TALLINN UNIVERSITY OF TECHNOLOGY
Faculty of Information Technology

IDX70LT

Margarita Aravina 100257IAPMM

DEVELOPING METHODS FOR ANALYSIS AND EVALUATION OF REGRESSION TESTING PROCESS

Master's thesis

Supervisor: Jaak Tepandi
Professor

Tallinn 2016

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

IDX70LT

Margarita Aravina 100257IAPMM

REGRESSIOONITESTIMISE PROTSESSI ANALÜÜSI JA HINDAMISE MEETODITE ARENDAMINE

Magistritöö

Juhendaja: Jaak Tepandi
Professor

Tallinn 2016

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Margarita Aravina

26.04.2016

Abstract

Regression testing is an integral part of the software quality assurance process, essential for gaining confidence in the quality of modified software. Regression testing is an extremely expensive activity – every new version of a software has to be regression tested, and the amount of regression testing effort increases as the software evolves. Thus, effective regression testing strategies and approaches are crucial for the effective quality assurance system. Many studies on optimization of regression testing exist, most of them are focused on the analysis and evaluation of various optimization methods. The current work focuses on the analysis and evaluation of the overall regression testing process. The main goal is to develop systematic methods for the analysis and evaluation of the overall regression testing process with the aim of its further improvement.

As a result of the given work two reference and guideline models have been developed for the analysis, evaluation and further improvement of the maturity, efficiency and effectiveness of regression testing process. The first model is intended for the analysis, evaluation and improvement of the level of process maturity, which is based on the evaluation of various process characteristics, like strategies and approaches employed. The second model represents a set of core metrics for the analysis and evaluation of the efficiency and effectiveness of regression testing.

The maturity model can also be used as reference model for establishing a mature and effective regression testing practice.

The models were evaluated by applying them to a real regression testing practice. As a result, key areas for improvement in the regression testing process have been detected and corresponding proposals for improvements have been made.

This thesis is written in English and is 68 pages long, including 6 chapters, 5 figures and 3 tables.

Annotatsioon

Regressioonitestimise protsessi analüüss ja hindamise meetodite arendamine

Regressioonitestimine on tarkvara kvaliteedi tagamise protsessi lahutamatu osa, mis on oluline modifitseeritud tarkvara kvaliteedi usaldatavuse saavutamiseks. Regressioonitestimine on äärmiselt kulukas tegevus – iga uut tarkvaraversiooni tuleb regressioonitestida ning tarkvara arenedes suureneb ka regressioonitestimise maht. Seega on efektiivsed regressioonitestimise strateegiad ja lähenemisviisid otsustava tähtsusega efektiivse kvaliteeditagamise süsteemi jaoks. Regressioonitestimise optimeerimise kohta on olemas mitmeid uuringuid ning enamus neist keskenduvad erinevate optimeerimismeetodite analüüsile ja hindamisele. Käesoleva töö keskmes on üldise regressioonitestimise protsessi analüüs ja hindamine. Põhiülesandeks on luua süstemaatilised meetodid üldise regressioonitestimise protsessi analüüsiks ja hindamiseks nende edasise täiustamise eesmärgil.

Käesoleva töö tulemusena koostati kaks etalon- ja juhtmudelit regressioonitestimise protsessi valmiduse, tõhususe ja efektiivsuse analüüsiks, hindamiseks ja täiustamiseks. Esimene mudel on mõeldud protsessi valmiduse analüüsiks, hindamiseks ja arendamiseks, mis põhineb erinevate protsessi karakteristikute hindamisel, nagu kasutusel olevad strateegiad ja käsitlused. Teine mudel kujutab endast põhiparameetrite kogumit regressioonitestimise tõhususe ja efektiivsuse analüüsiks ja hindamiseks.

Vastavaid mudeleid saab kasutada ka etalonmudelitena valmisoleva ja efektiivse regressioonitestimise praktikate loomisel.

Mudeleid rakendati reaalse regressioonitestimise praktika analüüsiks ja hindamiseks. Selle tulemusena on välja selgitatud täiustamist vajavad põhivaldkonnad ning esitatud vastavad parandusettepanekud.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 68 leheküljel, 6 peatükki, 5 joonist, 3 tabelit.

List of abbreviations and terms

RT	Regression Testing
CMMI	Capability Maturity Model Integration
TMMi	Test Maturity Model Integration
PA	Process Area
SG	Specific Goal
SP	Specific Practice
RTMM	Regression Testing Maturity Model
CI	Continuous Integration. The practice, in software engineering, of merging all developer working copies to a shared mainline several times a day [50].
Error Guessing	A test method in which test cases used to find bugs in programs are established based on experience in prior testing. The scope of test cases usually relies on the software tester involved, who uses past experience and intuition to determine what situations commonly cause software failure, or may cause errors to appear [27].
Exploratory Testing	An approach to software testing that is concisely described as simultaneous learning, test design and test execution [28].
GUI	Graphical User Interface
SUT	System Under Test

Table of contents

List of figures

List of tables

1 Introduction

1.1 Background

Regression testing, or testing of modified software with the aim to ensure that no regression defects were introduced in result of changes, is an integral part of the software quality assurance process. It is essential for gaining confidence in the quality of modified software, what is especially important when software is developed iteratively and incrementally, continually evolved and changed frequently. Every change, even a small one, is a potential source of the software regression – unexpected defects in the previously working software. Regression testing is intended to mitigate this risk: enable early detection of regression defects and ensure continuous quality assurance.

As much regression testing is important, so much it is expensive. Regression testing is a time and effort consuming activity – every time a new version of software is released it should be regression tested, also the amount of regression testing increases as software evolves. The frequency with which regression testing should be performed and the continuous increase of the amount of required effort lead to the fact that regression testing consumes much more resources than any other testing-related activity in a project. Studies show that up to 80% of the overall testing effort falls on the regression testing [24].

