respect to that, an uncommon situation can be identified - validation activity is failed while verification is passed (Graham, Veenendaal, & Evans, 2008). After software is verified on the paper (no running or functional application is required) it gets "passed" status. When the same verified points are actually developed, then the running application or product can fail while validation. This particular situation is caused by non-compliance of software specifications with the user requirements. In order to avoid such situations, more attention should be paid on the early phases of software development life cycle (hereinafter - SDLC) .

SDLC determines the period of time that begins when a software product is conceived and ends when the software is no longer available for use (“IEEE Standard Glossary of Software Engineering Terminology,” 1990). Typically SDLC includes the following six phases (Graham et al., 2008):

1. Requirement gathering and analysis - Business requirements are gathered according user needs and expectations. Validation of requirements is performed after requirements gathering.
2. Design - it is prepared from the requirement specifications and specifies hardware and system requirements and overall system architecture.
3. Implementation or coding - coding phase which starts after the work is divided in modules or units by system design documents.

4. Testing - the process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component ("IEEE Standard Glossary of Software Engineering Terminology," 1990).
5. Deployment - the product is delivered to the customer after testing is finished and no critical failures have been identified.
6. Maintenance - the process where modifying software, correcting faults, improving performance are taken for the developed product.

Each phase determines different activity that has a specific role in a SDLC, however we will focus more on requirements gathering and testing parts in order to examine testing techniques in further chapters. Thus, the first phase determines the presence of validation activity. Concluding the previous point that "non-compliance of software specifications with the user requirements was validated as failed", it can be confirmed as the activity of first SDLC phase and more attention should be paid on it. Whereas, verification can be ensured by different software testing techniques called as static testing or static analysis (Hass, 2008) which evaluates a system or component based on its form, structure, content, or documentation; it relies more on visual examination of development products to detect errors, violations of development standards, and other problems ("IEEE Standard Glossary of Software Engineering Terminology," 1990). Static testing process starts early in the SDLC once some code has been implemented. It is done without executing the program, but verifying a bundle of code according to coding conventions, consistency of code unit (module or interface), requirements specification and technical design documents. The main advantage of static testing - more defects are found during static testing, less cost of software development is expected; as defects are detected at the early stage of SDLC when the rework cost most often relatively low (Graham et al., 2008). There are few techniques of static testing that are discussed in the subchapter 2.1.1. In contrast, dynamic testing requires an execution of software which is done during validation process (Hass, 2008). It is noted that validation could be used in the first phase of SDLC (discussed before). On the other hand, in this particular case validation process is done after the code of specified unit (module or interface) is fully implemented in a system. Therefore, dynamic testing is performed after the third phase of SDLC. This technique reveals faults that are very difficult to identify in software: it could be the lack of code in a unit or component, memory leaks, pointer arithmetic errors such as null pointers, identification of time dependencies between different components or code iterations (Graham et al., 2008). The variety of dynamic testing techniques is presented in the subchapter 2.1.2. The SDLC involves all testing techniques in a different phases or done in parallel that affects software cost. Hence, all defects found at very early stage of SDLC during validation or verification can be resolved at that moment without impacting the other piece of code. As a result, it can significantly reduce the impact on SDLC cost and schedule. In order to conclude the purpose of validation and verification for SDLC, the main advantages of them are distinguished by (Graham et al., 2008) and illustrated in a table below (see Table 3, page 17).

Table 3. The advantages of Validation and Verification

Validation	Verification
During verification if some defects are missed then during validation process it can be caught as failures.	Verification helps in lowering down the count of the defect in the later stages of development.
If during verification some specification is misunderstood and development had happened then during validation process while executing that functionality the difference between the actual result and expected result can be understood	Verifying the product at the starting phase of the development will help in understanding the product in a better way.
Validation is done during testing like feature testing, integration testing, system testing, load testing, compatibility testing, stress testing, etc.	It reduces the chances of failures in the software application or product.
Validation helps in building the right product as per the customer's requirement and helps in satisfying their needs.	It helps in building the product as per the customer specifications and needs.

Prepared by author according to Source: (Graham et al., 2008)

Besides validation and verification activities there are few more QA activities, such as definition of quality criteria and quality reporting (Hass, 2008). Validation and verification and their purpose have been discussed already. Although, their role in the process of QA activities remains uncovered. As the figure (see Figure 1, page 17) below shows, each of activity has a specified order in a process. Validation can be performed in parallel with verification as it was noted before.

$$\begin{tikzcd}\boxed{\textbf{Definition of Quality Criteria}} \arrow[r] & \boxed{\textbf{Validation}} \arrow[r] & \boxed{\textbf{Verification}} \arrow[r] & \boxed{\textbf{Reporting}}\end{tikzcd}$$

Figure 2. The process of Quality Assurance Activities

Source: (Hass, 2008)

Thus, quality criteria should be defined in the first place. These attributes of software development process are aimed to explicate the particular quality factor(-s) in order to express the quality level that must be reached (Hass, 2008; McCall et al., 1977). The elicitation of the right criteria is a complex process which depends on the business needs and the software type. McCall et al. (1977) further proposes the grouping of candidate quality factors into a smaller, concise number of elements by following aspects:

- User-oriented terms are potential factors, software-oriented terms are potential criteria;
- Synonyms that are identified are grouped together.

The grouping helps to cover the comprehensive set of software quality factor characteristics desired. Moreover, quality metrics should be introduced in the process of quality criteria definition as well - they measure the relationship between the criteria or sub criteria and quality factors. After criteria and related quality factors are defined, testing process starts by using validation and verification activities. According to Hass (2008), these activities check whether the quality criteria have been met by the specified software requirements under testing. In case the testing of particular functionality has failed the validation and verification, a bug is reported and handed to the right authority for fixing. Once the testing is passed to live up to the specified quality criteria, it should be provided in repots of passed results. In both ways, QA reports on the findings and results should be generated. In order to keep software quality it is essential follow the provided process of QA activities.

In addition to software quality through quality factors, it could be analyzed as the absence of defects as well. According to Graham et al. (2008), the term defect in general determines a non-compliance between the software functionality and specified requirements (or actual and expected results of test). Sometimes the terms, defect, error, fault and failure, are used as synonyms, but each term has a specified situation to be identified. The difference between these terms are distinguished by Graham et al. (2008) and Hass (2008):

- defect - a bug introduced by programmer inside the code that leads to non conformance between the software functionality and specified requirements;
- error - mistake made by a programmer (human action that produces an incorrect result) that could be done because of some following reasons: confusion in understanding the functionality of the software, some miscalculation of the values, misinterpretation of any value, etc.;
- failure - the inability of a system or component to perform its required functions within specified performance requirements ("IEEE Standard Glossary of Software Engineering Terminology," 1990).

Hass (2008) argues that the defect causes no harm until it is not encountered by anybody, but if it is detected in a later phases of SDLC, it can rise to a failure. The types of error and defect in the different stages of SDLC is shown in a figure below (see Figure 2, page 19). Failures affect the cost of software development the most, so more attention should be paid on earlier stages of SDLC when defect cost is the least.

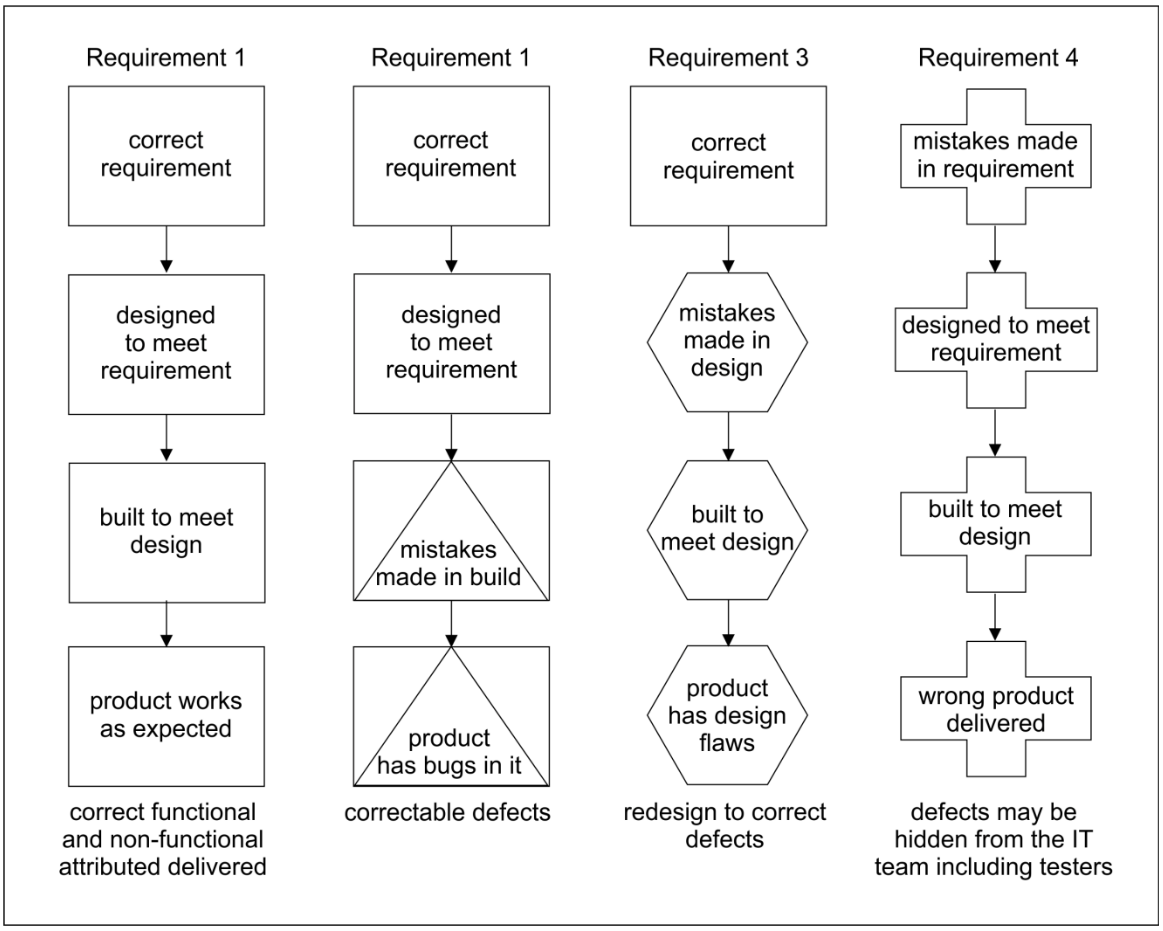


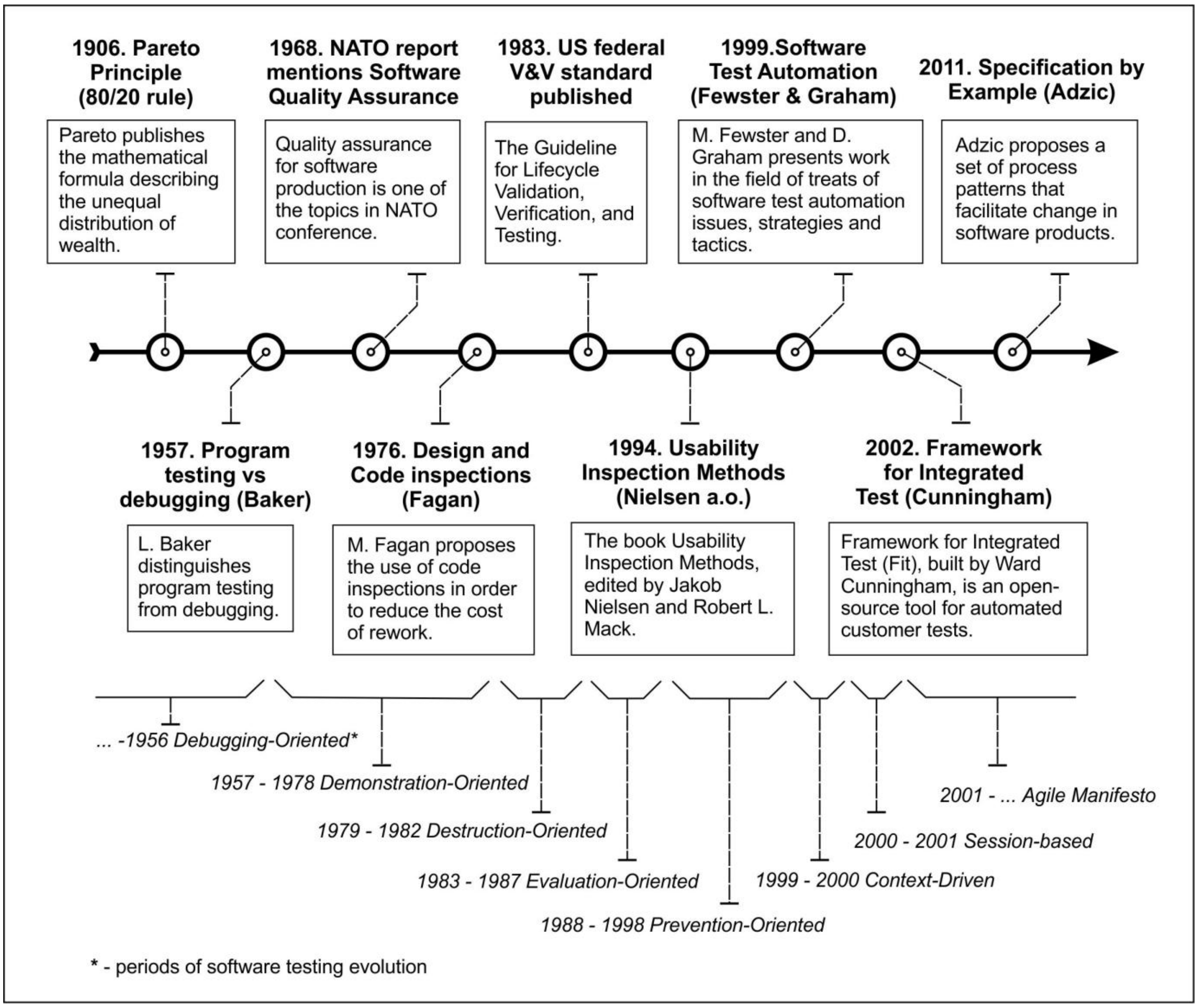
Figure 3. The types of error and defect in the SDLC

Source: (Graham et al., 2008)

To summarize all that was presented, we can provide a definition which describes the software quality in general: the degree to which the software (its functionality and requirements) meets specified requirements and user needs and expectations. The main principles of software QA are specified: QA activities, such as validation and verification, ensure the software quality by exact quality criteria that are aimed to explicate the particular quality factor(-s) in order to express the desired quality level; quality factors are conformed in development phase and it leads to expectations of post-production activities; all defects found during validation or verification should be reported and handed to the right authority for fixing as soon as possible. All mistakes made in at the early stage of SDLC can rise to failures if they are not detected earlier. As a result, it can significantly increase the impact on SDLC cost and schedule. Therefore, more attention should be paid on the early phases of software development life cycle when the rework cost most often relatively low in order to improve software quality and avoid costly failures.

1.2. The overview of Software testing fundamentals: Concepts, History, Main principles

Software testing has become an important software development activity when the terms software testing and debugging were separated. Until the differentiation of terms which was introduced by (Baker, 1957), the view of testing was presented in different ways, such as The Debugging-Oriented, Demonstration-Oriented, Destruction-Oriented, Evaluation-Oriented or Prevention-Oriented (Gelperin & Hetzel, 1988). Later more views as models were introduced: Context-Driven Testing, Session-based testing, Agile Manifesto (“The History of Software Testing,” 2015). For instance, during the first period, debugging-Oriented, testing focused on hardware and the approach of programming was interpreted as "you wrote a program and then you checked it out". While other approaches focused on: 1) "make sure the program runs", 2) "the primary goal is to find errors", 3) introduced methodology of software development with its following elements: SDLC, analysis, reviews, test activities, validation and verification techniques, 4) generalized standard for unit testing process, main focus on defects prevention by activities, such as planning, analysis of requirements and objectives, preparing a detailed architectural design, implementation, execution and maintenance of tests, 5) "the value of any practice depends on its context ", 6) combined accountability and exploratory testing to provide rapid defect discovery, management control and metrics reporting, 7) following rules by Agile Manifesto which analyze testing from the customer perspective and give insights on early testing of SDLC. These approaches and other important historical events (see Figure 3, page 21) shows the growth of software testing. Further, the terms and main purpose of software testing should be analyzed in order understand the role of testing and make a view how the testing has changed historically.



1906. Pareto Principle (80/20 rule)
Pareto publishes the mathematical formula describing the unequal distribution of wealth.
1968. NATO report mentions Software Quality Assurance
Quality assurance for software production is one of the topics in NATO conference.
1983. US federal V&V standard published
The Guideline for Lifecycle Validation, Verification, and Testing.
1999.Software Test Automation (Fewster & Graham)
M. Fewster and D. Graham presents work in the field of treats of software test automation issues, strategies and tactics.
2011. Specification by Example (Adzic)
Adzic proposes a set of process patterns that facilitate change in software products.
1957. Program testing vs debugging (Baker)
L. Baker distinguishes program testing from debugging.
1976. Design and Code inspections (Fagan)
M. Fagan proposes the use of code inspections in order to reduce the cost of rework.
1994. Usability Inspection Methods (Nielsen a.o.)
The book Usability Inspection Methods, edited by Jakob Nielsen and Robert L. Mack.
2002. Framework for Integrated Test (Cunningham)
Framework for Integrated Test (Fit), built by Ward Cunningham, is an open-source tool for automated customer tests.
... -1956 Debugging-Oriented*
1957 - 1978 Demonstration-Oriented
1979 - 1982 Destruction-Oriented
1983 - 1987 Evaluation-Oriented
1988 - 1998 Prevention-Oriented
1999 - 2000 Context-Driven
2000 - 2001 Session-based
2001 - ... Agile Manifesto
* - periods of software testing evolution

Figure 4. The Software Testing timeline

Prepared by author according to Sources: (Gelperin & Hetzel, 1988; “The History of Software Testing,” 2015)

Hass (2008) claims that "there is no universal set of definitions of test concepts", consequently the universe of testing has been defined as multidimensional. The most used facets of universe are listed below (see Table 4, page 21) in order to cover some part of the complexity of the testing.

Table 4. The general facets of multidimensional testing

• Coding languages • Development models • Development paradigms • Incidents	• Maturity models • Money • People skills • People types • Process improvement • Product	• Product paradigms • Product risks • Quality assurance activities • Quality factors • Quality goals	• Standards • Testing obstacles • Testing progress • Test approaches • Test basis • Test effort • Test levels	• Test processes • Test process improvement • Test project risks • Test scopes • Test techniques • Test tools

• Incident handling	architectures	• Resources • Risk willingness	• Test objectives • Test policy	• Test types • Time

Prepared by author according to Source: (Hass, 2008)

These terms can be interpreted as dependent factors of all software testing process. Each of them affects software quality and style of testing. Some of them have been already discussed in previous subchapter, others - will be introduced later. Despite the multidimensional view of testing the main terms should be provided for further discussion of testing techniques. The classic definition is as follows: "Testing is the process of executing a program with intention of finding errors" (Myers, 1979). Much more extended and formal term is suggested by "IEEE Standard Glossary of Software Engineering Terminology" (1990)**:**

(1) The process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component. (2) The process of analyzing a software item to detect the differences between existing and required conditions (that is, bugs) and to evaluate the features of the software items".