The importance of regression testing and its high cost imply that effective strategies for regression testing are crucial for the effectiveness of the test process, quality assurance, and generally for the success of a software development project.

There are many studies on how to improve the efficiency and effectiveness of regression testing. Most of them are focused on the optimization of the process, specifically on the regression testing optimization techniques, such as minimization, selection and prioritization, and the effectiveness of these techniques in different contexts. The given work focuses on the regression testing process in general. The primary goal of this

thesis is to develop systematic methods for the analysis and evaluation of regression testing process aimed at improving it. No relevant studies were identified.

In order to be able to analyze and evaluate the process of regression testing from different perspectives and aspects, two methods of analysis are proposed. One of the proposed methods can be referred as static analysis. Static analysis of regression testing process is performed based on the evaluation of relatively static characteristics of the process, like strategies and approaches employed, and aimed at assessing the level of process maturity. The introduced model for static analysis of regression testing is intended for the analysis, evaluation and improvement of the level of process maturity, which depends on strategies and approaches employed, and the effectiveness of these approaches in the context of regression testing. The other method can be referred as dynamic analysis. Dynamic analysis of regression testing process is performed based on process performance indicator data obtained during the execution of the process, and aimed at assessing the efficiency and effectiveness of the process. The introduced model for dynamic analysis of regression testing represents a set of core metrics for measuring the efficiency and effectiveness of the process.

The proposed models are intended to serve as reference and guideline models for the regression testing process establishment, analysis, evaluation and further improvement. The models can be considered as complementary and can be used in combination in order to get better insight into a process of regression testing – analyze and evaluate the existing process from different perspectives, considering its various aspects. The maturity model can also be used as reference models for establishing mature and effective regression testing practices.

1.2 Goals and Objectives

The goals and objectives of this thesis are the following:

- Develop methods and models for the analysis and evaluation of regression testing process:
    - Develop a framework for analysis and evaluation of maturity of regression testing process.

    - Develop a metrics model for analysis and evaluation of the efficiency and effectiveness of regression testing process.

- Apply the proposed models to the real world context:
    - Analyze and evaluate a real regression testing practice based on the proposed methods.
    - Examine and evaluate the proposed models in terms of their applicability to the real world context.

1.3 Scope

The current work is focused on the analysis and evaluation of maturity, efficiency and effectiveness of the overall regression testing process. The analysis and evaluation of the effectiveness of various test strategies and approaches (e.g. manual vs. automated testing), regression testing optimization techniques (minimization, selection and prioritization) and methods are out of scope.

1.4 Outline of the Thesis

The given work is organized as follows:

- Chapter 2 contains introduction to regression testing:
    - The definition of regression testing;
    - Systematic review on regression testing;
    - Review of researches on regression testing;
    - Review of regression testing practices;
- In Chapter 3 the Regression Testing Maturity Model (RTMM) is introduced. This chapter includes:
    - Research on Test Maturity Model (TMMi);
    - Development of maturity model for regression testing process;
- In Chapter 4 the Regression Testing Metrics Model is introduced. This chapter includes:
    - Research on software measurement and analysis;
    - Research on software measurement process standards and frameworks;

    - Research on software metrics and software testing metrics;
    - Development of core regression testing metrics;
- Chapter 5 contains analysis and evaluation of industrial regression testing practice based on the proposed methods:
    - Analysis and evaluation of industrial regression testing practice based on RTMM.

2 Regression Testing

2.1 Software Regression

Software regression is a software quality deterioration which may happen in result of introducing any kind of changes to the previously stable parts of a software system. Software regression can be:

- Functional – meaning faults in the tested and previously operating functions of a software system;
- Non-functional – meaning deterioration of non-functional characteristics of a software system, for example decrease in software performance, increase of software vulnerability to security threats, etc.

The changes that may affect the quality of a software system include:

- Change of the executable code of a software (for example, bug fixes, system enhancements, system optimization);
- Change of the underlying components (third-party components or operating system components);
- Change of the configuration of a software system (for example, hardware / network changes);
- Change in any user data. [23]

In order to ensure that no regressions (functional and non-functional regression defects) have been introduced to a software system in result of changes, or reveal regression defects, regression testing is performed.

2.2 Regression Testing Definition, Goals and Purposes

ISTQB Glossary gives the following generalized definition for regression testing:

“Regression Testing is a testing of a previously tested program following modification to ensure that defects have not been introduced or uncovered in unchanged areas of the software as a result of the changes made. It is performed when the software or its environment is changed.” [16]

In a broader sense regression testing can be defined as follows:

- Regression testing is a distinct type of software testing, which essentially contrasts from the primary testing process (test planning and verification of new (untested) functional and non-functional features of a software product) due to the differences in purposes, goals and methods.

- Regression testing is a repetitive testing of the previously tested software components, functional and non-functional features of a software product with the aim to ensure that no regressions (functional defects, decline in performance, etc.) were introduced in result of changes in a software or its environment.

- The triggers of regression testing are any changes or modifications introduced to a software product or its environment (bug fixes, system enhancements, system optimization, system configuration changes, system components changes, network / hardware changes, etc.).

- Regression testing can be performed on any test level (component, integration, system), with any types of tests (unit, functional, performance, etc.), and at different stages of the development process, depending on triggers, goals and purposes.

- As software development is a highly continuous process, the results of regression testing serve as an important indicator of software quality.

2.3 Regression Testing in Software Development Process

2.3.1 Sequential Software Development Methodologies

The life cycle model of sequential software development methodologies, like Waterfall, consists of distinct, sequential phases, where each subsequent phase starts when its preceding phase has been successfully completed. Waterfall phases in the order of their

execution are the following: System and Software Requirements, Analysis, Design, Coding, Testing, and Operations and Maintenance [17].

In Waterfall development regression testing usually takes place during Maintenance phase which consists in fixing defects found during operational use of a software system, optimization, adding enhancements to product functionality or deleting obsolete capabilities to meet evolving customer requirements. Each new release of the product has to be regression tested to ensure that previously working functionality has not been broken by introduced changes.

2.3.2 Iterative-incremental Software Development Methodologies

The life cycle model of iterative-incremental software development methodologies, like Agile (XP, Scrum), consists of multiple iterations, where each iteration involves different software development activities, such as planning, designing, coding, and testing, and results in a working product.

The main idea of iterative-incremental development, and Agile methodology in particular, is to “build properly tested product increments within short iterations” [18]. Therefore, testing in general and effective regression testing in particular have an important role in Agile development. In agile context the main purposes of regression testing are to ensure continuity of business functions, and to give a rapid feedback on quality of a software system to allow development team to respond efficiently on encountered issues [19].

In Agile development regression testing usually takes place either at the end of each iteration or as a part of continuous integration process after each build in the form of automated tests [20, 21].

2.4 Regression Testing as part of overall Test Plan

Regression testing, as part of overall testing process on the one hand, and as quality assurance discipline with its own specific goals and methods on the other hand, requires planning and certain degree of formalization. It is considered a good practice to plan regression testing activity as a part of the overall testing strategy, and outline regression testing goals and methods in the outset of the project. When planning regression testing

activity, the following points need to be clearly defined:

- The goals of regression testing, and / or the quality goals expected to be accomplished by employing regression testing;
- The risks addressed by regression testing;
- Regression testing strategy, for example:
    - Systematic or exploratory testing, or combination of both;
    - Manual or automated testing, or combination of both;
    - Automation strategy.
- Regression testing approach, for example:
    - At what level(s) regression testing is performed (e.g. component level, system level, business level) [49];
    - Types of testing to be performed for regression testing (e.g. functional testing, non-functional testing);
    - Types of test cases to be used for regression testing (e.g. GUI test cases, functional test cases, system test cases, positive test cases, negative test cases, boundary / corner test cases, etc.);
    - 'Retest all' or selective testing;
    - The source of regression test cases (e.g. re-use existing test cases, create dedicated regression test cases);
    - Regression testing optimization techniques to be applied (minimization, selection, prioritization);
    - The methods for test suite minimization, test case selection and prioritization to be applied (ad hoc, error guessing, systematic approaches);
    - The scope and coverage of regression testing:

        - Items and features to be regression tested;
        - Criteria for the selection of test cases (modification-based, coverage-based, risk analysis-based);
        - Expected regression test coverage.

- Regression testing application policy:
    - Regression testing triggers (e.g. system enhancements, bug fixes, system configuration changes, etc.);
    - Regression testing execution schedule:
        - Periodic execution (e.g. daily, weekly, monthly, etc.);
        - Rule-based execution (e.g. after any change, after changing critical parts only, after certain amount of changes, etc.).

2.5 Regression Testing Approaches and Methodologies

Two main questions arise when planning regression testing: how to test, and what to test.

The first question comes down to the selection of a testing approach to be applied, and there are at least two possible options: applying a systematic approach (or so-called scripted testing) or performing heuristics and experience based exploratory testing*.

The second question consists in determining a scope of regression testing, and may include:

- Determination of items and features to be covered with regression tests. The selection can be done based on different kinds of risks, for example, based on change risks – the risks concerned with introduced changes (selecting items, functional areas or features that are more affected or may be potentially affected by changes), business risks (selecting items, functional areas or features that are more critical or prioritized from business point of view). Also selection may be dictated by the requirements to test coverage specified in the

regression test strategy (e.g., to cover to some degree all items, functional areas or features);

- Determination of test scenarios and test cases to be executed. The main requirement in the selection of test cases is that they have to ensure adequate regression testing:
    - On one side, ensure proper testing – address potential risks and the most critical functionalities; ensure proper coverage, both in terms of functional coverage and in terms of how these functions are examined (with what kind of test cases: GUI test cases, functional test cases, system test cases, positive test cases, negative test cases, boundary / corner test cases, etc.);
    - On the other side, ensure optimal testing – avoid duplication of test effort and thus be cost effective in terms of time and resources required for execution.

In case of systematic regression testing approach (scripted testing), it is expected, that test scenarios have been defined during previous testing sessions and prescribed test cases and procedures exist. Thereby the determination of a scope of regression testing mostly comes down to the question of how to select proper test cases among the existing ones. The most well-known approaches to regression testing and techniques for selection of effective regression tests – widely studied, described in various literatures and used in practice – are the following:

- **'Retest all' approach.** Means execution of all existing test cases for regression testing. It may be considered as the most reliable approach, but in most cases practically impossible due to its high cost, time and resource restrictions.
- **'Selective Testing' approach.** Means execution of a subset of existing test cases for regression testing. Selective regression testing implies application of certain regression testing optimization techniques. These techniques are minimization, selection and prioritization. Regression test suites are compiled and optimized using various minimization, selection and prioritization methods:

- In an ad hoc way, without application of any systematic approaches.
- With **error guessing**, or experience and heuristics based approach, where test cases are selected based on testers knowledge and experience with a system under test:
    - Based on domain knowledge and experience: what are the business processes, what are the most critical functions, how the system is used in practice, what are the needs and goals of real users, etc.
    - Based on technical knowledge and experience: how the system is built, what technologies are used, how the system communicates with third-party applications, etc.
    - Based on testing experience and intuition: what are the typical problems and defects, how the changes may affect the system, etc.
- Based on strict rules or guidelines for the selection of tests cases for regression testing. These rules or guidelines are normally defined as a part of the regression testing strategy, and may indicate the risks to be addressed when selecting test cases, the requirements for test coverage that has to be fulfilled, certain items, functional areas or features that has to be covered with regression tests, types of tests to be selected, etc.
- By application of systematic methods and algorithms for test suite minimization, test case selection and prioritization.