(Galin, 2004) proposed an alternative term consisting of previous ones: *Software testing is a formal process carried out by a specialized testing team in which a software unit, several integrated software units or an entire software package are examined by running the programs on a computer. All the associated tests are performed according to approved test procedures on approved test cases.*

We will use the more formalized term by "IEEE Standard Glossary of Software Engineering Terminology," 1990) in order to take a wider look on testing and focus on compliance of requirements or "differences between existing and required conditions" which is called a bug or defect. Hamlet (1995) examined the terms as well with the assumption that software quality is comprehended as the absence of defects. Some methods proved the theoretical possibility of "zero-defect" software with the binary ideal of "correct" /"not correct". Although, the correctness of software could be interpreted in different ways and theoretical calculations could not fit to practical usage. DeVolder et al. (2008) supports this idea adding the explanation: testing can never completely establish the correctness because whether a failure is thrown, the software does not do what the user expects (there is no correctness). Moreover, Hamlet (1995) argued against Myers (1979) on the argument "the best development process is to find failures" (Myers, 1979). Hamlet (1995) stated that failure-finding is a dangerous measure. As a result, he suggested to use a construct of two definitions for testing in order to define software quality clearly. The suggested term is inherited from Dahl, Dijkstra, & Hoare (1972) as "Testing can show the presence of errors, never their absence." and Myers (1979) "The purpose of testing is to find errors". Indeed, Graham et al. (2008) agrees that usually the main purpose of testing is to find defects. However, in spite of software quality as we defined before, we would rather recommend focusing on the objectives listed by (Hambling & Morgan, 2011):

- Finding mistakes made by the programmer while developing the software;
- Gaining confidence in and providing information about the level of quality;
- Prevention from defects;
- Compliance of business and user requirements;
- Ensuring that software meets the Business Requirement Specification (hereinafter BRS) and System Requirement Specifications (hereinafter SRS).

Further, DeVolder et al. (2008) distinguishes software testing as a process with a purpose to measure the quality of developed software by quality factors. Classification of quality factors by McCall et al. (1977) have been introduced in the subchapter 1.1 (see Table 2, page). The extended table with the requirements and their corresponding tests are presented further (see Annex 1, page 70). Some of quality factors are divided in sub factors with corresponding tests in order to get full coverage of the respective requirements. In fact, the list of provided tests describes generally, what types of tests should be prepared before testing. Such tests can be executed by different testing techniques or tools - this will be discussed in further chapter.

To get full coverage of tests and perform efficient testing, some regulations should be followed. For this purpose Graham et al. (2008) extracted the main principles of testing. They are as follows:

1) **Testing shows presence of defects**: that was discussed in previous paragraphs as the possibility of "zero-defect" software. Is was concluded that this theoretical assumption cannot be applied in practice. Even after testing is completed it cannot be stated that the product reached the effect "zero-defect". Testing always reduces the number of undiscovered defects, but there are always remained issues that could be detected in the last phase of SDLC as we discussed before (the rise from error to failure) or even more later if the undiscovered defect affects only one component of software which is used rarely. Moreover, even if no defects are found, it is not a proof of correctness as well.

2) **Exhaustive testing is impossible**: testing everything including all combinations of inputs and preconditions is not possible because of time constraints, the vast of data used in preconditions or for inputs, limited human capabilities to overcome the full testing (all combinations of inputs and preconditions). Myers et al. (2011) follows the idea of exhaustive testing and adds some implications of this: it is impossible to test a software to guarantee that it is error free (or "zero-defect" as it was discussed in the first principle) and a fundamental consideration in software testing is one of economics. Thus, instead of doing the exhaustive testing, the risk and priorities are suggested to use to distribute testing efforts. As in the researches (Elbaum et al., 2002; Rothermel et al., 1999; Srivastava, 2008), test case prioritization can be used for assisting with regression testing by setting priorities for test cases and performing testing from the test with highest priority (lowest priority test can be skipped

in case of tight schedule by taking some risk). These techniques will be discussed more detailed in further chapters. Therefore, accessing and managing risk take an important role in QA activities in any project.

3) **Early testing**: software testing activities should start as early as possible in order to avoid costly failures in SDLC. As it was concluded before - all mistakes made in early stages of SDLC could rise to failures; thus this incident affects the quality of software.

4) **Defect clustering**: "a small number of modules contains most of the defects discovered during pre-release testing or shows the most operational failures". This means that the distribution of defects are not across the application but rather centralized in limited sections of the application, such as the small amount of particular modules or units that are used rarely or not fully testing because of different causes, as for instance, lack of people skills on product architecture and paradigms, incident handling, test levels, unclear test objectives and test process, testing obstacles, limited resources and time devoted for testing etc. Defect clustering in software testing is based on the Pareto principle, also known as the 80-20 rule, where it is stated that approximately 80% of defects are contained in 20% of software components (Boehm & Basili, 2001; Ostrand, Weyuker, & Bell, 2005). Studies by Boehm & Basili (2001), Gittens, Kim, & Godwin (2005), Li et al. (2009) and Ostrand et al. (2005) have proved that this principle fits defect distributions by components. The main benefit of defect clustering is that testing can be prioritized on focusing the same component rather than performing full testing of all components. Hence, the more number of defects will be found in shorter period. As a result, it increases testing efficiency and reduces the cost of software development.

5) **Pesticide paradox**: this paradox appears when the same kinds of tests are performed a certain number of iterations without any additional steps. In fact, this leads to more not detected defects because the same set test cases will no longer be able to find any new bugs. To overcome this “Pesticide Paradox”, it is essential to updated test cases regularly by adding some new and different tests to cover different units, components (modules) of the software or system in order to detect potential bugs.

6) **Testing is context dependent**: testing is basically context dependent. Different kinds of software are tested differently because of various reasons, such as different testing goals, changed BRS or SRS, available resources for testing, including the testing skills of employees and the tools provided by enterprise etc.

7) **Absence - of - errors fallacy**: if the system built is unusable and does not fulfill the user’s needs and expectations then finding and fixing defects does not help. In a first place, the more efforts should be taken to solve the issues related with broken system built and testing should be postpone a while.

After discussion of Software Testing Fundamentals the main conclusions should be made. Firstly, brief history of testing evolution showed that there are different approaches of testing in a timeline, such as The Debugging-Oriented, Demonstration-Oriented, Destruction-Oriented, Evaluation-Oriented, Prevention-Oriented, Context-Driven, Session-based, Agile Manifesto. Only after Debugging-Oriented period the differentiation between terms "debugging" and "testing" was presented. Secondly, the vast of concepts has been defined as multidimensional and the most used facets of testing, such as QA activities, quality factors, incident handling, standards, test levels, test types, test tools, people skills, resources, time etc., have been distinguished. The more formalized term by ISO standard decided to use for a wider look on testing and focus on compliance of requirements or "differences between existing and required conditions" which is called a bug or defect. Thirdly, the objectives of testing have been discovered: to find mistakes while developing the software, to gain confidence in and provide information about the level of quality, to prevent from defects, to ensure compliance of business and user requirements, to ensure that software meets the BRS and SRS. Fourthly, the classification of quality factors with the requirements and their corresponding tests has been illustrated in the extended table. Fifthly, the main principles of testing are identified in order to get full coverage of tests and perform efficient testing. They are as follows: 1) Testing shows presence of defects, 2) Exhaustive testing is impossible, 3) Early testing,4) Defect clustering, 5) Pesticide paradox, 6) Testing is context depending, 7) Absence – of – errors fallacy. Moreover, it is stated that the risk and priorities should be used to distribute testing efforts instead of doing the exhaustive testing.

Finally, the relationship between software testing and quality assurance can be distinguished after generalization of Software Testing Fundamentals and Quality Assurance is made. The main purpose of software testing is to discover defects (including prevention) while developing the software, provide information about the level of quality, to ensure compliance of business and user requirements, and to ensure that software meets the requirements as well. Whereas, quality assurance is defined as a process, set of activities (instructions) on how to ensure the consistency of software according the requirements during testing phase. The main activities of quality assurance are verification and verification that are performed by different software testing techniques. In particular, the relationship between software testing and software quality is essential for reducing impact on software life cycle cost and schedule.

2. AN OVERVIEW OF SOFTWARE TESTING TECHNIQUES AND THEIR USE IN ENTERPRISES

This chapter presents the software testing techniques, including static and dynamic testing as a code analysis, test design based methods to create test cases, software testing levels that are analyzed as a stage of software development. Test execution types, including manual and automated testing, are analyzed as well. Further, the practical use of software testing techniques in enterprises is examined to identify how enterprises adopt those software testing techniques and what benefits and limitations they are facing while using any of software testing techniques.

2.1. **Testing techniques as code analysis**

The activities for software quality assessment can be divided into two broad categories, such as static analysis and dynamic analysis (Naik & Tripathy, 2008). Static analysis and dynamic analysis are related with each other from code's perspective, as the first one describes the testing without executing the code, while other uses that analyzed code for execution (it evaluates the dynamic behavior). Some researchers suggests to create a hybrid analysis that combines both approaches for better effectiveness (Ernst, 2003). It is noticed that both should be performed repeatedly and alternated. To understand better each of those code analysis techniques, the main principles and their types will be introduced in the following subchapters.

2.1.1. *Static testing*

Static testing (static analysis) is performed before the code is executed or completed. It has been already introduced in subchapter 1.1 as a technique for verification process. The following types of static testing are distinguished by Graham et al. (2008) and Myers et al. (2011) and analyzed below:

- Code or Design Inspection - the most formal review and aimed at detecting all faults, violations of development standards, and other problems in design and code. According to Fagan (2001), all required documents, including detailed design in specific areas like paths, logic of code, should be prepared and presented for inspection meeting. During inspection process the code is inspected in order to found defects that are handed to the author for fixing.
- Review (informal, peer, technical, management) - in practice, technical reviews vary from quite informal to very formal (Graham et al., 2008). Review are performed by the experts (such as architects, designers, key users). During review actual work is compared with established standards to determine whether the product is ready to proceed with the next phase of SDLC.
- Walk-through - a non formal process when a programmer leads team members and other interested parties through a segment of documentation or code, and the participants ask questions and

make comments about possible errors, violation of development standards, and other problems (Graham et al., 2008; “IEEE Standard Glossary of Software Engineering Terminology,” 1990). Although, DeVolder et al. (2008) and Hass, (2008) define an additional technique - Audit which is the most formal static testing technique. Audits are performed by external auditors with the purpose of providing "an independent evaluation of an activity’s compliance to applicable process descriptions, contracts, regulations, and/or standards" (Hass, 2008, p. 301). The author discovers the main disadvantages of audits - they are quite expensive and the least effective static testing type; however, audits are usually performed because they are mandatory in some context. The main similarities and differences in the most commonly used techniques are illustrated in a table below (see Table 4, page 27).

Table 5. General principles for Static Testing techniques

	Walk-through	**Technical Review**	**Management Review**	**Inspection**
Primary purpose	Finding defects	Finding defects	Finding defects	Finding defects
Secondary purpose	Sharing knowledge	Make decisions	Monitor and control process	Process improvement
Preparation	Usually none	Familiarization	Familiarization	Formal preparation
Usage of basis	Rarely	Maybe	Maybe	Always
Leadership of the meeting	Author	As appropriate	As appropriate	Trained moderator
Recommended group size	2-7	3 or more	3 or more	3-6
Formal procedure	Usually not	Sometimes	Sometimes	Always
Volume of material	Relatively low	Moderate to high	Moderate to high	Relatively low
Collection of metrics	Usually not	Sometimes	Sometimes	Always
Output	Sometimes an informal report	More or less formal report	More or less formal report	Defect list, measurements, and formal report

Source: (Hass, 2008)

Regarding the common features of techniques, the main purpose of static testing in general can be defined - defects prevention in early stage of SDLC and improvement of software quality (correctness of code, compliance of requirements) with the aid of team members involved. Whilst, the researchers (Nidhra & Dondeti, 2012; Saglietti et al., 2008) presents more detailed definition static testing purpose: to check whether the code meets functional requirements, design, coding standards; to identify whether and all functionalities are covered; to uncover incorrect programming assumptions; to find logical and random typographical errors in the program code. Emanuelsson & Nilsson (2008) distinguishes the main runtime problems (errors) that are detected by static testing:

- **Improper resource management** - resource leaks of dynamically allocated memory, files, sockets that are no longer used;
- **Illegal operations** of arithmetic functions, illegal values, arrays addressing, null pointers referencing etc.;
- **Dead code and data** - code and data that is not reachable or not used;
- **Incomplete code** - missing initialized variables, functions with unspecified return values and incomplete branching statements.

Some of such errors can be detected by tools instead of manual testing (Hass, 2008). In spite of the variety of static analysis tools available on the market (e.g. "PolySpace", "C Verifier", "SonarQube"), or as open source systems (e.g. ARCHER, BOON, SPLINT, UNO) ("SonarQube," 2016; Zitser et al., 2004), some struggling issues can be faced while choosing the right tool or considering the need of it: the functionality of tool depends on the specified programming language which is designed for; more complex system requires deeper analysis compared with a simple one; limited enterprises resources restrict the choice of desired tool. Some of tools are standard development tools, such as compilers or linkers, while others are aimed for code analysis that monitor and track the following issues (Graham et al., 2008; Hass, 2008): the flow of code instructions; the data flow accessed and modified by code; compliance to standards that consists of a set of programming rules and other conventions; calculation of code metrics that analyze the depth of nesting, cyclamate number and number of lines of code.

After discussion of static testing features, the value for all SDLC is identified: static testing reduces the chances of failures in later phases of SDLC; it prevents from runtime problems (errors) that are detected mainly by static software testing technique (Emanuelsson & Nilsson, 2008); a vast of complex rules in the coding standards can be verified by tools instead of a time-consuming manual review. It is noticed that missed defects during static testing could be detected at the latest phases of SDLC; thus, it affects the cost of whole software development process. As we discussed in subchapter 1.1, all defects found at very early stage of SDLC can be fixed at that moment with relatively low cost.

2.1.2. *Dynamic testing*

Dynamic testing (or dynamic analysis) compared with static testing executes the software actually. It is defined as the process of evaluating a system or a component based upon its behavior during execution in order to expose possible program failures (Hass, 2008). It is done by tools that helps to gather run-time information about the behavior and state of software, thus, Graham et al.

(2008) explains that they are ‘analysis’ rather than ‘testing’ tools. The main features of dynamic analysis tools are listed below:

- to report on the state of software during its execution (Naik & Tripathy, 2008);
- to monitor the allocation, use and reallocation of memory (Naik & Tripathy, 2008);
- to identify unassigned pointers (Hass, 2008; Naik & Tripathy, 2008);
- to detect memory leaks (Naik & Tripathy, 2008; Graham et al., 2008; Hass, 2008);
- to identify pointer arithmetic errors, e.g. null pointers (Graham et al., 2008; Hass, 2008; Naik & Tripathy, 2008);
- to identify time dependencies (Graham et al., 2008; Naik & Tripathy, 2008);
- coverage analysis (Hass, 2008) - these tools provide objective measurement for some white-box test coverage metrics (e.g. statement coverage or branch coverage; both will be presented in the further subchapter);
- performance analysis (Hass, 2008) - it measures the performance of a product under the controlled circumstances before the product is released;

Some tools (called as memory debuggers) used for detecting memory leaks and uses of dead storage are as follows: "Purify" and "LCLint" (Ernst, 2003). Whereas, other tools are more powerful and include more dynamic analysis features mentioned above - "VB Watch" (Aivosto, 2016) or "IBM Rational AppScan" (“IBM - Software - IBM Security AppScan,” 2016).

Dynamic testing executes the software and validates the output with the expected outcome and it can be either black or white box testing (Graham et al., 2008). Since this technique is performed during validation process, the testable levels (test levels will be analyzed more detailed in the subchapter 2.3) are distinguished:

- **Unit Testing**: individual units or modules are tested by the developers. It involves testing of source code by developers as well.
- **Integration Testing**: individual modules are grouped together and tested by the developers. The purpose is to determine that modules are working as expected once they are integrated together.
- **System Testing**: checking whether the system or application meets the BRS and SRS by testing the whole system.

After discussion of both static and dynamic testing, the main differences can be identified and presented in a table (see Table 7, page 30).

Table 6. The differences between Static Testing and Dynamic Testing

	Static testing	Dynamic testing
Executing the software	No	Yes
Process for evaluating software	Verification	Validation
Main focus	Prevention of defects	Finding and fixing defects
Methods	Checklist and process	Test cases used for execution
Code coverage	The structural and statement coverage testing	The executable file of the code
Cost of fixing defects	Less	High
Recommendations for good quality	More reviews and comments	More defects
Meetings required	Loads of meetings	Less meetings

Prepared by author according to Sources: (DeVolder et al., 2008; Ernst, 2003; Graham et al., 2008; Hass, 2008)

To summarize both static and dynamic testing, the main features are identified. Static testing reduces the chances of failures in later phases of SDLC; focus on prevention of defects during verification process. However, it is time consuming activity. Whereas, dynamic testing executes the software and validates the output with the expected outcome and it can be either black or white box testing. The main focus is on finding defects. Both techniques can be performed by tools, however, there are some limitations. Automated tools of static analysis do not support all programming languages, while tools for dynamic analysis provide a false sense of security that everything is being addressed.

2.2. Test design based Techniques

Traditionally software testing techniques can be broadly divided into white-box testing and black-box testing (Liu & Kuan Tan, 2009), however there are few more test design techniques that are used rarer than white-box and black-box techniques. They are as follows: experience-based (Graham et al., 2008; Hambling & Morgan, 2011; Hass, 2008; Myers et al., 2011) and error guessing (Myers et al., 2011) or called as defect-based (Hass, 2008). Sometimes the gray-box technique is separated as the different approach even if it based on both white-box and black-box techniques (Mohd Ehmer Khan & Khan, 2012). All these approaches focus on the sources of information for test design. There are many advantages of using techniques to design test cases. They provide good insights for finding possible faults - this is the most essential objective for all software development. Indeed, white-box testing and black-box testing techniques can be perform by static and dynamic analysis in order to find defects.

White-box testing and black-box testing are considered corresponding to each other. Some researchers underline that it is essential to cover both specification and code actions in order to test software more efficiency (Jorgensen, 2016; Liu & Kuan Tan, 2009). Whereas, Hass (2008) see test design based techniques are as a very precise and systematic analysis of BRS or SRS which makes testing more effective and corrective. Designing test cases by these techniques also shows the experience of testers, whereas, other testers are able to learn from provided test cases by executing them. One of the most important thing for black-box and white-box testing is to achieve a full coverage of what is required to cover: it could be requirements, or statements, or paths - it depends on selected technique and test objectives. It is noted that some difficulties could be faced with even when the full coverage is obtained: faults could remain undetected because of non conformance of the code and users expectations. To overcome this or mitigate this risk as much as possible, the validation of the requirements should be performed narrowly before starting the dynamic testing. As we discussed in a previous chapter - first focus should be made on the first phase of SDLC. More detailed white-box testing and black-box testing will be analyzed further in this subchapter.

Grey box is seen as the combination of white-box and black-box techniques (Mohd Ehmer Khan & Khan, 2012; Sawat, Bari, Chawan, & P. M., 2012). In grey box testing the tester must have knowledge of internal data structures and algorithm of application, for the purpose of designing test cases (M. E. Khan, 2011a). In spite of combination of two techniques, the grey box testing won't be discussed detailed; the main focus is on mostly used techniques.

According to Hambling & Morgan (2011) experience-based techniques are based on the users' and the testers' knowledge and skills to determine the most important areas of a system to be chosen to test. Experience-based techniques go together with specification-based and structure-based techniques, and are also used when there is no specification from which to derive specification-based test cases, or an inadequate or out of dated specification is used, or there is no time to run the full structured set of tests. It is recommended to use experience-based techniques even when specifications are available. Structured tests could be augmented with some additional steps in order to find defects similar to those which are founded by experience in other similar systems. Some types of experience-based techniques are as follows (Graham et al., 2008; Hass, 2008): error guessing, checklist-based. Error guessing depends on experience of tester as good testers know where the defects are most likely to be. Second type is uses checklists to guide testing where the checklist is basically a high-level list, or a reminder list, of areas to be tested. Finally, the main focus of exploratory testing is on exploring software with intent to understand its behavior. The main feature of these types that they are based by tester's experience. The may be used before the other techniques to uncover "weak" areas, but experience-based techniques must never be the only technique to be used.

Taking into consideration defect-based technique, it is defined as less systematic than the previously discussed techniques, since it is usually not possible to make exhaustive collections of expected defects. Whereas, experience-based testing techniques are based on the tester's experience with testing, development, similar applications, the same application in previous releases, and the domain itself (Graham et al., 2008).

Furthermore, the main test design techniques can be classified in smaller techniques, while the wider categorization group, static and dynamic testing, covers all previously mentioned techniques. The tree categorization of mostly used software testing techniques is presented below (see Figure 4, page 32). These classified techniques will be discussed further in this chapter.