Systematic regression testing approach can be implemented both through manual execution of pre-designed and pre-defined test cases or by automating them.

Exploratory testing does not imply the existence of pre-designed test cases, and test cases are not defined in advance. Testers identify potential risks, risky areas and test cases addressing these risks based on error guessing* - prior experience and intuition. Exploratory testing is done manually.

Regression testing approaches and methodologies are depicted in the picture below.

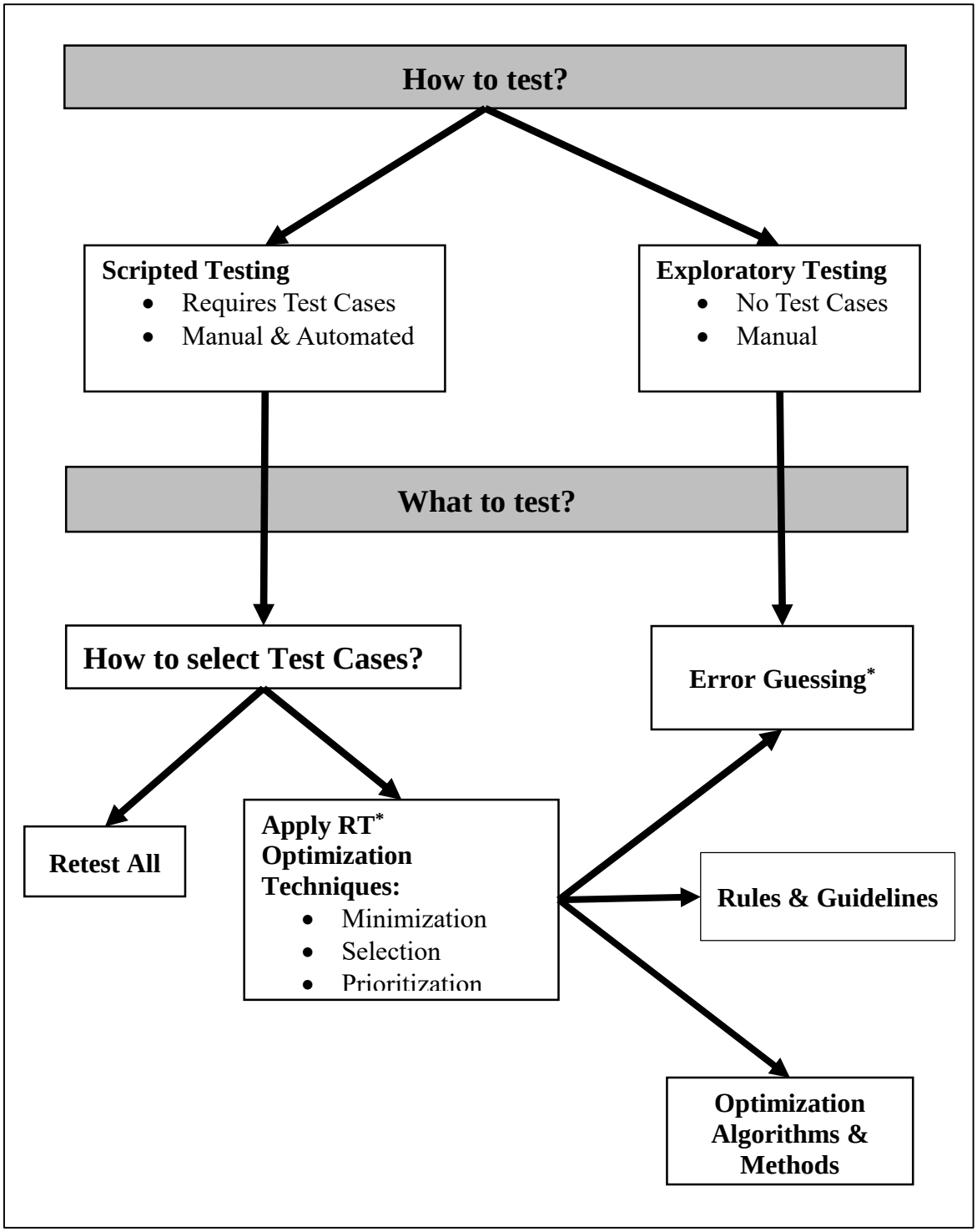

Figure 1. Regression Testing Approaches and Methodologies

2.5.1 Automated vs. Manual

Two general approaches to software testing exist – manual testing and automated

testing. Manual testing is carried out manually by human, and automated testing is executed automatically with the assistance of software test automation tools.

There are many advantages and disadvantages of both automated testing and manual testing and the benefits of one over another, and also many opinions and debates in the software development community on which approach (fully manual, fully automated or some combination) is more reliable and cost-effective, in what cases, and what are the criteria to define that. Among these disputes, the question of automation of regression tests is the most topical, because of the repetitive and tedious nature of regression testing. There is broad agreement that regression tests are the number-one candidates for automation, because:

- Regression tests are expected to be executed repeatedly over a long period of time;
- Regression testing is expected to be as fast as possible (especially in case of agile development, aggressive delivery schedules and time limits), but still reliable, both in terms of accuracy and the amount of test coverage;

Automated regression tests have numerous apparent advantages in comparison with manual test execution:

- Automated tests can be executed repeatedly for an extended period of time;
- Automated tests are significantly faster than manual execution;
- Automated tests can be run at any time, even overnight;
- Automated tests can be run simultaneously on different machines, with different operating systems, against different configurations and test data, etc.;
- Automated regression testing can ensure higher test coverage due to greater amount of test cases that can be executed in certain time frames compared to manual execution;
- Few human resources are required to execute automated regression tests.

Also there are some controversial advantages that in some cases can be considered as

drawbacks:

- Automated tests are considered to be more accurate and reliable, because they precisely perform the same operations, and never miss defects that can be overlooked by a human. The problem is that automated tests never find defects that are out of the strictly defined path of execution, while testers by means of exploratory testing and error guessing are able to discover these issues.

The disadvantages of automation are the following:

- Test automation skill-set, including programming skills, is required from testers;
- Time investment is required to establish test automation. This may include time for developing test automation strategy, developing test automation tools and frameworks (if internal automation tools are expected to be used), developing automated tests, etc.
- Financial investments are required. This may include the costs of automaton tools (if third party automation tools are expected to be used), trainings, automation environments (hardware), etc.
- Automated tests maintenance can be complicated and time consuming.

When deciding what approach to apply, it is essential to weight all costs and benefits. At least the following aspects need to be considered [26]:

- The ability of project team to carry out automated testing;
- The complexity of a system under test and, as a consequence, the complexity of a test automation solution;
- Development cycle and the frequency of test runs;
- The profitability of test automation (cost-benefit analysis).

2.5.2 Regression Testing Optimization Techniques

As it was mentioned in previous sections, there are three main techniques for regression testing optimization exist. These techniques are – minimization, selection and

prioritization.

- Test suite **minimization** (in some literature referred as reduction) is a process that seeks to identify and eliminate obsolete or redundant test cases from a test suite [30]. The goal of test suite minimization is to create a representative regression test suite containing minimum number of test cases that ensure proper functional coverage and effective in finding regression defects.

- Test case **selection** deals with the problem of selecting a subset of test cases that will be used to test changed parts of a software [30].

- Test case **prioritization** concerns the identification of the 'ideal' ordering of test cases that maximizes desirable properties, such as early fault detection [30].

As it was also already mentioned, test suite minimization, test case selection and prioritization can be conducted in different ways:

- In an ad hoc way, without application of any systematic approaches;

- Based on error guessing (tester's knowledge, prior experience and intuition);

- Based on rules or guidelines specified in the test strategy;

- By application of systematic methods and algorithms.

2.5.3 Selection, Minimization and Optimization Methods

There are many various methods and algorithms for test suite minimization, test case selection and prioritization have been proposed by researchers and practitioners so far. In order to determine which of the proposed methods is most appropriate for a certain situation several factors need to be considered [40]. One of the factors is the input – an artifact of software development, which is required for a particular technique, and which serves as a basis for test case selection. Most of the proposed techniques are code based (source, intermediate or binary code) [e.g. 29, 30, 43], but there are also exist various model-based techniques [e.g. 46], specification-based techniques [e.g. 32, 33, 44, 45], and techniques based on different project data [e.g. 41, 42, 46], for example failure reports and tests execution history.

Also an important factor that needs to be considered is empirical evidence. Evaluative case studies and comparative experiments are conducted in order to evaluate different regression testing techniques and optimization methods and indicate most effective ones in specific contexts [e.g. 39]. In order to have an overview of regression testing techniques reported in the literature and empirically evaluated, several surveys have been undertaken [29, 31].

2.6 Researches on Regression Testing

David Parsons, Teo Susnjak and Manfred Lange, in their work 'Influences on Regression Testing Strategies in Agile Software Development Environments' [20], widely analyzed literature on regression testing and identified a number of common research themes. They categorized them into three sets of concerns:

- Operational concerns of regression testing. These researches relate to techniques for optimization of test suites: selection, minimization and prioritization. Much prior research has been done in these areas, including extensive reviews of the literature such as E. Engström, P. Runeson and M. Skoglund (2010) [29], who analyze previous work on test selection, and S. Yoo, M. Harman who additionally look at prioritization and minimization.

- Organizational concerns of regression testing. These researches relate more to the day-to-day management of regression testing within an organization. This covers areas such as test plans, testing standards and metrics. This also has been well explored in the literature.

- Regression testing strategy. This relates to the impact of context on how an organization adopts, organizes and evolves its regression testing policies and implementation. The context can be external (e.g. market) and internal (e.g. the level of maturity of an organization).

All three levels of concerns, indicative literature (as per D. Parsons, T. Susnjak and M. Lange) and some literature that influenced the current work are presented in the table below:

Table 1. Researches on Regression Testing

Level 1: Operational Regression Testing	
RT Optimization Techniques	E. Engström, P. Runeson and M. Skoglund (2010). A systematic review on regression test selection techniques [29] R. P. Gorthi, A. Pasala, K. K. Chanduka, B. Leong (2008). Specification-based Approach to Select Regression Test Suite to Validate Changed Software [32] S. Yoo, M. Harman (2012). Regression Testing Minimization, Selection and Prioritization: A Survey [30] M. Salehie, S. Li, L. Tahvildari, R. Dara, S. Li, M. Moore (2011). Prioritizing Requirements-Based Regression Test Cases: A Goal-Driven Practice [33] A. S. A. Ansari, K. K. Devadkar, P. Gharpure (2013). Optimization of test suite-test case in regression test [34] R. R. Miranda, O. S. Gomez, G. D. Rodriguez (2015). 15 Years of Software Regression Testing Techniques: A Survey [31]
Level 2: Organizational Regression testing	
RT Practices	E. Engström, P. Runeson (2010). A Qualitative Survey of Regression Testing Practices [24] M. Puleio (2006). How Not to do Agile Testing [35]
Metrics	M. Gittens, H. Lutfiyya, M. Bauer, D. Godwin, Y. W. Kim, P. Gupta (2002). An empirical evaluation of

	system and regression testing [36]
Level 3: Regression Testing Strategy	
External Context	H. Svensson and M. Host (2005). Introducing an Agile Process in a Software Maintenance and Evolution Organization [37]
Internal Context	L. Damm, L. Lundberg, D. Olsson (2005). Introducing Test Automation and Test-Driven Development: An Experience Report [38]

2.7 Regression Testing Practices

In order to better understand the variety of regression testing practices in industry, identify their strengths and weaknesses, a qualitative survey [24] has been conducted by Emelie Engström and Per Runeson from Lund University, Sweden.