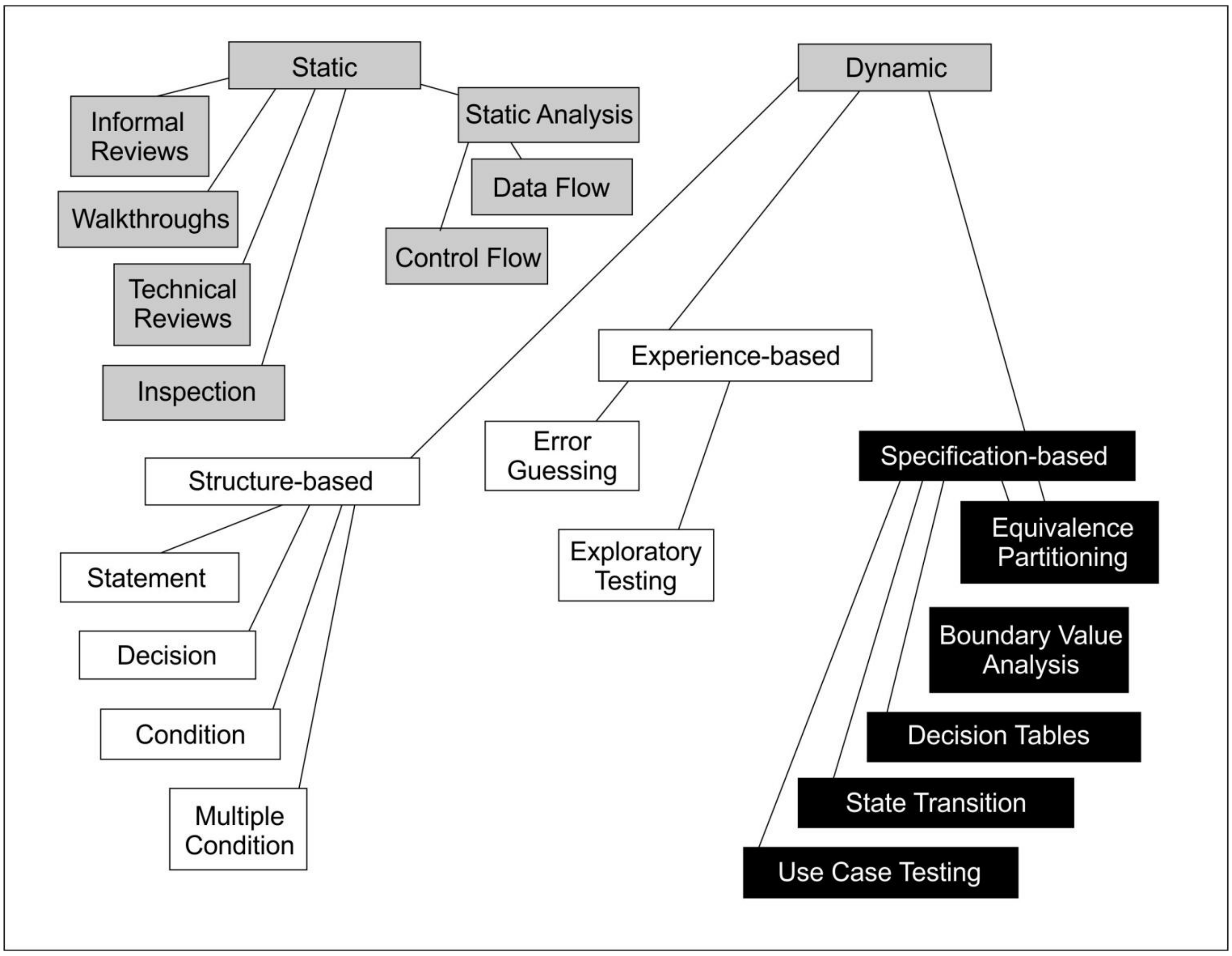


Figure 5. The tree structure of the testing techniques

Source: (Hambling & Morgan, 2011)

2.2.1. *Structure-based (white-box) techniques*

White-box testing techniques are called structural testing techniques or as Myers et al. (2011) noticed - logic-driven techniques. Structural testing is defined as testing that takes into account the internal mechanism of a system or component ("IEEE Standard Glossary of Software Engineering Terminology," 1990). Indeed, its techniques are based on deriving test cases directly from the internal structure of a component or system with intent to explore system or component structures at several levels. Traditionally the internal structure has been interpreted as the structure of the code (Liu & Kuan

Tan, 2009). According to Hass (2008) and Naik & Tripathy (2008), in structural testing (white-box), the main focus is on the testing of code and they are primarily used for component testing and low-level integration testing. The researchers also notes the use in system (Graham et al., 2008; Sawat et al., 2012) and acceptance testing (Graham et al., 2008) with the different structures (e.g. the coverage of menu options could be the structural element in system or acceptance testing). Acceptance testing is defined as "formal testing conducted to determine whether or not a system satisfies its acceptance criteria and to enable the customer to determine whether or not to accept the system (“IEEE Standard Glossary of Software Engineering Terminology,” 1990). Acceptance testing will be discussed more detailed in second subchapter. Developing the previous example of code coverage, the following code coverage criteria of structural test case design techniques are enumerated further (Hass, 2008):

1. Statement testing - test cases are designed to execute statements that are defined as a no comment or nonwhite space entity in a programming language,
2. Decision testing/branch testing - testing of decision outcomes. Mostly a decision has two outcomes, such as "True" or "False", but it might have more outcomes, for example, in "case of ..." statements.
3. Condition testing - testing of conditional expressions (e.g. AND, OR, a < b etc.).
4. Multiple condition testing - combinations of condition outcomes are tested in order to get fully multiple combination coverage.
5. Condition determination testing - testing of branch condition outcomes that independently affect a decision outcome.
6. LCSAJ (loop testing) - testing of loop iterations that start at a specific point in the code and end with a jump (or at the end of the component).
7. Path testing - testing of a sequence of executable statements in a component from an entry point to an exit point to get full coverage of paths.
8. Inter-component testing - this testing technique is used in integration testing where the test objects are interfaces (interfaces exist between interacting components and systems).

These different techniques exercise every visible path of the source code to minimize errors and create an error-free environment. The main view of white-box testing is to get knowledge on which line of the code is being executed and being able to identify what the correct output should be. Galin (2004) and Graham et al. (2008) identify the main advantages: structure-based techniques can be used at all levels of testing starting from unit (component) and ending at acceptance testing, direct statement-by-statement checking of code ensures software correctness as expressed in the processing paths, including whether the algorithms were correctly defined and coded. The research by M. E. Khan (2011b) presents few more benefits: white-box testing techniques reveals error in hidden code by removing extra lines of code and maximum coverage is attained during test scenario writing. Whereas, some disadvantages are seen as well: it is very expensive testing techniques as they require a skilled tester to perform such testing; many paths remain untested because of difficulties to discover hidden errors in a complex system; some of the codes omitted in the code could be missed out (M. E. Khan, 2011b). In addition, there is no ability to test software performance in terms of reliability, load durability, and other testing classes related to operation, revision and transition factors (Graham et al., 2008).

To conclude this subchapter, the main features of white-box techniques are distinguished. First of all, white-box techniques are based on deriving test cases directly from the internal structure of a component or system and the main purpose is to explore system or component structures at several levels. Furthermore, in spite of the fact that they are primarily used for component testing and low-level integration testing, system and acceptance levels are tested by white-box techniques as well. Finally, white-box testing techniques are seen as very expensive testing techniques as they require a skilled tester to perform such testing, whilst, it reveals error in hidden code by removing extra lines of code and maximum coverage is attained during test scenario writing

2.2.2. *Specification-based (black-box) techniques*

Black-box techniques, known as specification-based techniques, are also called as functional testing (Liu & Kuan Tan, 2009) or input/output driven testing techniques (Graham et al., 2008; Sawat et al., 2012) because they view the software as a black-box with inputs and outputs generated in response to selected inputs and execution conditions. According IEEE Standard Glossary of Software Engineering Terminology (1990) functional testing is defined as "testing conducted to evaluate the compliance of a system or component with specified functional requirements". Thus, the main focus of functional techniques is on validating the software whether it meets requirements. These techniques design test cases based on the information from the requirements specification, including both functional and non-functional (e.g. performance, usability, portability, maintainability, etc.) aspects. Software tester is concentrating on what the software does according the specified requirements instead of analyzing how the system works. According to Hass (2008), these test case design techniques can be used in all stages and levels of testing, especially, they are useful in high-level tests, such as acceptance testing and system testing, where the test cases are designed from the requirements. Test cases can be supplied with structural or white-box test in order to get full coverage. The functional test case design techniques are enumerated by Hass (2008):

1. Equivalence partitioning and boundary value analysis - equivalence partitioning can reduce the number of test cases, as it divides the input data of a software unit into partition of data from which test cases can be derived. While boundary value analysis focuses more on testing at boundaries, or where the extreme boundary values are chosen (Graham et al., 2008; M. E. Khan, 2011a).

2. Domain analysis - it can be used to identify efficient and effective test cases when multiple variables can should be tested together (as multidimensional partitions or domain).

3. Decision tables - this technique is applied to specific situations or inputs where there are different combinations of inputs that result in different actions as well (Graham et al., 2008).

4. Cause-effect graph - testing begins by creating a graph and establishing the relation between the effect and its causes (M. E. Khan, 2011a).

5. State transition testing - it is used where some aspect of the system can be defined as a ‘finite state machine’. A system where different output is get for the same input, depending on what has happened before, is a finite state system (Graham et al., 2008). It is useful for navigation of graphical user interface (M. E. Khan, 2011a).

6. Classification tree method - partitioning of different classes are made by identifying test relevant aspects (classifications) and their corresponding values (classes).

7. Pair wise testing - test cases are designed to execute possible combinations of each pair of input parameters (M. E. Khan, 2011a).

8. Use case testing - testing the main flow and alternative flow (if it is needed) step by step as it is specified in the description of use case.

9. Syntax testing.

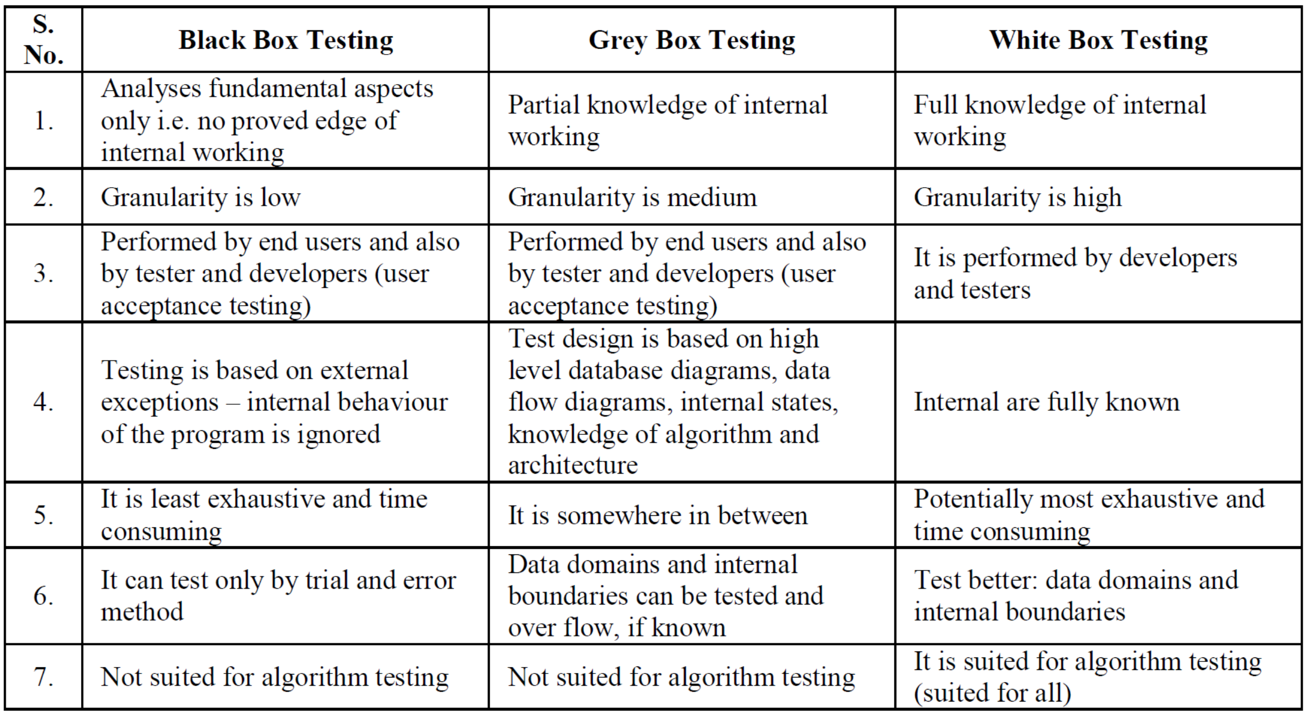
Regarding the testing techniques enumerated above it is assumed that black-box testing techniques have the biggest collection of testing methods that mainly focus on compliance of requirements and user needs (Graham et al., 2008; Myers et al., 2011; Nidhra & Dondeti, 2012; Sawat et al., 2012). Thus, these techniques are the most used while validating the software by BRS and SRS.

Research that was made by M. E. Khan, (2011a) has represented the main advantages of black box testing: efficient for large code segment, users perspective are clearly separated from developers perspective (programmer and tester are independent of each other). However, there are some limitations as well: test coverage is limited as the access to source code is not available; it is difficult to associate defect identification in distributed applications. Moreover, many software paths remain untested because of absence of control of line coverage (Galin, 2004). As test cases are created according to specified requirements (from business perspective), some part of the code lines could not be covered by test cases, as a result, black box tests may not execute particular code lines that are not covered by test cases.

To summarize the main features of black-box testing techniques some conclusions are made. Firstly, these techniques design test cases based on the requirements specification, including both functional and non-functional aspects, with intent to validate whether the software meets requirements. Further, these techniques can be used in all stages and levels of testing and they are seen as efficient for large code segments. Moreover, the independent work of programmer and tester enables efficient testing from user's perspective. However, some software paths could still remain untested as the functionality (derived from business requirements) covered by test cases does not include code coverage.

After discussion of box testing approaches, the main differences between them (including grey-box) can be distinguished (see Table 8, page 36).

Table 7. The comparison between three box approaches techniques



S. No.	Black Box Testing	Grey Box Testing	White Box Testing
1.	Analyses fundamental aspects only i.e. no proved edge of internal working	Partial knowledge of internal working	Full knowledge of internal working
2.	Granularity is low	Granularity is medium	Granularity is high
3.	Performed by end users and also by tester and developers (user acceptance testing)	Performed by end users and also by tester and developers (user acceptance testing)	It is performed by developers and testers
4.	Testing is based on external exceptions – internal behaviour of the program is ignored	Test design is based on high level database diagrams, data flow diagrams, internal states, knowledge of algorithm and architecture	Internal are fully known
5.	It is least exhaustive and time consuming	It is somewhere in between	Potentially most exhaustive and time consuming
6.	It can test only by trial and error method	Data domains and internal boundaries can be tested and over flow, if known	Test better: data domains and internal boundaries
7.	Not suited for algorithm testing	Not suited for algorithm testing	It is suited for algorithm testing (suited for all)

Source: (Mohd Ehmer Khan & Khan, 2012)

To sum up all analyzed design based testing techniques, they can be broadly divided into white-box testing and black-box testing. Other techniques, such as, experience-based and defect-based, are used rarely. All these approaches focus on the sources of information for test design. White-box testing and black-box testing techniques can be perform by static and dynamic analysis in order to find defects. The black-box techniques design test cases based on the requirements specification, including both functional and non-functional aspects, with intent to validate whether the software meets requirements. While, white-box techniques are based on deriving test cases directly from the internal structure of a component or system with intent to explore system or component structures at several levels. Grey box testing is seen as the combination of white-box and black-box techniques.

2.3. **Software Testing Levels and corresponding Testing Types**

There are generally four recognized levels of testing that need to be completed before a software can be delivered for users (Naik & Tripathy, 2008; Sawat et al., 2012): unit testing, integration testing, system testing and acceptance. However, some authors tend to include more testing types to categorization by levels: Alpha testing and Beta testing (Graham et al., 2008; Mailewa, Herath, & Herath, 2015), Installation testing (Myers et al., 2011), component (module) testing (Mailewa et al., 2015; Myers et al., 2011), regression testing (Naik & Tripathy, 2008). In our opinion, some techniques, such as Alpha testing, Beta testing and regression testing are different types of testing and they are not related with previous levels which describe levels from code's perspective. In other words, some part of code is merged with another part until the system is fully integrated with all small units (components). Therefore, those techniques will be discussed later as testing types.

Software tests are frequently grouped by software development process, or by the level of specificity of the test. Each phase of SDLC goes through the testing. Thus, main testing levels mentioned before are enumerated and described more detailed below:

1. **Unit testing**: "testing of individual hardware or software units or groups of related units" ("IEEE Standard Glossary of Software Engineering Terminology," 1990). Graham et al. (2008) and Myers et al. (2011) identify the main purpose: to find discrepancies between the program's units (modules) and their interface specifications, and to determine whether the application functions is designed correctly and meet the user specifications. One of the biggest benefits of this testing phase is that it can be run every time a piece of code is changed, allowing issues to be resolved at that moment. However, more attention to maintenance of such tests should be paid as from an every minor code change in a component, to the general refactoring can affect whole system and the tests will likely require revision (Di Tommaso & Roche, 2011).

2. **Integration testing**: a level of the software testing process where individual units are combined and tested as a group in order to test the behavior and functionality of both the modules after integration. There are few types of integration testing (Hass, 2008): Big bang integration testing, Top down, Bottom up, Functional incremental. The main purpose of this level of testing is to reveal faults in the interaction between integrated units and to construct a reasonably stable system for system level testing (Naik & Tripathy, 2008).

3. **System testing**: according to Hass (2008) and Naik & Tripathy (2008) testing is performed on a complete, integrated system to evaluate the system's compliance with its specified requirements and to check that it meets quality standards. It includes a wide scope of testing techniques, for instance, functionality testing, security testing, load testing, stress testing, performance testing etc. System testing level is seen as a critical phase of SDLC because of the need to meet a tight schedule, to detect most of all faults, and verify that fixed defects are working properly without causing new faults.

4. **Acceptance testing**: acceptance testing focus on customer side and the main goal is to ensure that the requirements of the specification are met and the software satisfies the customer 's requirement (Hass, 2008).

In order to complete testing and detect the majority of defects (the exhausting full testing is impossible as we discussed before), Myers et al. (2011) suggests to use the model of test levels corresponding phases of SDLC (see Annex 2, page 71). This approach focuses on distinction of each testing process toward distinction of each development process by verifying each step separately. It means that each development step should be followed by appropriate testing technique which would discover the particular class of errors. The main advantage of this structure - it helps to avoid useless redundant testing and prevents from overlooking large classes of defects.

Whereas, Mailewa et al. (2015), Myers et al. (2011) and Sawat et al. (2012) support the idea of categorization component (or module) testing as well. The main purpose is the same as for previous testing levels - to find defects and to verify their proper functionality that satisfies BRS and SRS. Component testing may be performed in isolated system part which do not depend on development life cycle model chosen for that particular application (Sawat et al., 2012)

Further, the research by Nidhra & Dondeti (2012) identifies more testing techniques related with testing levels that were defined as a part of level testing by some researchers (Graham et al., 2008; Mailewa et al., 2015; Naik & Tripathy, 2008). The testing techniques and their purpose are as follows:

- **Regression Testing** - "Selective retesting of a system or component to verify that modifications have not caused unintended effects and that the system or component still complies with its specified requirements" (“IEEE Standard Glossary of Software Engineering Terminology,” 1990). In other words, the main aim is to ensure that the reliability of each software release and testing after changes has been made. Moreover, after retesting fixed defects tester should verify whether new defects into the system were not appeared (Jorgensen, 2016).
- **Alpha Testing** - this technique is defined more like a strategy instead of testing method according to Graham et al. (2008). Alpha testing is usually done at the developer’s site by a group that is independent of the design team in order to observe the users and note identified problems (Graham et al., 2008; Nidhra & Dondeti, 2012).
- **Beta Testing** - comparing with Alpha Testing, this technique more focuses on the user’s perspective and practices. It is done at the customer’s site with no developer in site. The main purpose is to discover any flaws or issues with user’s help (Graham et al., 2008; Nidhra & Dondeti, 2012).
- **Functional Testing** is performed for a completed software; this testing is to verify that all functionality are implemented by BRS and SRS and the software works as expected. This technique was already discussed as black-box testing technique for designing test cases. There are some functional testing types, namely, usability, smoke, automated, acceptance, regression etc. Although,

this categorization has been made by Mailewa et al. (2015). On the other hand, some researchers include acceptance testing and regression testing into test level categorization as it was mentioned before. Moreover, according Graham et al. (2008) usability should be classified as non-functional testing as it tests the software without prepared requirements and checks whether the software is built in user-friendly form by following criteria: learnability, efficiency, satisfaction, memorability etc. The other technique, smoke testing is defined as a type of functional testing as it most often uses prepared test cases and verifies the conformance between system and requirements. The main difference compared with other functional techniques, it ensures that the major and the most critical functionalities (not full coverage) of the application are working properly. Some of previously examined techniques can be automated and used as automation testing tools, but this approach will be discussed later.