A total of 46 software engineers from 38 different organizations participated in the focus group and questionnaire survey. The questionnaire consisted of questions on what regression testing is, how regression testing is performed (what regression testing techniques are used, manual vs. automatic testing), and how satisfied the respondents are with the regression testing practices taken in their organizations. The results of the survey were analyzed and represented based on Zachman framework [24] and focused on three categories: what, how and when.

What

There is a common understanding of what the regression testing is and what its main purpose is. The majority of the survey respondents defined regression testing as a repetitive test execution with the aim to check if previously working software has not been broken by changes.

The goal of the regression testing varies between different organizations – it can be find defects, obtain a measure of the quality of a software, or obtain a guide for further

priorities in a project (for example, focus on bug fixing instead of further development of new features). In general, most of the respondents agreed that regression testing should ensure that nothing has been affected or destroyed in result of changes.

The kind of changes to a software system that may affect its quality and serve as a basis for starting regression testing, mentioned in the focus group discussion and confirmed by the majority of the respondents, are the following: new versions, new configurations, fixes, changed solutions, new hardware, new platforms, new designs and new interfaces. One third of the respondents indicated that regression testing is applied regardless of changes. The amount and frequency of regression testing is determined by the assessed risk, the amount of new functionality, the amount of fixes and the amount of available resources.

When

Regression testing is carried out at different levels (module level, component level, system level) and on different stages of the development process. Some organizations perform regression testing as early as possible in order to enable early detection of defects, some postpone regression testing as late as possible in the process for certification or type approval purposes, some carry out regression testing continuously throughout the whole development process.

Regarding how often the regression testing is performed, the most common approach is to regression test before release (indicated by 95% of the respondents), and only 10% of the respondents do regression testing daily.

How

It was identified that tests used for regression testing may be a selection of developer's tests, a selection of tester's tests, a selection of tests from a specific regression test suite, or new test cases are designed. The most common is to reuse test cases designed by testers. Strategies for regression test selection mentioned in the focus group were: complete retest, combine static and dynamic selection, complete retest of safety critical parts, select test cases concentrating on changes and possible side effects, ad-hock selection, smoke test, prioritize and run as many as possible, and focus on functional test cases. It is common to run a set of specified regression test cases every time,

together with a set of situation dependent test cases.

Both manual and automatic regression testing are applied. 50% of the respondents indicate an equal amount of manual and automatic regression testing, while 30% perform regression testing exclusively manually.

Regression testing problem areas

As a result of focus group discussions a lot of problem areas related to regression testing have been identified. The most specific of them are presented below:

- ***Test case selection***. As was mentioned by the respondents, it is hard to assess the impact of changes on existing code and to make a good selection of test cases, prioritize test cases with respect to product risks and fault detection ability, be confident in not missing safety critical faults, determine the required amount of tests, and assess the test coverage. Participants emphasized the importance and need of regression testing guidelines at different stages of a project with respect to quality aspects.

- ***Automated vs. manual regression testing.*** Manual regression testing is less problematic, but time and resource consuming. Automating regression testing causes problems, such as assessment of cost/benefit of test automation, building and maintaining environments for automated testing, implementing and maintaining automated tests, analysis and presentation of test results, but can be more efficient in terms of time and resource usage. Testing automation is particularly important for regression testing, as the same tests are repeated many times. Nonetheless, most participants pointed on the importance of a good balance between automated and manual testing. Use of manual testing was recommended for testing of user experience and for exploratory testing.

- ***Regression test suite and test case maintenance*** were recognized to be a problem. Much of the regression testing becomes redundant with respect to test coverage because of poor methods, tools and processes for maintenance of tests in case of changes in a product, traceability between test cases and requirements and minimization of redundant tests.

Survey conclusions

Besides answering the research questions posed: what is meant by regression testing in industry, which problems or challenges related to regression testing exist and which good practices on regression testing exist; the researchers have come to the conclusion that many of the challenges and practices, highlighted in the study, are not specific to regression testing but common to testing in general. However, they have a significant impact on how effective the regression testing becomes, and this indicates, that regression testing should not be addressed nor researched in isolation; rather it should be an important aspect of the software testing practice and research to take into account.

3 Regression Testing Maturity Model (RTMM)

Further, a model is proposed for static analysis of regression testing process.

Static analysis of regression testing process is proposed to be performed based on relatively static characteristics of the process, like strategies and approaches employed. The degree of process quality is proposed to be determined by the level of process maturity, whereas the level of process maturity depends on strategies and approaches employed, and the effectiveness of these approaches in the context of regression testing. In the development of the proposed model, the effectiveness of particular strategies and approaches was considered as absolute truth, proven by experience (like, automated regression testing is more cost effective than manual regression testing), without consideration of the particular context. While during the evaluation of concrete regression testing practices, the context always should be taken into account, as the effectiveness of the approaches depends on various parameters and may vary in different contexts.

The proposed model for static analysis of regression testing process is Regression Testing Maturity Model, or RTMM. RTMM is based on the TMMi framework and can be considered as elaboration of TMMi with primary focus on regression testing process.

The TMMi focuses on the entire test process and does not address regression testing in sufficient detail. The main idea of RTMM is to provide a detailed reference and guideline framework for regression testing process establishment, assessment and improvement.

The main purposes of RTMM are the following:

- Support in analyzing, evaluating and improving current regression testing practices:
    - RTMM is intended to serve as a reference model for analysis and evaluation of the current regression testing practice within an organization or a project. RTMM is expected to help in detecting

problematic areas and areas for improvement.

    - RTMM is intended to serve as a guideline for improvement of the current regression testing practice within an organization or a project. RTMM is expected to help in identification of improvement opportunities.

- Support in developing and establishing a new process for regression testing.

The main goal of RTMM is to establish an efficient and effective regression testing process.

3.1 Introduction to Test Maturity Model (TMMi)

Test Maturity Model integration, or TMMi framework [22], is a guideline and reference framework for test process assessment and improvement. TMMi is developed by TMMi Foundation [47], a non-profit organization dedicated to improving test practices, and positioned as a complementary model to the software process improvement model CMMI [48].

The main purpose of TMMi model is to support in establishing of a more effective and efficient test process:

- TMMi provides a reference model to be used during the assessment of the testing process – understanding the current position relative to the selected model or standard, and identification of areas for improvement and improvement opportunities.

- TMMi provides a full framework to be used as a reference model during test process improvement.

3.1.1 TMMi Maturity Levels and Process Areas

As CMMI, TMMi uses the concepts of maturity levels. In the TMMi there are five levels of maturity which represents an evolutionary path to test process improvement. Each level has a set of process areas, goals and practices that need to be implemented by an organization in order to achieve maturity at that level. TMMi maturity levels and respective process areas are shown in the picture below.

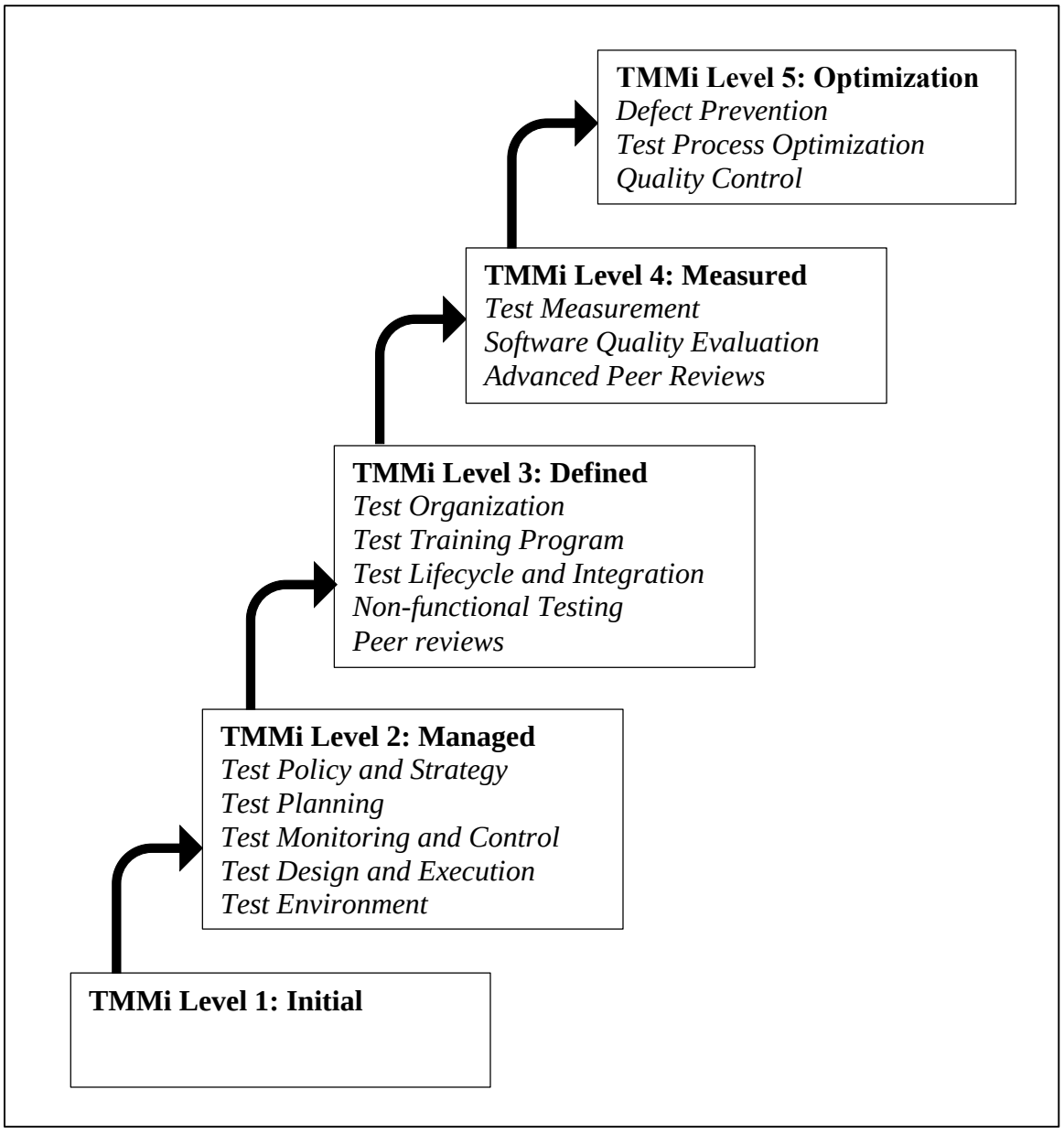

Figure 2. TMMi Maturity Levels and Process Areas

On **Level 1 – Initial** testing is an undefined process:

- Testing is performed in an ad-hoc way.

The main objective of testing at TMMi level 1 is to ensure that the software runs without major failures.

On **Level 2 – Managed** testing becomes a managed process:

- Test policy is established.

- Test strategy is established.
- Test plans are developed:
    - Product risks assessment performed.
    - Test approach is defined.
    - Test effort estimated.
    - Commitments to the test plan are established with stakeholders.
- Testing is monitored and controlled.
- Test design techniques are applied for deriving and selecting test cases from specifications.

The main objective of testing at TMMi level 2 is to verify that the product satisfies the specified requirements.

At TMMI level 2 testing is considered as post code, execution-based activity, which takes place relatively late in the development life-cycle, for example, during design or implementation.