The differences between the main techniques enumerated before are illustrated in a table (see Annex 3 , page 72). These testing types are based on white-box (structural) or black-box (functional) techniques, however the third category can be subtracted as well - non-functional testing. Indeed, this category is not a part of test design based techniques as it not requires test cases. Non-functional testing focus more on aspects of the software that may not be related to a specific function or user action. Non-functional testing includes the various types; the main activities are as follows (Graham et al., 2008):

- Usability testing - as it was observed before.
- Maintainability testing - with refers to quality factor "maintainability".
- Portability testing - with refers to quality factor "portability".
- Compliance testing - it verifies, whether the software meets the defined IT standards by the company.
- Performance testing - "Testing conducted to evaluate the compliance of a system or component with specified performance requirements" (“IEEE Standard Glossary of Software Engineering Terminology,” 1990)
- Security testing - this testing is about to ensure the security mechanisms in the software, such as user data, user authority, privacy (Myers et al., 2011).
- Stress testing - "Testing conducted to evaluate a system or component at or beyond the limits of its specified requirements" (“IEEE Standard Glossary of Software Engineering Terminology,” 1990).
- Internationalization testing and Localization testing - these techniques tests the issues related with different languages used in a software. They verify whether the various languages and regions are adapted in a system and translations are made correctly (Graham et al., 2008). The correspondence

between non-functional testing and test levels are similar like functional testing - both of them can be performed at all levels (Graham et al., 2008).

This chapter presented testing levels and distinguished the main four levels, such as unit testing, integration testing, system testing and acceptance. Further, the two main categories of techniques - functional and non-functional - have been examined and then listed some testing types under each main category. As software goes through testing at each phase of SDLC, hence each of testing techniques can be applied at each level. It is applicable for both, functional testing and non-functional testing. Finally, the table is provided to show the differences between the main techniques.

2.4. Automated testing

All techniques of testing discussed in previous subchapters can be defined as manual testing because of human involvement in test execution with a purpose to ensure that software's behavior is as expected, while automated testing does the same thing, except the fact that some manual testing activities are automated by tools. In more specific terms, “test automation is the use of special software (separate from the software being tested) to control the execution of tests and the comparison of actual outcomes with predicted outcomes” (Huizinga & Kolawa, 2007). Indeed, manual testing is widely used comparing with automated testing. According to Mailewa et al. (2015) manual testing is applied more for smaller projects or for companies with limited financial resources; whereas, other enterprises see the benefits of automating some tests instead of running them manually. Mulder & Whyte (2013) states that software test automation could help to reduce testing costs and time dedicated for testing in software development. However, automated tests may not be useful if it is not applied in the right time, right context and with the appropriate approach. Implementation and maintenance of automated tests are expensive for company, thus more attention should be paid on when and what to automate (Garousi & Mäntylä, 2016). As Mulder & Whyte (2013) explains, wrong decisions made in selecting areas that should be automated, can lead to disappointments and major expenditures of software development, including human efforts and the cost of engagement automation tools, and automated testing sometimes is seen as a high-risk activity (Persson & Yilmazturk, 2004). The automation tools cover a wide range of activities and are applicable for use in all phases of the systems development life cycle. They could automate varies areas, some of them are as follows (Perry, 2006):

- **Executable specs**. This tool enables automatic execution of requirements specification. However, specification should be written in a such way that it can be compiled into a testable program.
- **Test data generator**. The main objective of this tool is to generate test data automatically for test purposes. It is useful foe large amounts of test transactions**.**

- **Tracing**. A representation of the paths followed by computer programs as they process data or the paths followed in a database to locate one or more pieces of data used to produce a logical record for processing.

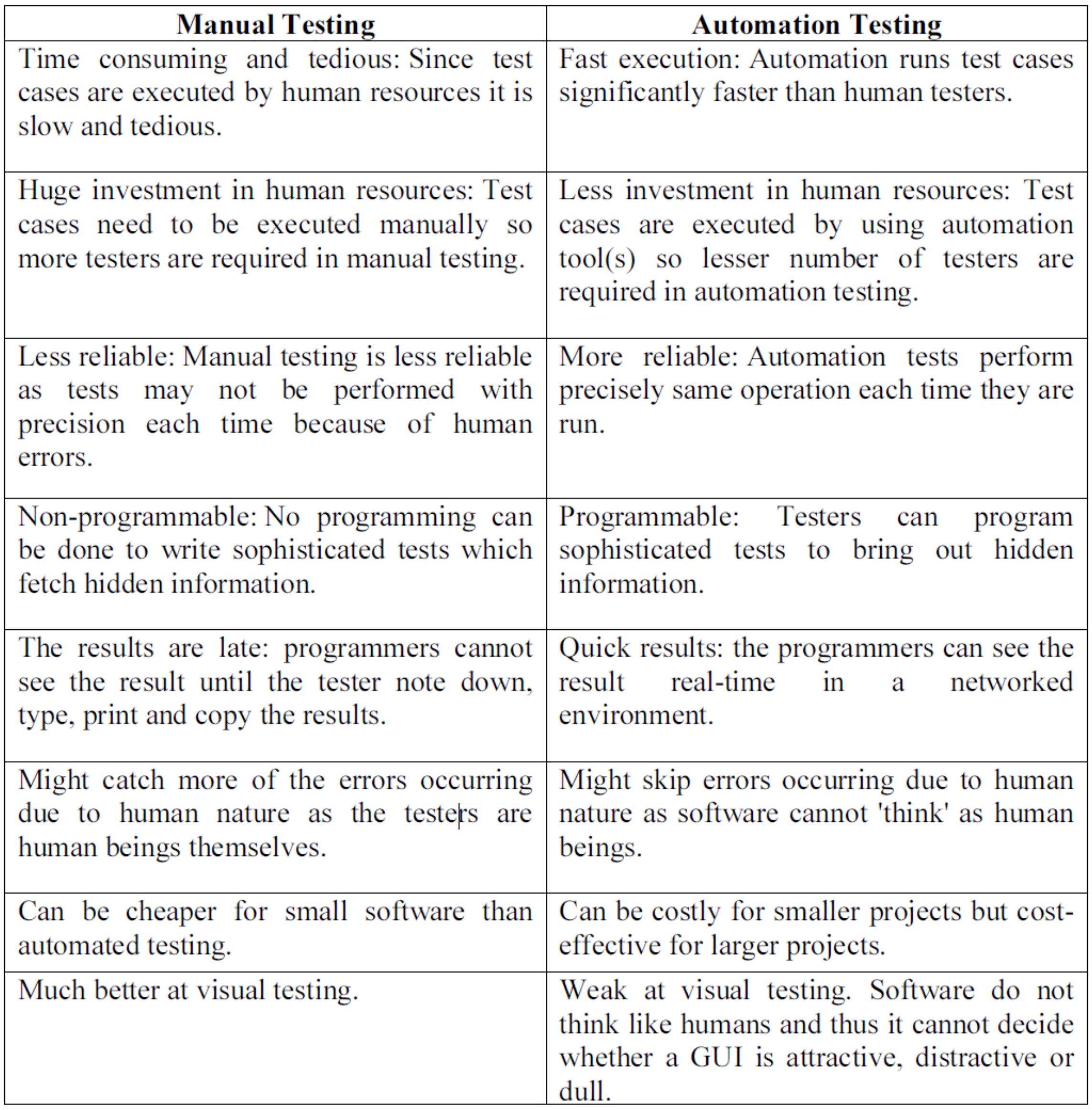
Although, besides these tools there are tools (e.g. "Selenium", "HP Quick Test Professional", "TestComplete", "LoadRunner") that helps to execute specified test automatically without human intervention. They mainly automate some testing techniques that we discussed before, for instance (Cem Kaner, 2014):

- **Function equivalence testing** - generating random input data and comparing the behavior of the function under test with a reference program.
- **Random regression testing** - system reuse already passed tests and then executes them in a random way automatically. Automated regression tests are useful in order to check whether the previous functionality are still working on every daily build version after changes (Graham et al., 2008). Daily build can be generated by automated tools as well (e.g. by Jenkins which creates a job to deploy an application every day with the newest changes in a code).
- **Hybrid performance and functional testing -** running the system under load and monitoring system responsiveness (performance testing) as well as behavioral correctness.

The survey conducted by (D. M. Rafi et al., 2016) showed the main benefits and limitations of test automation from practitioner's perspective. Practitioners explained that automation saves their efforts in test executions, and according them, tests can be reused as well as repeated again. The other advantage is seen when several regressions testing rounds are needed and regression test coverage is improved as well by automated tests (D. M. Rafi et al., 2016; Naik & Tripathy, 2008). Naik & Tripathy (2008) adds one more advantage - increased test effectiveness. Regarding the limitations, the high initial cost for automation and its tools are highlighted (D. M. Rafi et al., 2016) - Mulder & Whyte (2013) also agrees with these disadvantages. Furthermore, training the staff is considerable question as well. Most of practitioners argue that that current test automation tools offer a poor fit for their needs or they need more training on specific tool. Despite the limitations enterprises still think about full automated testing which helps to reduce human efforts (Garousi & Mäntylä, 2016). On the other hand, the full coverage of test by automated testing is impossible in practice due to budget and time constraints according to research conducted by Garousi, Coşkunçay, Betin-Can, & Demirörs (2014) in Turkey. This view is supported by survey made by D. M. Rafi et al. (2016) as well.

In order to achieve successful use of test automation, the enterprise should asses their capabilities to use such tools by analyzing following issues (Naik & Tripathy, 2008; Persson & Yilmazturk, 2004): the system should be stable and functionalities are well defined, test cases that need to be automated should be prepared correctly, adequate budget should be allocated for testing tools, test automation strategy should be defined clearly etc. Without enterprise assessment, automation process could be done in a wrong way which leads to failure of automation engagement. Furthermore, to understand better test automation purpose, the differences between manual testing and automation testing are distinguished and illustrated in a table (see Table 9, page 42) below.

Table 8. The difference between Manual Testing and Automation Testing



Manual Testing	Automation Testing
Time consuming and tedious: Since test cases are executed by human resources it is slow and tedious.	Fast execution: Automation runs test cases significantly faster than human testers.
Huge investment in human resources: Test cases need to be executed manually so more testers are required in manual testing.	Less investment in human resources: Test cases are executed by using automation tool(s) so lesser number of testers are required in automation testing.
Less reliable: Manual testing is less reliable as tests may not be performed with precision each time because of human errors.	More reliable: Automation tests perform precisely same operation each time they are run.
Non-programmable: No programming can be done to write sophisticated tests which fetch hidden information.	Programmable: Testers can program sophisticated tests to bring out hidden information.
The results are late: programmers cannot see the result until the tester note down, type, print and copy the results.	Quick results: the programmers can see the result real-time in a networked environment.
Might catch more of the errors occurring due to human nature as the testers are human beings themselves.	Might skip errors occurring due to human nature as software cannot 'think' as human beings.
Can be cheaper for small software than automated testing.	Can be costly for smaller projects but cost-effective for larger projects.
Much better at visual testing.	Weak at visual testing. Software do not think like humans and thus it cannot decide whether a GUI is attractive, distractive or dull.

Source: (Mailewa et al., 2015)

To summarize automated testing, the main features are defined. The main difference between manual and automated testing is that manual testing uses human intervention in test execution with a purpose to ensure that software's behavior is as expected, while automated testing does the same thing, except the fact that some manual testing activities or techniques are automated by tools. The main benefits are as follows: automation saves their efforts in test execution, tests can be reused as well as repeated again, the coverage of automated regression tests is improved. However, high initial cost for automation and its tools are highlighted.

2.5. Use of Software Testing techniques in Enterprises

Enterprises use a variety of software testing techniques that are tend to improve software quality. Moreover, they should help testers on designing precise test cases and executing them more effective. However, we think some techniques are depreciated, while others are applied often, because users are used to use them. In fact, there is no consensus on which technique is the most effective and appropriate to use; it depends on context. On the other hand, some factors could influence the decisions about which technique to choose. The majority of factors are presented by Vegas et al. (2002) in a table (see Annex 4, page 73) and some of them are listed below by Graham et al. (2008):

- **Models used in developing the system** – appropriate technique can be chosen by models that are used to develop the current system. For example, state transition testing is an appropriate technique to use for a state transition diagram included in specification.
- **Similar type of defects** – knowledge of the similar kind of defects (found in previous levels of testing or previous version of software) prompts to apply the same technique as the defect was detected (e.g. regression testing).
- **Risk assessment** – the greater the risk (e.g. safety-critical systems), the more formal testing. technique should be used.
- **Customer and contractual requirements** – sometimes customer specifies the particular testing techniques to use (most commonly statement or branch coverage).
- **Type of system used** – for example, "a financial application involving many calculations would benefit from boundary value analysis".
- **Regulatory requirements** – some industries should use specified testing by techniques regulatory standards. For example, "the aircraft industry requires the use of equivalence partitioning, boundary value analysis and state transition testing."
- **Time and budget of the project** – limited time and budget make to apply the techniques that are known the best for detecting more defects.

Indeed, time and budget of the project are very important issues which affects all SDLC. In fact, it is stated that testing activities are more costly comparing with other activities of SDLC. In 1979, it is seen that approximately 50 percent of the elapsed time and more than 50 percent of the total cost of project management budget is allocated for testing (Myers et al., 2011). While a later survey performed in 1994, shows the decrease of cost spent for testing in a whole software development process: about 24% of the overall software budget and 32% of the total cost of project management budget (Perry, 2006). According to World Quality Report (Hans van Waayenburg & Raffi Margaliot, 2016), the steady growth of quality assurance and testing budgets is seen since 2012. In addition, on average the enterprises are now spending 31% of its information technology budget on testing, compared with 35% in 2015, 26% in 2014, 23% in 2013, and 18% in 2012. In order to reduce the cost, more attention should be paid on the first stage of SDLC. This aspect was presented by McCall et al. (1977) as a guidelines in how to objectively specify the desired amount of quality at the system requirements specification phase. The research introduces into software quality factors and QA activities as it was already discussed in the first chapter.

Although, there are more studies conducted as a guide for testing to determine the best testing practices. For instance, the one by Bertolino (2007), Glass, Collard, Bertolino, Bach, & Kaner (2006), Vegas, Juristo, & Basili (2002) and Juristo, Moreno, & Strigel (2006). These research studies presented very significant amount of knowledge on good testing practices. The researchers determined the importance of the elicitation of main testing goals, management of testing processes, identification of test criteria on selection of appropriate testing technique. In fact, such knowledge can help to manage and improve software testing practices effectively and efficiently. Additionally, similar studies are prepared by Ng, Murnane, Reed, Grant, & Chen, (2004), Causevic, Sundmark, & Punnekkat (2010) and Lee, Kang, & Lee (2012), except the fact that the authors are using qualitative and quantitative methods instead of theoretical data gathering methods. The survey (Ng et al., 2004) was conducted to study the software testing practices in Australia. The research identified the major testing activities performed in enterprises: designing test cases, documenting test results, re-using the same test cases after changes were made to the software. Almost all surveyed enterprises agreed that formal tests were performed to ensure the developed software meets its requirements and specifications and they suggested to use more user acceptance testing. Regarding the defects statistics, it was found that between 40 to 59 % of such faults were related to specification defects; thus such amount of defects increases the cost of bug-fixing. Moreover, if those bugs were detected in later phases of SDLC, the more significant increase of cost is seen as defects become faults. In such case, more attention should be paid in the first stage of SDLC during validation of requirements specification as we discussed in the first chapter. Further, the most critical barrier to adopt specific testing technique was reported as a lack of expertise, while the adoption of automated tools is seen as costly to use. Despite these facts, a bit more than half of surveyed enterprises stated that they have automated some of their testing activities. While the regular staff training on automated testing and other issues related with software testing was provided only in some enterprises. Most of enterprises agreed that the main reason is cost for such training.

Another published research study (Causevic et al., 2010) presents results of an industrial survey on contemporary aspects of software testing. Their study gives crucial information about discrepancies observed between the current practices and the perceptions of respondents which could prove beneficial in shaping future research on software testing; however, we believe that the explanations for these observed discrepancies were provided based on researchers assumptions, or in some cases the explanations were not defined clearly. The later survey (Lee et al., 2012) investigates the state of software testing practices in terms of software testing methods and tools with a view to identify: current practices, perceived weaknesses and needs for additional capabilities of software testing methods and tools. The research showed that a half of test is executed manually, while a bit less is automated by tools. Comparing test levels by their use, almost the same percentage is devoted for integration testing and system testing, whereas, unit test and acceptance test are not very popular to use.

Furthermore, some researchers noted that while using an appropriate testing technique, test cases creation and prioritization (Elbaum et al., 2002; Rothermel et al., 1999; Srivastava, 2008) are also considered as a crucial part of software testing. Chang, Liao, Chapman, & Chen (2000) provided a novel approach to generate test scenarios based on formal specification and usage profile. In fact, this approach was developed later, and the new framework of formal notation for requirement specification has been presented (Baig & Khan, 2011). The suggested framework should provide a complete software testing technique which is expected to be accurate, structured technique to test software at each step of software development process contrary to existing practice. Although, the research will give statistical results only after completion of the entire three modules of the study as the researcher presented the first, theoretical, part.

In spite of limited resources and rush to finish projects on time project managers are likely to reduce the testing activities (Galin, 2004). In fact, this can bring bad side effects on software quality, therefore to achieve benefit of software testing under limited resources, it becomes necessary to identify the best software testing practices and create a mapping between various existing software methods and tools.

The main conclusions of the use of software testing techniques in enterprise are made. It is essential to identify the main testing goals, test criteria while selecting the appropriate testing technique; thus such knowledge can help to manage and improve software testing practices effectively and efficiently. Some factors are enumerated that could influence the decisions about which technique is better to choose. The main factors are customer and contractual requirements, time and budget of the project, type of system used and tester's experience. The case studies of testing techniques are generalized. The main features of case studies are identified: almost a half of all faults were related to specification defects; thus more attention should be paid on the first stage of SDLC. Further, a half of test is executed manually, while a bit less is automated by tools. Finally, the need of training related with software testing is agreed by all surveyed enterprises, however, the regular staff training was provided only in some enterprises.

3. INTRODUCTION TO RESEARCH "THE EFFECTIVENESS OF SOFTWARE TESTING TECHNIQUES" METHODOLOGY

This chapter presents the theoretical framework of research methodology within a specific enterprise. We will introduce to the methodology of empirical study that examines the effectiveness of software testing techniques a specific enterprise. The research methodology, strategy, questions, and data collection methods will be presented as well as their justification and appropriateness to achieve the goal of our research. Further, we provide the validity of research, including the selection criteria of experts, limitations, and ethics. And finally, the main characteristics of selected experts are illustrated in a table.

3.1. Research Design and Method Selection

The research methodology and strategy attitudes. A qualitative research strategy was chosen, since it enables the exploration of a phenomenon within its context using a variety of sources of evidence and allows multiple aspects to be revealed and understood (McGloin, 2008; Yin, 2012). According to Bryman & Bell (2011, pp. 26-28), "qualitative research investigates on the understanding and interpretation of individuals regarding their social world which leads to the epistemological position of interpretivism". We seek to observe organizational case study and to construct theories based on interpretations. Thus, an inductive approach is used for this purpose. The case study research is an appropriate strategy for us as we tent to “explore in depth a program, an event, an activity, a process” and “collect detailed information using a variety of data collection procedures” (Creswell, Plano Clark, Gutmann, & Hanson, 2003). According (Yin, 2003) categorization of case study, we choose to adopt exploratory type as our research is used to "explore those situations in which the intervention being evaluated has no clear, single set of outcomes". Since our research objective is to explore the most problematic areas and potential improvements in software testing process, we seek to gather preliminary information that will help define the problems that are faced with during software testing and then hypotheses for later studies could be suggested. In addition, outcomes are not clear and consist of various problematic areas. Each problematic issue in software testing process depends on different project, processes used in specific team, management decisions etc. Our qualitative case study focuses on detailed analysis of specific enterprise which faces with undiscovered problems that delay software testing process. In order to explore the topic in depth and get wider spectrum of potential software testing limitations, all of research and development (hereinafter - R&D) teams are involved in our research. All teams have a common quality assurance process, because the software components of each team are very close related with other teams and the well known incident handling process (e.g. bug reporting) should be used for all teams. On the other hand, the internal processes used in different teams can vary slightly as long as they do not affect other team processes. Thus, wider population of case study respondents will give more insights on exploring the limitations and improvements in software testing process, including the effective use of software testing techniques and processes. However, some researchers (Baxter & Jack, 2008; Yin, 2003) criticize that qualitative case study is too subjective and it is difficult to generalize the findings to other context and to maintain it in terms of credibility and validity. Indeed, this research results are relevant only for specified enterprise and its R&D teams. However, the results could fit partly for other organizations if their processes are similar to our enterprise's processes and based on: 1) Agile methodology; and 2) The International Software Testing Standard ISO/IEC/IEEE 29119 ("ISO/IEC/IEEE 29119 Software Testing Standard," 2014). After evaluation of these factors it can be assumed that our research outcomes could give some relevant insights for other organizations. Furthermore, to overcome limitation of subjectivity as far as it is possible, we choose to use mixed method of qualitative and quantitative which is supported by Creswell, Plano Clark, Gutmann, & Hanson (2003). Bryman & Bell (2011) states that mathematical calculations can be applied to qualitative research as well. Thus, regarding this point, effectiveness of software testing techniques in terms of limitations and improvements are explored combining both qualitative case study method and quantitative document content analysis method.

Case study research is based on experts interview which was provided as semi-structural questionnaire. Few reasons for selecting this method are as follows: experts have more knowledge on specific field (software quality assurance and testing techniques) and they could provide a more comprehensive view on specified topic. Further, a semi-structural questionnaire is chosen to investigate only specific fields of problematic issues, as it gives detailed insights on desired topic. Our research focuses mostly in such specific areas: test levels, test techniques, test process management. We have chosen only these areas in order to avoid a vast of problematic issues, starting from employees characteristics and ending at process management in organization globally. Regarding the quantitative part of research, we use the analysis of statistical documents of enterprise. These documents provide trends on defects reported by different teams. After documents are analyzed and experts are surveyed, we can combine both methods. This approach is a combination based on results interpretations of experts survey and generalized insights of statistical documents. This is not even interpretations of experts results, but the statistical proof as well. Thus, compromising the subjective interpretations (including our interpretations and the thoughts of experts), the mixed qualitative and quantitative methodologies have been chosen.

The validity and reliability of research. The objective of our research is to explore the most critical issues related with software testing in specific areas, as we discussed before. To achieve this objective the experts interview method is used for collect research data. As this method belongs to qualitative methodology, the objective approach should be followed (Yin, 2012). Firstly, according to Bryman & Bell (2011), "derived findings through qualitative research rely considerably on the researcher and its assessment". To avoid the subject interpretations of research findings, we use the mixed qualitative and quantitative methodologies by analyzing our findings with statistical documents as we discussed before in more detailed. Secondly, to ensure the objectivity of experts, we asked to provide formal answers as representative person of all team. In addition, to conduct the valid research, it is essential to elicit the appropriate experts so that all of them would have the common knowledge on the same field, such as: social, financial, physical areas (Kardelis, 2007). Thus, the main criteria for selection of experts have been applied: bachelor or higher degree on IT field, more than 3 years experience as a QA test specialist and not less than 1 year experience in the current team of selected enterprise, one or two expert corresponding each R&D team. We seek to conduct data from all R&D teams that consist of 2-3 QA test specialists in average, only one team has one test specialist. After evaluation of the characteristics of all QA test specialists (16 employees), 7 experts have been selected for our research. Only one selected expert did not satisfy all criteria (overall experience as a test specialist is 2 years), however he was selected as the best expert corresponding his team comparing with other team members. We will seek to get the wider observations from experts as they are professionals on their field.

The selected experts have been introduced into our research purpose and problem. And all of the 7 experts agreed to take a part in our research. After one week they received the semi-structural questionnaire. The questionnaire was provided online (http://www.questionpro.com/). The interviewers provided the answers after 3 days. Some of answers were not fully answered so we asked to add more insights in specific questions. After all questionnaires were completed we analyzed data by both the online tool (http://www.questionpro.com/) and MS Office Excel.

The limitations of research. One of the possible limitations of the investigation is the limited number of experts who are available for interviewing. The all come from different professional backgrounds, their level of experience is dependent on team, type of project implemented, and subordination level. Their domain of work might be slightly different because of highly nuanced nature of each step of software development process. There are also differences in procedures which each of these experts execute in their teams. These factors might distort some information gathered from the experts during the interviews. We would like to stress out that the enterprise is dependant and relying on the professionals which it employs and data information gathered from the experts is only applicable to certain case scenario and should not be evaluated as transferable to other organizations at full scale.

The second limitation arise from the instrument used for investigation. Informants were provided with questionnaire which is included open and closed questions. Since the interview based questions were provided in straight forward nature we did not had a chance to ask additional questions which could arise from the answers of the experts in a real time. However, after the answers were received we asked the interviewees to provide some additional information to some questions.

Third limitation is related to the nature of quantitative questions. These were given to very few respondents. Since the population is low, it is impossible to generalize the results for broader population.

The ethics of research. According to Miller (2012), it is essential to inform informants about research problem and purpose and use ethical elements. Regarding this, we provided the required information about research process, main goals, and research problem to our informants. All questionnaires were given to informants personally by email in order to ensure that no information sharing has been made between them and anonymity of each of them is not revealed. In addition, confidentiality principle is one of the most important which ensures that the information about enterprise processes, teams, and clients remain unrevealed. To achieve this principle and gain trust of informants, we gave other names to teams and informants. Enterprise name is not used in this research as well. Thus, our experts were informed about all process of empirical study and their anonymity were ensured.

Research questions. Some questions are raised for the research in order to explore the effectiveness of software testing techniques in terms of limitations and improvements:

1. What are the main software testing techniques used to ensure software quality in an enterprise?
2. How to improve the use of software testing techniques and testing processes in order to ensure software quality?
3. What are the most problematic areas in software testing process? How do these aspects affect software quality? What process of these areas can be automated?
4. What topics on software quality assurance are lacking?

3.2. Setting and Participants

The selected company (hereinafter - Company) specializing in developing software core systems for insurance companies was chosen for this case study. Company is headquartered in San Francisco, USA, and its offices are in different locations: Lithuania, Belarus, China, Japan, Brazil, Australia and New Zealand. The medium-sized (according to EU SME categorization) Company in Lithuania has been chosen to take part in our research due to several reasons. Firstly, it is the main office which focuses on core system development, while others work directly for client's needs by customizing and maintaining the system. Secondly, QA team are more specialized as their activities are separated from developers and business analysts. Thirdly, the core system requires more testing because of vast of new functionality implementation. And finally, all R&D teams are involved in software development, and each of them specializes on different elements (component) of software. Nevertheless, their processes are very related with each other, so the communication between them are continual. In spite of these facts, the common processes are used for all teams to ensure quality of software. As we discussed before, each team has a slightly different process on software testing, including the use of different techniques and tools. We will seek to explore the main similarities between them and the main limitations that they are facing with. Thus, the main characteristics of experts that participated in our research are presented in a table below (see Table 9, page 50). The changed names of teams and experts will be used for further conducted data analysis.

Table 9. The Experts characteristics

	R&D Team	**The current position**	**Overall work experience as a Test Specialist**	**Experience in a Company**
Expert A	Team 1	Recently as Business Analyst (before QA Senior Test Specialist)	3	4.5
Expert B	Team 2	QA Test Specialist	5	1
Expert C	Team 3	QA Test Specialist	2	2
Expert D	Team 2	QA Test Specialist	3	3
Expert E	Team 4	QA Test lead	7	4
Expert F	Team 5	QA Test lead	3	2
Expert G	Team 4	QA Test Specialist	3	3

Prepared by author.

This chapter includes research methodology, strategy, data collection methods, including validity, reliability, limitations and ethics criteria. To sum it up the case study was performed in specific organizational setting and had mixed methodological approach. Experts were chosen according to their years of practice in specific domain, representation of software development team in the enterprise. Each of the experts had to have a higher education degree in the field of informatics. Additionally some quantitative data were collected from the documentation of the company. In the next chapter we are going to analyze qualitative and quantitative information in order to answer research questions.

4. RESEARCH "THE EFFECTIVENESS OF SOFTWARE TESTING TECHNIQUES" RESULTS

This chapter presents empirical study results that are conducted during interviewing. The results of quantitative questions are presented in charts, while qualitative - are analyzed in textual form. Generalized experts thoughts and analyzed statistical documents are presented as a systematic view on software testing processes in order to answer research questions.

4.1.Results Analysis of the Effectiveness of Software Testing Techniques

In the beginning of questionnaire (see Annex 5, page 75) experts were asked to specify the exact number of years that shows their experience in a software testing field. During the selection of experts by criteria, the list of QA test specialists were provided by Company. Only general information about overall experience in Company and their education were seen. In order to get more detailed information about the work experience, in the beginning of questionnaire (see Appendix , page) we asked the experts to specify the exact number of years of their overall experience as test specialist as well. The most experienced expert in software testing field is Expert E: overall experience is 7 years and 4 of them - in our surveyed Company as a QA test lead. The position QA Test lead shows the highest competence of tester. It includes the advanced testing skills using different testing techniques, managing and improving test process. Especially, the technical knowledge on automated testing and software domain are highlighted. In addition, QA test lead manages his own QA team and makes decisions together with project manager which is responsible for all processes of software development. The Expert A has also a significant experience on testing (5 years), however only 1 year as a part of our Company. Other experts have quite similar experience in Company and overall. According Company regulations, the adoption process for new hired testers are about 1 year. Thus, all of our experts have completed this period and it can be assumed that they have enough knowledge on software domain and processes used in SDLC at this Company.

Furthermore, in order to answer the first question of our research we need to identify the main techniques used for QA testing in R&D teams, we need to separate techniques used by testers and developers: testing types and static analysis respectively. First of all, we asked the question "*What types of testing techniques are used in your team?"*. The general trends are illustrated in a chart below (see Figure 6, page 52).

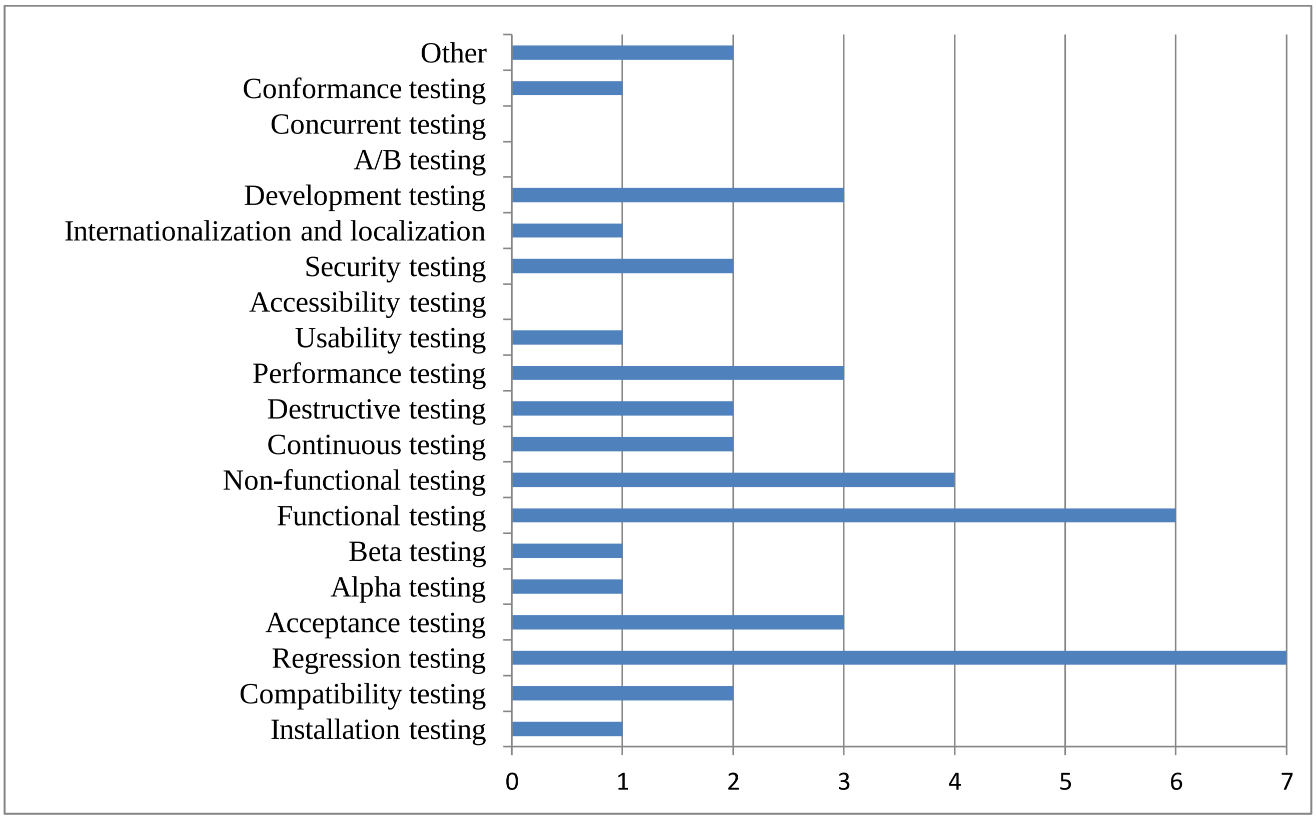


Figure 6. The trends of testing techniques used in R&D teams

Prepared by author according to empirical research results.

The experts' answers revealed that all teams are using regression testing to ensure the reliability of each software release. The second and third popular techniques are functional testing and non functional testing. Each of them are tended to discover specific types of defects. Further, performance development, and acceptance testing techniques are selected. Experts from Team 2 and Team 3 have chosen acceptance testing. The Expert B added a comment *"Our team is not using this technique directly, but other team is doing acceptance testing for our team"*. So we got new relevant information. Whereas, some experts mentioned additional testing techniques that were not added in a list. Expert A stated that his team uses *"Integration testing"* as well. Expert G added *"Exploratory and smoke testing"*.

Furthermore, we sought to identify what static analysis techniques are used in teams. Thus, the most used technique is Code or Design Inspection (see Figure 7, page 53). Second and third place belong to and Review (peer) and Review (technical) respectively. According to research results, Review (management) and Walk-through are not used at all.

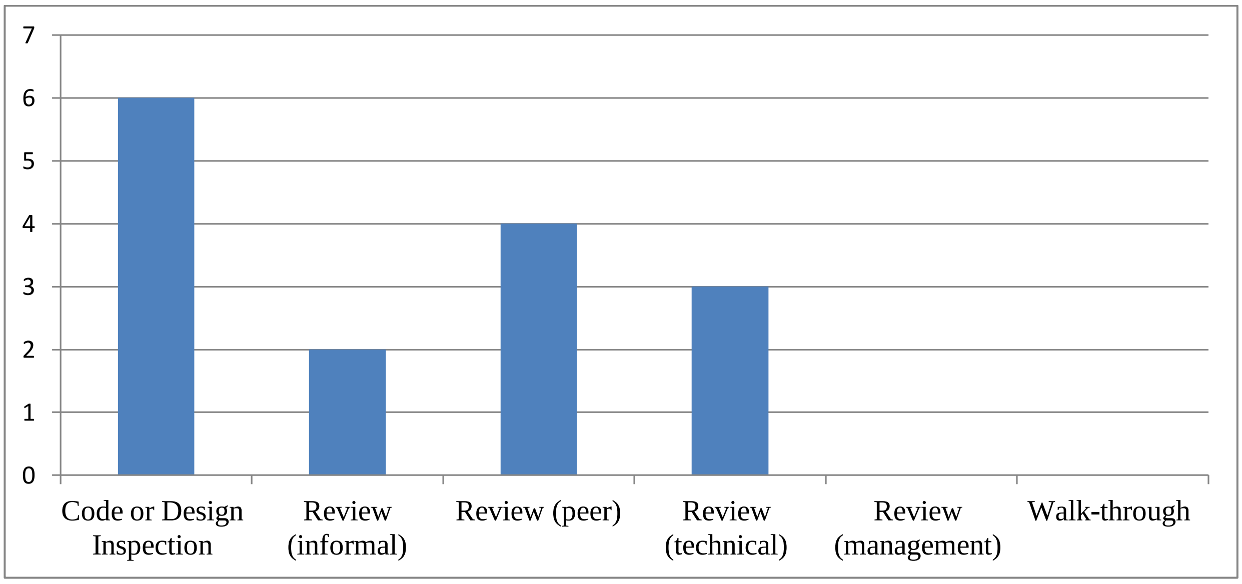


Figure 7. The trends of static testing techniques used in R&D teams

Prepared by author according to empirical research results.

After the first research question is answered, we need to focus on problems that teams are facing with during software testing process. Such problems include not even processes related with testing activities, but external issues that can affect team work. For example, organizational decisions related with planning time and QA team capacity may affect whole SDLC. If QA team are not available to handle all issues, the quality of software becomes poor and the price of software increases for few reasons. Firstly, if defects are detected after product (software) is delivered to customer, customer should report about defect - it takes time for him and his satisfaction is low, since it cannot use the current functionality that he needs. Secondly, reported defect will go through management side in order to confirm the validity of it, and then to bug fixing stage. Thus, only one issue requires all team involvement and efforts that cost. So it is better to discover defects at earlier stages of SDLC when the cost of defects are relatively low comparing with previous example. To overcome this, we need to discover what are the main problems that delays software testing and decrease the quality. These questions are related with research questions starting from to 2 to 4. We will try to answer by experts insights and statistical documents content analysis.

The sixth question of questionnaire *"Prioritize the following issues of software testing that should be used more effective in your team. Start from the highest point that should be improved."* uses the ranking scale which enables survey experts to rank a set of problematic issues from highest to lowest – most important to least important. The priorities are illustrated in a chart below (see Figure 8, page 54). The most problematic issues (1 place) that should be improved are as follows: Time, Test Techniques, Incident handling, Test design and Test Management (test goals, process). Further, 2 place is devoted for Test design, Time, Incident handling and Test Execution.

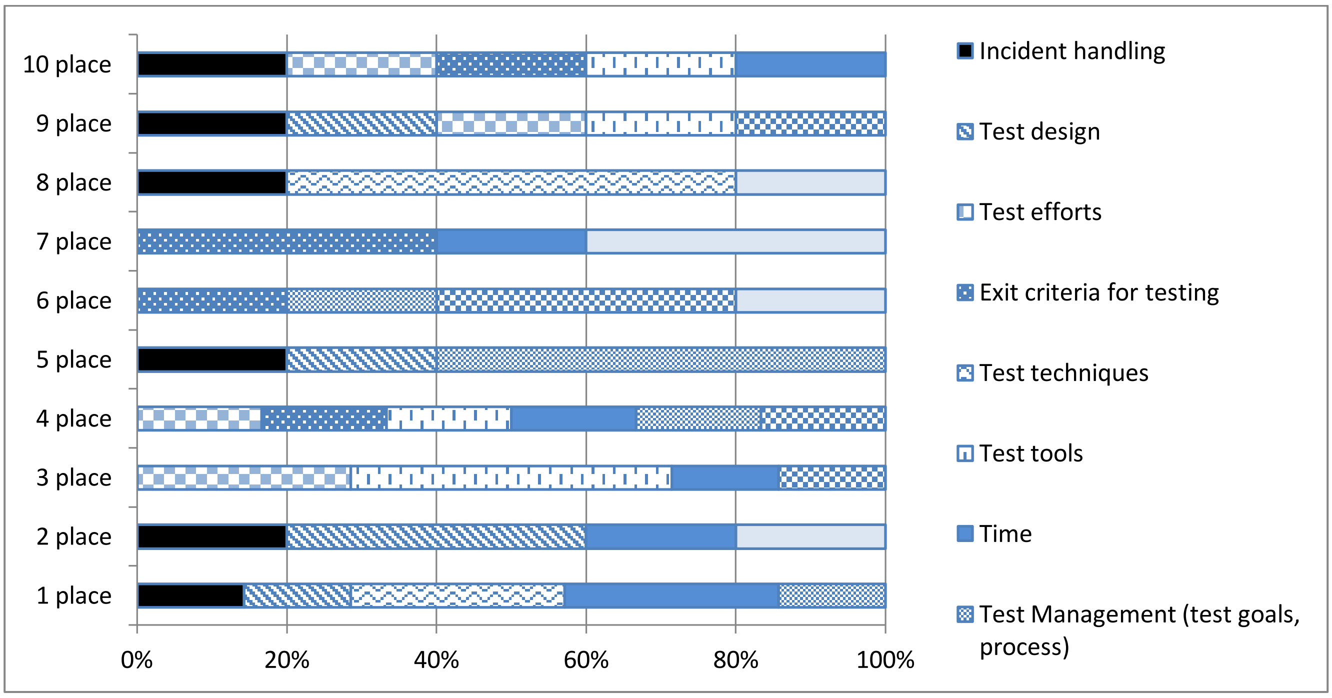


Figure 8. Problematic issues in software testing that needs to be improved by Experts

Prepared by author according to empirical research results.

In the third place, we can see Test tools, Test Efforts, Time and Test Planning. 2 experts, Expert B and Expert D, identified Time as a critical issue, while others, Expert C and Expert F - distinguished Test Techniques. Expert A states that his team needs to improve Incident handling activity, Expert E - Test Management (test goals, process), and Expert G - Test design. All experts agree that Time is always the most critical issue that, actually, could affect the software quality the most. If the defects are delayed to be fixed, other software components suffer as well because of close relationship between them. QA team are also stuck, since they cannot continue testing while the particular functionality is blocked by unfixed defect. In statistical documents (see Figure 9, page 54), we can see the main trends of overdue defects.

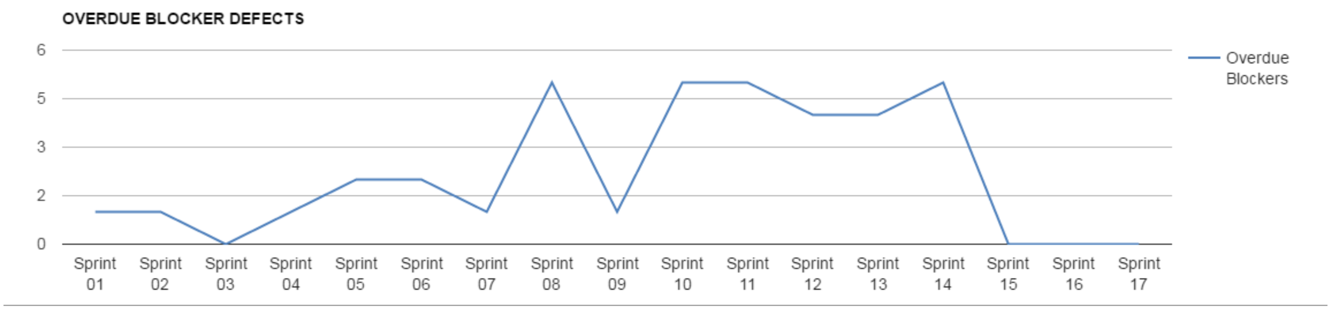


Figure 9. The trends of all overdue Blocker defects in different Sprints

Source: Company's statistical data

These defects has the highest level of criticality (also called as faults). It means that these defects make the biggest harm to the system and block specific functionality. These defects should be fixed as soon as possible. They should be detected in development phase while the harm to system is lesser. So time is very important factor for such defects.

Continuing the topic of problematic issues, we asked the experts *"What are the most critical problems in some or all different areas? Why? Give some examples below after selection."* in order to identify the area in terms of test levels, test types. The results are presented in a chart below (see Figure 10, page 55).

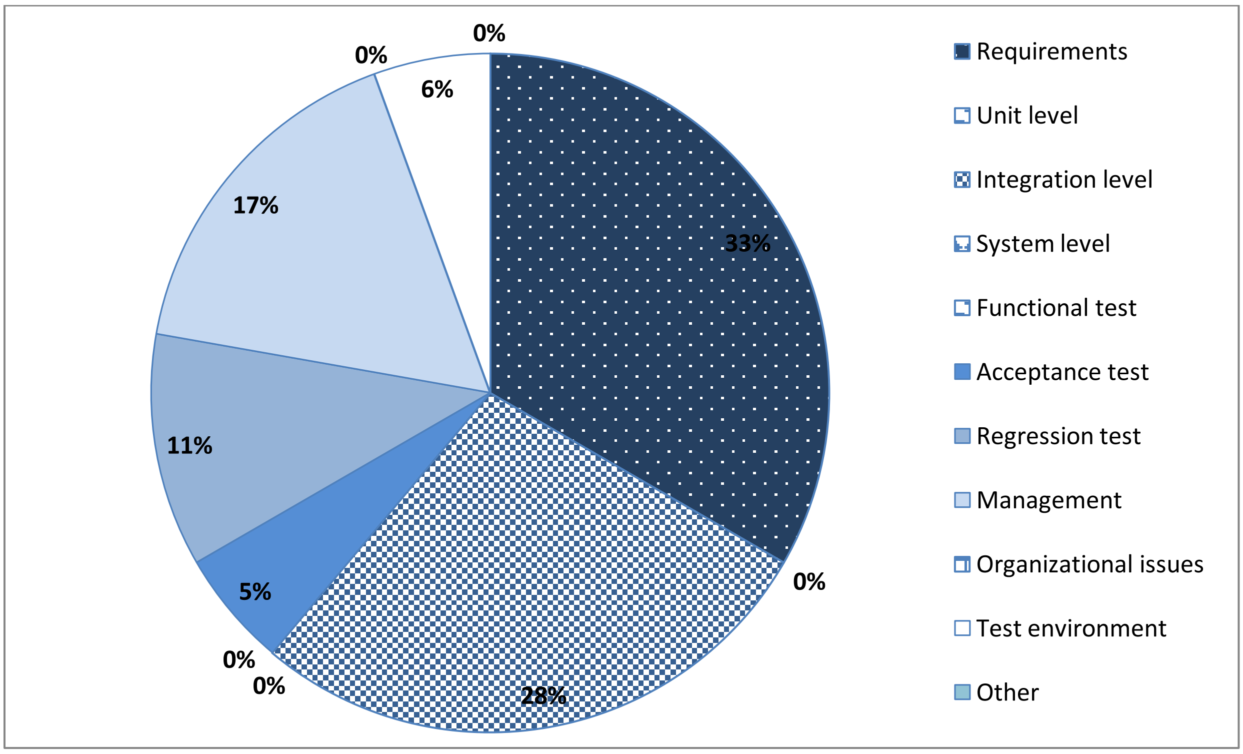


Figure 10. The most critical areas in software testing by Experts

Prepared by author according to empirical research results.

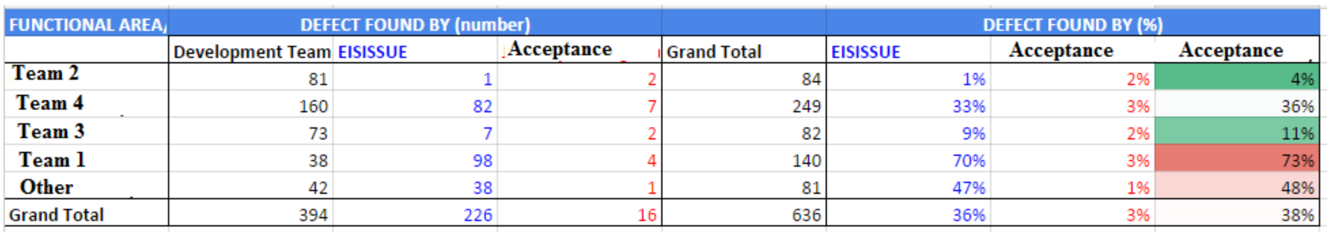
The most critical area "Requirements" was defined by Experts. Expert B argues that *"Requirements are changing during implementation phase and a lot of updates for test design is needed"*. Expert C, Expert D, Expert E agree with statement about changing requirements and add more additional arguments. Expert C states that *"The requirements are not clear and changes during testing"*, whereas Expert D provides his opinion: *"Too high level requirements. Usually changing according to implementation because of poor initial analysis"*. Expert E also agree by saying that *"Requirements change is a problem for Team 4. Agile methodology doesn't imply requirement change during the sprint. Changing requirements delay development and test design phases"*. Expert F also complains about requirements, but provides different view that other experts. He states that *"We are testing bugs from different components , so we have to know a lot about each component to be capable catch bugs. Sometimes it's hard to find out what are expected results of system behavior."* An finally, Expert G defines the problem related with mistakes in Requirements by saying that *"Defects* *introduced in this phase are most difficult to fix and are also most expensive"*. Our investigation of scientific literature discovered this problematic issue as well.

The second most problematic area is integration level. Experts share their opinions about limitations in this test level. Expert A identifies the relationship between components: "*Too much components related with other teams. Every change affects almost every team"*. Expert B states his own arguments that are related with Expert's A statement: *"Team merge problems during integration. Unresolved issues of other teams delay integration testing"*. Expert D also states his opinion on components relationship as *"Poor communication with other teams, everyone's trying to lower their workload by increasing other teams workload"*. And Expert C sees the problems of components relationship when communication is poor between teams: *"It is not clear what should tested when cross feature functionality is integrated"*. And finally, Expert E gives a comprehensive answer to identifying the problematic issues: "*Integration testing is performed on very initial level using stubs, it's supported only for Team 4 and Team 2 components, though team x, team y also require integration testing. Integration is a weak element in Company, because necessary integration testing is not provided, but we should remember that clients don't use separate components (in the majority) but the whole integrated system."*

The Management is identified as the third problematic area. This area includes decisions making, planning activities, overall satisfaction of employees. Expert B concerns about planning issues by saying that there are *"Problems with planning. The most critical items come in testing almost the last week of sprint."* Expert D agrees with previous statement and provides few examples: *"Poor planning which gives a lot of space for unintentional error. Too much time is spent for low priority items and occasionally high priority tickets appear mid-sprint. Time spent analyzing and designing test for improvements which are later forgotten and closed as not actual anymore is time spent in vain."* Expert E also agrees with Expert B and Expert D. and explains how the management decisions affect all SDLC, software quality and the satisfaction of employees: *"Weak management, planning without risk prediction, absence of any statistics and ignoring previous experience lead to delays on different SDP stages, low product quality, great number of bugs from a client, acceptance team (and increasing time to bug fixing accordingly) and as a result - motivation decrease from team side."*

Some other problems are distinguished in Acceptance testing level. Expert D states that *"Acceptance testing is usually just step by step test execution with no understanding of the functionality or requirements (with some exceptions)."* As we can see in a table below (see Table 10, page 57), the ratio of defects found by Acceptance testing is very low for all teams - only 1-3 %. We can assume that Expert's D opinion could be reasonable, as if the acceptance team do only step by step acceptance testing, they are not able to discover more defects.

Table 10. Defects found by Acceptance testing camparring with R&D teams



FUNCTIONAL AREA	DEFECT FOUND BY (number)				DEFECT FOUND BY (%)		
	Development Team	EISISSUE	Acceptance	Grand Total	EISISSUE	Acceptance	Acceptance
Team 2	81	1	2	84	1%	2%	4%
Team 4	160	82	7	249	33%	3%	36%
Team 3	73	7	2	82	9%	2%	11%
Team 1	38	98	4	140	70%	3%	73%
Other	42	38	1	81	47%	1%	48%
Grand Total	394	226	16	636	36%	3%	38%

Source: Company's statistical data

After definition of problematic areas, some questions about improvements on testing process have been provided to experts. 3 experts argued that regression testing should be automated when we asked *"What testing processes or test types should be automated? Why? Give some examples."* Expert A: *"Regression testing. More attention should be paid on checking the old functionality after new improvements."* Expert B: *"More regression tests should be automated because old functionality becomes buggy after new improvements."* Expert E: "*Regression testing should be automated as much as possible and regression tests should run on nightly basis in order to find regression issues. Complex regression tests can be run on back ported version, or on central environment instead of central manual testing."* Some other experts talked about the same issue - end to end testing that are used to test whether the flow of an application right from start to finish is behaving as expected. The purpose of performing end-to-end testing is to identify system dependencies and to ensure that the data integrity is maintained between various system components and systems. Expert C states that *"Tested data preparation and 'end to end' tests"*, whereas Expert D identifies more issues related with end-to-end testing: *"End to end testing and integration level testing. End to end testing should be automated so that new functionality would not introduce major+ bugs. Integration tests should be automated so that one team would not depend heavily on other teams mistakes."* Expert G gave different opinion comparing with all other experts: *"Critical areas, scenarios of bugs that are being reopened several times after fixing, functional tests - because continuous integration changes tend to break software."*

Finally, we sought to identify that knowledge or skills QA team would like to improve. The question *"In your opinion what topics related with Software Testing should be included in internal or external training/courses in deeper detail?"* has been provided. Most of the experts, Expert A, Expert B, Expert C and Expert F, thinks that their teams need to get more information about Tools for testing. Expert C adds more additional improvements: *"Tools which helps to test application. The theory is not used in practice. Moreover, more testable areas should be covered such as PHP C++ Python"*. *"Fundamentals of testing and testing life cycles"* should be included in courses by Expert D. Expert E identified 3 different aspects related with software testing: *"1) Software Development Process, types of testing (functional/not functional); 2) Test designs creation; 3) Testing approaches and techniques"*.

While Expert E agrees with the third statement of Expert E and he provides the reasons why it would be useful by saying that *"Testing techniques and testing tools, for every tester it would be useful and interesting, to know something more about such things and try to apply them in work."* Expert G adds not mentioned topics: *"Functional testing, risk based testing."* These topics would help to learn more about testing fundamentals that are required for qualified QA test specialist.

To sum up result from empirical research, the main problematic issues are distinguished. The most critical areas in software development phase are Requirements, Integration level and Management. Requirements that are changing a lot during software development stage. They are too high level and complex and mostly updated according implementation. Integration is a weak element in Company and. Poor communication with other teams is seen. As the software has a lot of relationships with other components of different teams, it needs to be tested more effective. The acceptance testing is also distinguished. The ratio of defects found by Acceptance testing is very low for all teams. The one of possible reasons could be: the acceptance team do only step by step acceptance testing, so they are not able to discover more defects. Regression testing should be automated as much as possible and regression tests should run on nightly basis, back ported versions and on central environment instead of central manual testing in order to find regression issues. Most of the experts agreed that their teams need to get more information about Tools for testing and techniques.

THE CONCLUSIONS

The purpose of this thesis was to fill in the gap in knowledge about the most critical problems in software testing process, including test levels, test techniques in order to improve software quality. The conclusions of our case study are as following:

1. After scientific literature analysis and generalization of both software testing and quality assurance, the relationship between these concepts are distinguished. The quality assurance activities, such as verification and validation, are performed by different software testing techniques in order to discover defects and ensure compliance of software and user requirements - ensure software quality in general. The main differences between verification and validation are analyzed and provided in a table.

2. The main features of testing techniques are distinguished by analyzing scientific literature and empirical studies by researchers. Techniques are categorized into four parts: static and dynamic analysis, test design based techniques, testing levels and test execution types. Their use in enterprise has been analyzed by identifying main techniques and limitation that are faced with. Techniques fall into four parts: static and dynamic analysis, test design based techniques, testing levels and test execution types.

2.1. Static testing (without code execution) and dynamic testing (executing code) can be performed by tools, however, there are some limitations, as not all programming languages are supported and security issues are seen. Despite these facts, static testing reduces the chances of failures in later phases of SDLC and dynamic testing validates whether the software meets requirements.

2.2. Design based testing techniques, are broadly divided into white-box testing and black-box testing. Other techniques, such as, experience-based and defect-based, are used rarely. All these approaches focus on the sources of information for creating test cases. The black-box techniques design test cases based on the requirements specification, while white-box techniques - on deriving test cases directly from the internal structure of system. White-box testing and black-box testing techniques can be perform by static and dynamic analysis in order to find defects.

2.3. The main test levels and their corresponding test types are compared and their features are provided in a table. The main levels: unit testing, integration testing, system testing and acceptance. The relationship between test levels and techniques is seen: software goes through testing at each phase of SDLC, so each of testing techniques can be applied at each level.

2.4. The main difference between manual and automated testing is identified. Manual testing uses human intervention in test execution with a purpose to ensure that software's behavior is as expected, while automated testing - does the same thing with automated tools (manual testing activities or techniques can be automated). The main benefits are as follows: automation saves their efforts in test execution, tests can be reused as well as repeated again, the coverage of automated regression tests is improved. However, high initial cost for automation and its tools are highlighted.

2.5. Use of testing techniques in enterprises involves the right selection of technique and effective use of them. Some factors are enumerated that influence the decisions about which technique is better to choose: customer requirements, time and budget of the project, type of system used and tester's experience. The case studies showed that almost a half of all faults were related to specification defects; thus more attention should be paid on the first stage of SDLC. Automated testing and manual testing is used almost in equal parts. However, the need of training related with software testing and automated tools is highlighted.

3. The case study investigation allowed to observe specific organization setting from qualitative and quantitative approach. The qualitative method was the leading during the study and relayed on expert interview. The informants were chosen according to certain criteria like job experience in specific software developments teams and education. Additionally, some quantitative data were collected from the documentation of the company to perform content analysis.

4. The results of case study revealed that the most critical areas in software development phase are:

4.1. Requirements, Integration level, and Management. Because of the constant change and complexity of Requirements, they are mostly updated based on software functionality in the real time. Integration level is a bottleneck in a Company and it faces the difficulties of proper communication among teams. Since the components of software are strongly interrelated, more of rigorous testing is required because defects in certain parts of the system may echo to the quality of overall product. The According to experts, Management level struggles to provide coherent plans and milestones for development cycle and this negatively impacts overall software quality. The acceptance testing is also pointed out by experts. The ratio of defects found during Acceptance testing is very low compared to overall statistics of defects. This situation arises from the fact that the acceptance team executes test cases only in step by step nature, so they unable to discover more defects.

4.2. According to experts, regression testing should be automated more frequently and regression tests should be performed on nightly basis. Whereas, complex regression tests should be run on back ported versions of software or central server instead of central manual testing.

4.3. Most of the experts agreed that their teams need to have more training on Tools and testing techniques used for testing.

RECOMMENDATIONS

Regarding the conclusions of our research we have some recommendations on software testing techniques improvements in quality assurance process. Further investigation of this case study are also will be recommended. The suggestions are as follows:

1. Requirements, Integration, and Management levels were highlighted as the most problematic areas, so they need to be improved involving all development team. We suggest simplification of Requirements in more understandable way and to improve knowledge sharing among business analysts. Corresponding to second aspect, which is requirements changing during all software development life cycle, we suggest to take more time efforts and human resources for planning the tasks. Communication level between business analysts and strategists should be improved in order to ensure the minimum dynamics of requirements.

2. We suggest to generate more automated test for regression testing to ensure the stability of the main functionality in each software release. Such tests should be executed during night in order to not disturb testing process during work hours.

3. Corresponding to training aspects, we suggest to organize more internal and external training for quality assurance testers. The following topics should be included in program: tools and testing techniques used for testing.

4. For further investigation of this case study, we suggest to conduct quantitative research in development teams. The questionnaire should consist of problematic issues identified by experts.

REFERENCE LIST

1. Aivosto. (2016). VB Watch: Profiler, Debugger and Protector. Retrieved December 4, 2016, from http://www.aivosto.com/vbwatch.html

2. Baig, M. M., & Khan, A. A. (2011). A Formal Software Testing Technique. *Pakistan Journal of Science*, *63*(4), 194–196.

3. Baker, C. (1957). Mathematical Tables and Other Aids to Computation. *Digital Computer Programming*, *11*(60), 298–305.

4. Baxter, P., & Jack, S. (2008). Qualitative Case Study Methodology: Study Design and Implementation for Novice Researchers. *The Qualitative Report*, *13*(4), 544–559.

5. Bertolino, A. (2007). Software Testing Research: Achievements, Challenges, Dreams. In *2007 Future of Software Engineering* (pp. 85–103). Washington, DC, USA: IEEE Computer Society. https://doi.org/10.1109/FOSE.2007.25

6. Boehm, B., & Basili, V. R. (2001). Software defect reduction top 10 list. *Computer*, *34*(1), 135–137.

7. Bryman, A., & Bell, E. (2011). *Business Research Methods 3e*. OUP Oxford.

8. Causevic, A., Sundmark, D., & Punnekkat, S. (2010). An Industrial Survey on Contemporary Aspects of Software Testing. In *Verification and Validation 2010 Third International Conference on Software Testing* (pp. 393–401). https://doi.org/10.1109/ICST.2010.52

9. Cem Kaner, J. D. (2014). Inefficiency and Ineffectiveness of Software Testing: A Key Problem in Software Engineering. Retrieved from http://www.kaner.com/pdfs/Top5SEissues.pdf

10. Chang, K. H., Liao, S.-S., Chapman, R., & Chen, C.-Y. (2000). Test scenario generation based on formal specification and usage profile. *International Journal of Software Engineering and Knowledge Engineering*, *10*(2), 185–201. https://doi.org/10.1142/S0218194000000110

11. Creswell, J. W., Plano Clark, V. L., Gutmann, M. L., & Hanson, W. E. (2003). *Handbook of Mixed Methods in Social & Behavioral Research*. SAGE. Retrieved from http://media.library.ku.edu.tr/reserve/resfall08_09/CSHS501_JDixon/Week5.pdf

12. Crosby, P. B. (1979). *Quality is Free*. New York: McGraw-Hill.

13. Dahl, O. J., Dijkstra, E. W., & Hoare, C. A. R. (Eds.). (1972). *Structured Programming*. London, UK, UK: Academic Press Ltd.

14. D. M. Rafi, Moses, K. R. K., Petersen, K., & Mäntylä, M. V. (2016). Benefits and limitations of automated software testing: Systematic literature review and practitioner survey. *Proceedings of the 7th International Workshop on Automation of Software Test*, 36–42.

15. Deutsch, M. S., & Willis, R. R. (1988). *Software quality engineering: a total technical and management approach*. Englewood Cliffs, NJ: Prentice Hall.

16. DeVolder, D., Ghazanshahi, S., & Zadeh, J. (2008). Software Testing and Quality Assurance. Presented at the The 12th World Multi-Conference on Systemics, Cybernetics and Informatics: WMSCI 2008. Retrieved from http://www.iiis.org/CDs2008/CD2008SCI/SCI2008/PapersPdf/S461JT.pdf

17. Di Tommaso, D., & Roche, F. H.-L. (2011). Unit testing as a cornerstone of SAS application development. *Pharmaceutical Programming*, *4*(1/2), 85–90. https://doi.org/10.1179/175709311X13166801334316

18. Elbaum, S., Malishevsky, A. G., & Rothermel, G. (2002). Test case prioritization: a family of empirical studies. *IEEE Transactions on Software Engineering*, *28*(2), 159–182. https://doi.org/10.1109/32.988497

19. Emanuelsson, P., & Nilsson, U. (2008). A Comparative Study of Industrial Static Analysis Tools. *Electronic Notes in Theoretical Computer Science*, *217*, 5–21. https://doi.org/10.1016/j.entcs.2008.06.039

20. Ernst, M. (2003). Static and dynamic analysis: Synergy and duality.

21. Evans, M. W., & Marciniak, J. J. (1987). *Software Quality Assurance and Management*. New York: John Wiley & Sons.

22. Fagan, M. E. (2001). Design and Code Inspections to Reduce Errors in Program Development. In M. Broy & E. Denert (Eds.), *Pioneers and Their Contributions to Software Engineering* (pp. 301–334). Springer Berlin Heidelberg. Retrieved from http://link.springer.com/chapter/10.1007/978-3-642-48354-7_13

23. Galin, D. (2004). *Software Quality Assurance: From Theory to Implementation* (1 edition). Harlow, England ; New York: Pearson.

24. Garousi, V., Coşkunçay, A., Betin-Can, A., & Demirörs, O. (2014). A Survey of Software Engineering Practices in Turkey. *ResearchGate*, *108*(October 2015), 148–177. https://doi.org/10.1016/j.jss.2015.06.036

25. Garousi, V., & Mäntylä, M. V. (2016). When and what to automate in software testing? A multi-vocal literature review. *Information and Software Technology*, *76*, 92–117. https://doi.org/10.1016/j.infsof.2016.04.015

26. Garvin, D. A. (1984). What Does "Product Quality" Really Mean? *Fall*, (26), 25–43.

27. Gelperin, D., & Hetzel, B. (1988). The Growth of Software Testing. *Commun. ACM*, *31*(6), 687–695. https://doi.org/10.1145/62959.62965

28. Gittens, M., Kim, Y., & Godwin, D. (2005). The vital few versus the trivial many: examining the Pareto principle for software. In *29th Annual International Computer Software and Applications Conference (COMPSAC'05)* (Vol. 1, p. 179–185 Vol. 2). https://doi.org/10.1109/COMPSAC.2005.153

29. Glass, R. L., Collard, R., Bertolino, A., Bach, J., & Kaner, C. (2006). Software Testing and Industry Needs - ProQuest. *IEEE Software*, *23*(4), 55–57.

30. Gong, D., & Yao, X. (2010). Automatic detection of infeasible paths in software testing. *IET Software*, *4*(5), 361–370. https://doi.org/10.1049/iet-sen.2009.0092

31. Graham, D., Veenendaal, E. V., & Evans, I. (2008). *Foundations of Software Testing: ISTQB Certification*. Cengage Learning EMEA.

32. Hambling, B., & Morgan, P. (Eds.). (2011). *Software testing: an ISTQB-ISEB foundation guide* (Rev. 2. ed., reprinted (with revision)). Swindon: British Informatics Society.

33. Hamlet, D. (1995). Software Quality, Software Process, and Software Testing. In M. Zelkowitz (Ed.), *Advances in Computers* (Vol. 41, pp. 191–229). Elsevier. Retrieved from http://www.sciencedirect.com/science/article/pii/S006524580860234X

34. Hans van Waayenburg, & Raffi Margaliot. (2016). World Quality Report 2016-2017. Retrieved October 15, 2016, from https://www.sogeti.com/explore/press-releases/world-quality-report-2016-2017/

35. Hass, A. M. J. (2008). *Guide to advanced software testing*. Boston: Artech House.

36. Huizinga, D., & Kolawa, A. (2007). *Automated Defect Prevention: Best Practices in Software Management*. John Wiley & Sons.

37. IBM - Software - IBM Security AppScan. (2016, January 1). Retrieved December 4, 2016, from http://www.ibm.com/software/products/en/appscan, http://www.ibm.com/software/products/en/appscan

38. IEEE Standard Glossary of Software Engineering Terminology. (1990). *IEEE Std 610.12-1990*, 1–84. https://doi.org/10.1109/IEEESTD.1990.101064

39. ISO/IEC/IEEE 29119 Software Testing Standard. (2014). Retrieved December 11, 2016, from http://www.softwaretestingstandard.org/

40. Jorgensen, P. C. (2016). *Software Testing: A Craftsman's Approach, Fourth Edition*. CRC Press.

41. Juran, J. M. (1988). Juran's Quality Control. New York: McGraw-Hill.

42. Juristo, N., Moreno, A., & Strigel, W. (2006). Software Testing Practices in Industry. *IEEE Software*, *23*(4), 19–21.

43. Kardelis, K. (2007). *Mokslinių tyrimų metodologija ir metodai*.

44. Kasurinen, J., Taipale, O., & Smolander, K. (2010). Software Test Automation in Practice: Empirical Observations. *Advances in Software Engineering*, *2010*, e620836. https://doi.org/10.1155/2010/620836

45. Khan, M. E. (2011a). Different Approaches To Black Box Testing Technique For Finding Errors. *International Journal of Software Engineering & Applications*, *2*(4), 31.

46. Khan, M. E. (2011b). Different approaches to white box testing technique for finding errors. *International Journal of Software Engineering & Applications*, *5*(3), 1–14.

47. Khan, M. E., & Khan, F. (2012). A Comparative Study of White Box, Black Box and Grey Box Testing Techniques. *International Journal of Advanced Computer Science & Applications*, *3*(6), 12–15.

48. Kitchenham, B., & Lawrence, P., Shari. (1996). Software quality: The elusive target. *IEEE Software*, *13*(1), 12–21.

49. Kuliešius, T. (2008). Web aplikacijų testavimo veiklos tobulinimas IT organizacijoje (pp. 32–38). Presented at the 11-osios Lietuvos jaunųjų mokslininkų konferencijos „Mokslas – Lietuvos ateitis“, VGTU „Technika“. Retrieved from http://leidykla.vgtu.lt/conferences/jmk_informatika_2008/files/pdf/kuliesius_32-38.pdf

50. Last, M., Friedman, M., & Kandel, A. (2004). Using data mining for automated software testing. *International Journal of Software Engineering and Knowledge Engineering*, *14*(4), 369–393. https://doi.org/10.1142/S0218194004001737

51. Lee, J., Kang, S., & Lee, D. (2012). Survey on software testing practices. *IET Software*, *6*(3), 275–282. https://doi.org/10.1049/iet-sen.2011.0066

52. Li, Z., Gittens, M., Murtaza, S. S., Madhavji, N. H., Miranskyy, A. V., Godwin, D., & Cialini, E. (2009). Analysis of pervasive multiple-component defects in a large software system. In *2009 IEEE International Conference on Software Maintenance* (pp. 265–273). https://doi.org/10.1109/ICSM.2009.5306307

53. Li, Z., Harman, M., & Hierons, R. M. (2007). Search Algorithms for Regression Test Case Prioritization. *IEEE Transactions on Software Engineering*, *33*(4), 225–237. https://doi.org/10.1109/TSE.2007.38

54. Liu, H., & Kuan Tan, H. B. (2009). Covering code behavior on input validation in functional testing. *Information and Software Technology*, *51*(2), 546–553. https://doi.org/10.1016/j.infsof.2008.07.001

55. Mailewa, A., Herath, J., & Herath, S. (2015). A Survey of Effective and Efficient Software Testing. Presented at the The Midwest Instruction and Computing Symposium. Retrieved from http://www.micsymposium.org/mics2015/ProceedingsMICS_2015/Mailewa_2D1_41.pdf

56. McCall, J. A., Richards, P. K., & Walters, G. F. (1977). *Factors in Software Quality. Volume I. Concepts and Definitions of Software Quality*.

57. McGloin, S. (2008). The trustworthiness of case study methodology. *Nurse Researcher*, *16*(1), 45–54.

58. Miller, T. (Ed.). (2012). *Ethics in qualitative research* (2nd ed). Los Angeles, Calif: London : SAGE.

59. Mulder, D. L., & Whyte, G. (2013). A Theoretical Review of the Impact of Test Automation on Test Effectiveness. *Proceedings of the European Conference on Information Management & Evaluation*, 168–179.

60. Myers, G. J. (1979). *The art of software testing*. New York: Wiley.

61. Myers, G. J., Sandler, C., & Badgett, T. (2011). *The Art of Software Testing*. John Wiley & Sons.

62. Naik, K., & Tripathy, P. (2008). *Software testing and quality assurance: theory and practice*. Hoboken, N.J: John Wiley & Sons.

63. Ng, S. P., Murnane, T., Reed, K., Grant, D., & Chen, T. Y. (2004). A preliminary survey on software testing practices in Australia. In *Software Engineering Conference, 2004. Proceedings. 2004 Australian* (pp. 116–125). https://doi.org/10.1109/ASWEC.2004.1290464

64. Nidhra, S., & Dondeti, J. (2012). Blackbox and whitebox testing techniques-a literature review. *International Journal of Embedded Systems and Applications (IJESA)*, *2*(2), 29–50.

65. Org, E. (2012). Improving Regression Testing with Real-world Environments. *Wireless Design & Development*, *20*(5), 30–32.

66. Ostrand, T. J., Weyuker, E. J., & Bell, R. M. (2005). Predicting the location and number of faults in large software systems. *IEEE Transactions on Software Engineering*, *31*(4), 340–355. https://doi.org/10.1109/TSE.2005.49

67. Perry, W. E. (2006). *Effective methods for software testing* (3rd ed). Indianapolis, IN: Wiley.

68. Persson, C., & Yilmazturk, N. (2004). Establishment of automated regression testing at ABB: industrial experience report on “avoiding the pitfalls.” In *Proceedings. 19th International Conference on Automated Software Engineering, 2004.* (pp. 112–121). https://doi.org/10.1109/ASE.2004.1342729

69. Popescu, M. (2010). Integration of the Functional Testing with the General Theory of the Technical Diagnosis. *Informatica Economica*, *14*(4), 57–67.

70. Pressman, R. S. (2000). *Software Engineering – A Practitioner’s Approach,*. London: McGraw-Hill International.

71. Rothermel, G., Untch, R. H., Chu, C., & Harrold, M. J. (1999). Test case prioritization: an empirical study. In *IEEE International Conference on Software Maintenance, 1999. (ICSM ’99) Proceedings* (pp. 179–188). https://doi.org/10.1109/ICSM.1999.792604

72. Saglietti, F., Oster, N., & Pinte, F. (2008). White and grey-box verification and validation approaches for safety- and security-critical software systems. *Information Security Technical Report*, *13*(1), 10–16. https://doi.org/10.1016/j.istr.2008.03.002

73. Sawat, A. A., Bari, P. H., Chawan, & P. M. (2012). Software testing techniques and strategies. *International Journal of Engineering Research and Applications (IJERA)*, *2*(3), 980–986.

74. SonarQube. (2016). Retrieved from http://www.sonarqube.org/

75. Srivastava, P. R. (2008). Test case prioritization. *Journal of Theoretical and Applied Information Technology*, *4*(3), 178–181.

76. The History of Software Testing. (2015).

77. Uspenskiy, S. (2010). *A survey and classification of software testing tools.*

78. Vegas, S., Juristo, N., & Basili, V. (2002). What Information is Relevant when Selecting Software Testing Techniques? *International Journal of Software Engineering & Knowledge Engineering*, *12*(6), 657.

79. Williams, L., Kudrjavets, G., & Nagappan, N. (2009). On the Effectiveness of Unit Test Automation at Microsoft (pp. 81–89). IEEE. https://doi.org/10.1109/ISSRE.2009.32

80. Wong, W. E., Horgan, J. R., London, S., & Agrawal, H. (1997). A study of effective regression testing in practice (pp. 264–274). IEEE Comput. Soc. https://doi.org/10.1109/ISSRE.1997.630875

81. Yin, R. K. (2003). *Case study research: design and methods* (3rd ed). Thousand Oaks, Calif: Sage Publications.

82. Yin, R. K. (2012). *Applications of case study research* (3rd ed). Thousand Oaks, Calif: SAGE.

83. Zitser, M., Lippmann, R., & Leek, T. (2004). Testing Static Analysis Tools Using Exploitable Buffer Overflows from Open Source Code. In *Proceedings of the 12th ACM SIGSOFT Twelfth International Symposium on Foundations of Software Engineering* (pp. 97–106). New York, NY, USA: ACM. https://doi.org/10.1145/1029894.1029911

SUMMARY IN ENGLISH

The significance of software testing has gained more mainstream attention from information technology professionals as demand for computer software increases. Software testing might be costly and demanding in human effort or in technology which multiplies it. However it is often misjudged as a routine and low-level task. Despite these unjustified presumptions, testing a critical part of software development process determining the efficiency or even correctness of final product that is tended to be free of serious defects. Software testing faces a collection of challenges which are strongly related with the organizational contexts. The master thesis is focused on exploration of these contexts and provides insights about software testing nuances in specific enterprise.

The research problem. The effectiveness of software testing techniques. There are various software testing techniques, but the advantages of using one testing technique as opposed to another in a given situation are unclear. Additionally, the external problematic issues limit the effective testing.

The purpose of the research is to investigate the use of software testing techniques in terms of limitations and improvements in software quality assurance process at specific enterprise. **The objectives** are defined in order to achieve the purpose: to explore quality assurance process and identify the relationship between software testing and quality assurance by generalizing scientific literature analysis; to provide a comprehensive view on the main features of software testing techniques by examining theoretical studies and empirical studies of the best practices; to prepare a theoretical framework for conducting a case study for software testing techniques within a specific enterprise; to explore and define the most problematic areas and potential improvements in software testing process by generalizing results of case study and enterprise statistical documents.

Methods of the research: Theoretical methods: comparison and contrast, generalization, abstraction, analogy, modeling, scientific literature review. Empirical methods: case study based on expert interviews and quantitative statistical document analysis.

The research consists of four chapters each of them analyses the objectives provided above.

The main concepts and keywords: Quality Assurance, Software Testing Techniques, Automation Testing.

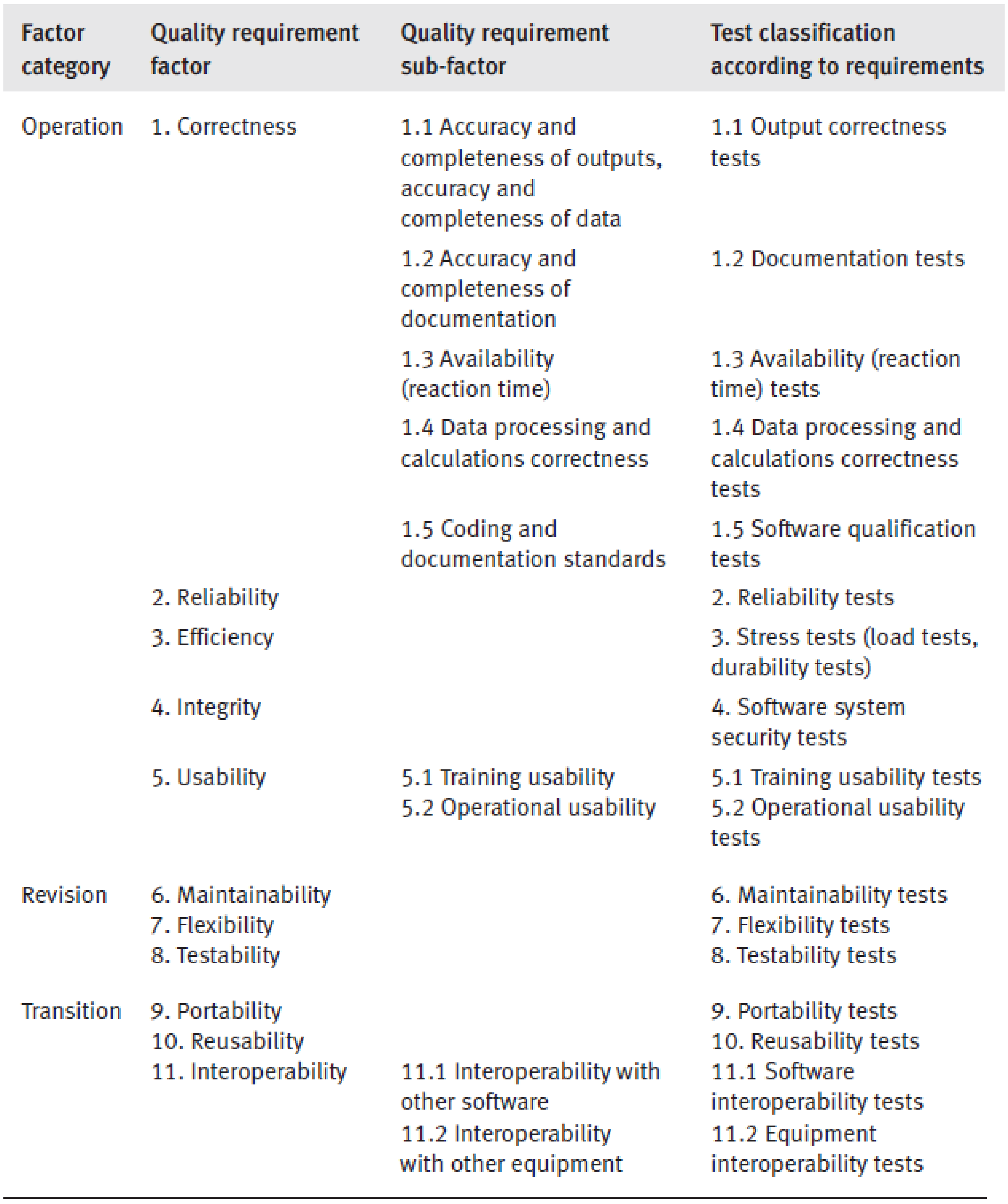
SANTRAUKA

Poreikis programinės įrangos testavimui išaugo sulig technologijų skvarba. Programinės įrangos testavimas neretai vertinamas kaip rutininė ir mažai įgūdžių reikalaujanti veikla, tačiau šis požiūris nėra teisingas atsižvelgiant į tai, jog testavimo procesas yra esminis užtikrinantis galutinio produkto kokybę. Visgi, testavimo technikų naudojimas bei proceso sklandumas dažnai priklauso nuo organizacijos konteksto. Šiame magistro baigiamajame darbe atskleidžiamos įvairių testavimo technikų ypatybės bei jų taikymas konkrečioje įmonėje taikant **atvejo studijos metodologiją**. Darbo **problema** kyla iš fakto, jog esti daug testavimo technikų, tačiau rekomendacijos, kokiais atvejais jas naudoti tinkamiausia, yra neaiškios. Dėl specifinių situacijų, su kuriomis susiduria įmonės vystydamos programinę įrangą, šis neapibrėžtumas gali suprastinti galutinio produkto kokybę bei daryti įtaką proceso efektyvumui. Magistro darbo **tikslas** - ištirti programinės įrangos testavimo technikas, jų galimybes bei ribotumą konkrečios įmonės atveju. Siekiant šį tikslą įgyvendinti keliami keturi **uždaviniai**: 1) atskleisti ryšį tarp testavimo bei galutinio programinės įrangos produkto kokybės užtikrinimo; 2) išanalizuoti skirtingas testavimo technikas bei pateikti geriausius jų taikymo pavyzdžius; 3) sukurti teorinį modelį atvejo analizės tyrimui konkrečioje organizacijoje; 4) atvejo analizės pagalba atskleisti bei išnagrinėti priežastis dėl kurių programinės įrangos testavimo procesas stringa konkrečioje organizacijoje. Tikslui bei uždaviniams pasiekti naudojama lyginamoji analizė, mokslinės literatūros analizė, modeliavimo bei analogijų teoriniai **metodai**. Empirinėje dalyje atliekama mišraus pobūdžio atvejo analizė, su išskirta kokybine kryptimi. **Kiekybiniai duomenys** gauti iš vidinės įmonės dokumentacijos. **Kokybinių duomenų** gavimui buvo atliekamas ekspertinis interviu su įmonėje dirbančiais profesionalais. Darbą sudaro keturi skyriai bei priedai.

Raktiniai žodžiai: programinės įrangos testavimas, kokybės užtikrinimas, programinės įrangos testavimo technikos, automatinis testavimas

ANNEXES

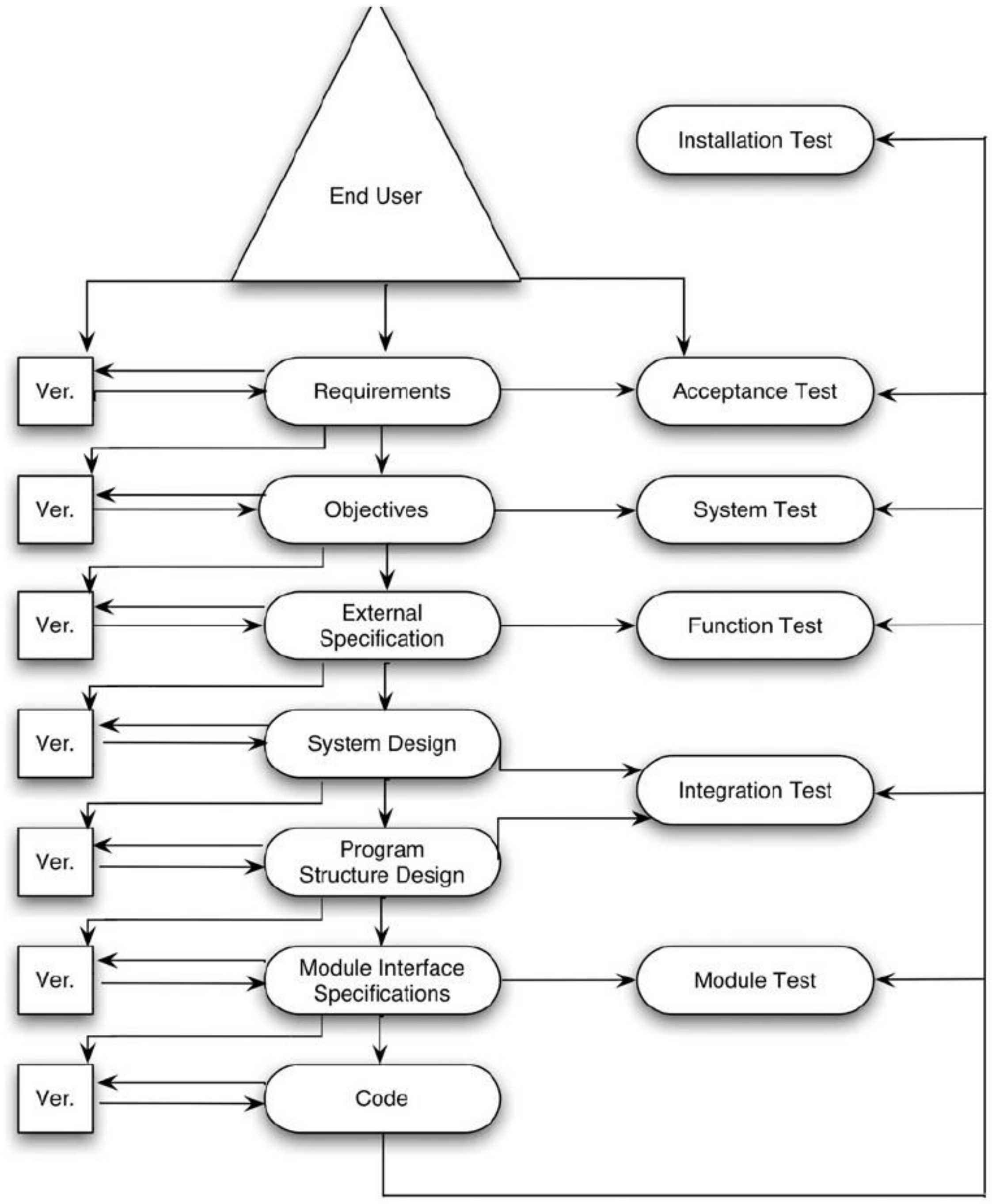
Annex 1. Software quality requirements and test classification



Factor category	Quality requirement factor	Quality requirement sub-factor	Test classification according to requirements
Operation	1. Correctness	1.1 Accuracy and completeness of outputs, accuracy and completeness of data	1.1 Output correctness tests
		1.2 Accuracy and completeness of documentation	1.2 Documentation tests
		1.3 Availability (reaction time)	1.3 Availability (reaction time) tests
		1.4 Data processing and calculations correctness	1.4 Data processing and calculations correctness tests
		1.5 Coding and documentation standards	1.5 Software qualification tests
	2. Reliability		2. Reliability tests
	3. Efficiency		3. Stress tests (load tests, durability tests)
	4. Integrity		4. Software system security tests
	5. Usability	5.1 Training usability 5.2 Operational usability	5.1 Training usability tests 5.2 Operational usability tests
Revision	6. Maintainability 7. Flexibility 8. Testability		6. Maintainability tests 7. Flexibility tests 8. Testability tests
Transition	9. Portability 10. Reusability		9. Portability tests 10. Reusability tests
	11. Interoperability	11.1 Interoperability with other software 11.2 Interoperability with other equipment	11.1 Software interoperability tests 11.2 Equipment interoperability tests

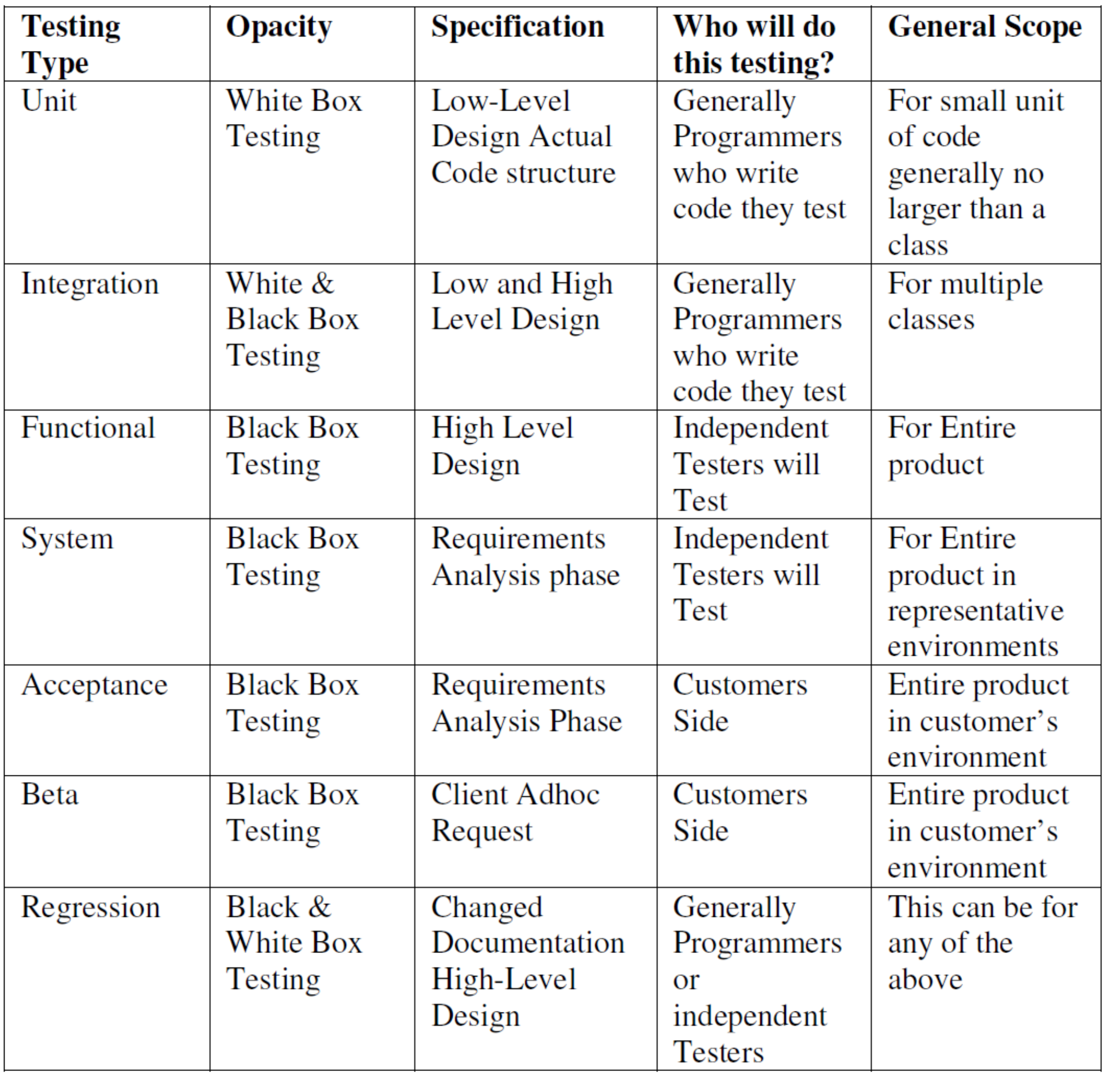
Source: (Galin, 2004)

Annex 2. The correspondence between development and testing processes



Source: (Myers et al., 2011)

Annex 3. The comparison between Software Testing Techniques

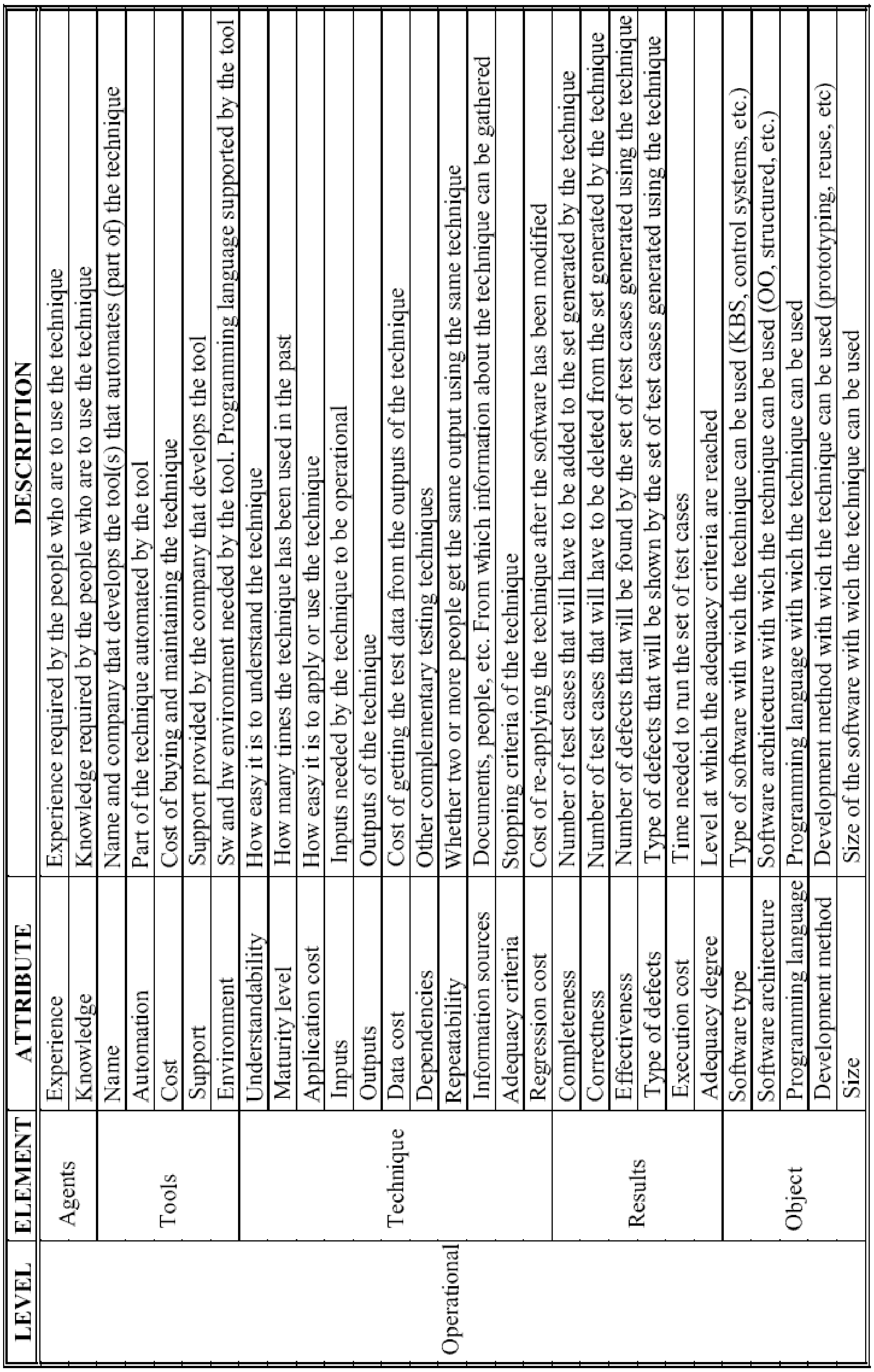


Testing Type	**Opacity**	**Specification**	**Who will do this testing?**	**General Scope**
Unit	White Box Testing	Low-Level Design Actual Code structure	Generally Programmers who write code they test	For small unit of code generally no larger than a class
Integration	White & Black Box Testing	Low and High Level Design	Generally Programmers who write code they test	For multiple classes
Functional	Black Box Testing	High Level Design	Independent Testers will Test	For Entire product
System	Black Box Testing	Requirements Analysis phase	Independent Testers will Test	For Entire product in representative environments
Acceptance	Black Box Testing	Requirements Analysis Phase	Customers Side	Entire product in customer's environment
Beta	Black Box Testing	Client Adhoc Request	Customers Side	Entire product in customer's environment
Regression	Black & White Box Testing	Changed Documentation High-Level Design	Generally Programmers or independent Testers	This can be for any of the above

Source: (Nidhra & Dondeti, 2012)

Annex 4. The factors that influence the selection of Testing Technique

Source: (Vegas et al., 2002)



LEVEL	ELEMENT	ATTRIBUTE	DESCRIPTION
Operational	Agents	Experience	Experience required by the people who are to use the technique
		Knowledge	Knowledge required by the people who are to use the technique
	Tools	Name	Name and company that develops the tool(s) that automates (part of) the technique
		Automation	Part of the technique automated by the tool
		Cost	Cost of buying and maintaining the technique
		Support	Support provided by the company that develops the tool
		Environment	Sw and hw environment needed by the tool. Programming language supported by the tool
	Technique	Understandability	How easy it is to understand the technique
		Maturity level	How many times the technique has been used in the past
		Application cost	How easy it is to apply or use the technique
		Inputs	Inputs needed by the technique to be operational
		Outputs	Outputs of the technique
		Data cost	Cost of getting the test data from the outputs of the technique
		Dependencies	Other complementary testing techniques
		Repeatability	Whether two or more people get the same output using the same technique
		Information sources	Documents, people, etc. From which information about the technique can be gathered
		Adequacy criteria	Stopping criteria of the technique
	Results	Regression cost	Cost of re-applying the technique after the software has been modified
		Completeness	Number of test cases that will have to be added to the set generated by the technique
		Correctness	Number of test cases that will have to be deleted from the set generated by the technique
		Effectiveness	Number of defects that will be found by the set of test cases generated using the technique
		Type of defects	Type of defects that will be shown by the set of test cases generated using the technique
		Execution cost	Time needed to run the set of test cases
		Adequacy degree	Level at which the adequacy criteria are reached
	Object	Software type	Type of software with wich the technique can be used (KBS, control systems, etc.)
		Software architecture	Software architecture with wich the technique can be used (OO, structured, etc.)
		Programming language	Programming language with wich the technique can be used
		Development method	Development method with wich the technique can be used (prototyping, reuse, etc)
		Size	Size of the software with wich the technique can be used

Annex 5. The questionnaire for Experts

You are invited to participate in our survey "Effective Quality Assurance in your Enterprise". It will take approximately 15 minutes to complete the questionnaire. As an expert in your field, your comments in response to a few questions would be greatly appreciated. Your survey responses will be strictly confidential and data from this research will be reported only in the aggregate. Your information will be coded and will remain confidential. Thank you very much for your time and support.

How many years are you working as a software test specialist?

How many years are you working for this company?

What team are your working for?

What types of testing techniques does your team use?

1. Installation testing
2. Compatibility testing
3. Regression testing
4. Acceptance testing
5. Alpha testing
6. Beta testing
7. Functional testing
8. Non-functional testing
9. Continuous testing
10. Destructive testing
11. Software performance testing
12. Usability testing
13. Accessibility testing
14. Security testing
15. Internationalization and localization
16. Development testing
17. A/B testing
18. Concurrent testing
19. Conformance testing or type testing
20. Other __________

What kinds of Static analysis are used in your team (mostly by developers)?

1. Code or Design Inspection
2. Review (informal)
3. Review (peer)

4. Review (technical)
5. Review (management)
6. Walk-through

Prioritize the following issues of software testing that should be used more effective in your team. Start from the highest point that should be improved.

- Incident handling __________
- Test design __________
- Test efforts __________
- Exit criteria for testing __________
- Test techniques __________
- Test tools __________
- Time __________
- Test Management (test goals, process) __________
- Test Planning __________
- Test execution __________

What are the most critical problems in some or all different areas? Why? Give some examples below after selection.

1. Requirements
2. Unit level
3. Integration level
4. System level
5. Functional test
6. Acceptance test
7. Regression test
8. Management
9. Organizational issues
10. Test environment
11. Other __________

What testing processes or test types should be automated? Why? Give some examples.

In your opinion what topics related with Software Testing should be included in internal or external training/courses in deeper detail?