On **Level 3 – Defined** testing is no longer considered as a phase that follows coding, testing is fully integrated into the development life-cycle and the associated milestones. Testing starts at early project stages, for example, during the requirements phase.

- A formal review program is implemented.
- Non-functional testing is included.
- A test organization and a specific test training program exist.

A critical distinction between TMMi maturity level 2 and 3 is the scope of standards, process descriptions, and procedures. At maturity level 3 the set of organization's standard test processes is established and improved over time. Project specific test processes are tailored from the organization's set of standards, and thus more consistent.

On **Level 4 – Measured** testing is a thoroughly defined, well-founded and measurable

process.

- Test measurement program is implemented.
- Reviews and inspections are fully integrated with the dynamic testing process.

On **Level 5 – Optimization** an organization is capable of continually improving its processes based on quantitative understanding of statistically controlled process. An optimized test process, as defined by the TMMi, is the one that is:

- Managed, defined, measured, efficient and effective;
- Statistically controlled and predictable;
- Focused on defect prevention;
- Supported by automation as much is deemed an effective use of resources;
- Able to support technology transfer from the industry to the organization;
- Able to support re-use of test assets;
- Focused on process change to achieve continuous improvement.

3.1.2 Structure of the TMMi

TMMi model consists of the following main components:

- Maturity Levels
- Process Areas
- Specific Goals
- Specific Practices

Maturity Level is an evolutionary stage of test process improvement. Each level progressively develops an important part of the organization's test process.

Each maturity level consists of several process areas (PA). Each process area identifies a cluster of test related activities on which an organization should focus to improve its test

process.

A specific goal (SG) describes the unique characteristics that must be present to satisfy the associated process area.

A specific practice (SP) is the description of an activity that is considered important in achieving the associated specific goal.

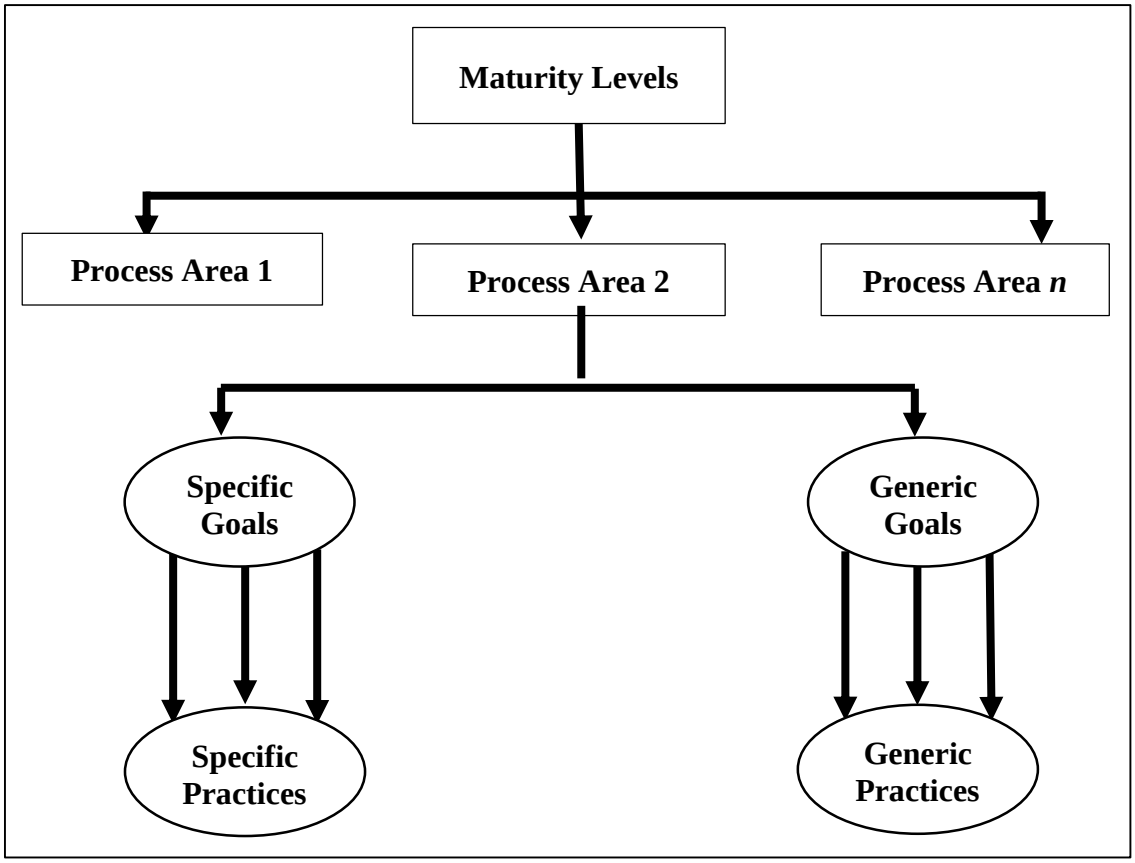

Figure 3. TMMi Structure and Components

3.2 RTMM Overview

The maturity of regression testing process is proposed to be assessed on the basis of the following characteristics of the process:

- The level of formalization of the regression testing process;
- Regression testing strategies employed:
    - Scripted testing vs. exploratory testing:

        - The amount of scripted testing effort vs. exploratory testing effort;
    - Manual testing vs. automated testing:
        - The amount of automation vs. manual testing effort;
    - ‘Retest all’ vs. selective testing;
- Regression testing optimization techniques employed;
- Regression testing optimization methods employed:
    - Systematic methods vs. non-systematic methods;
- The ability of the process to be measured.

The level of the maturity of regression testing process depends on the presence of these characteristics, the maturity of these characteristics in terms of the effectiveness of used approaches, and the adequacy and applicability of these approaches to certain contexts.

As TMMi, RTMM has a staged architecture. Five levels of the maturity of regression testing process are proposed. These levels are:

1 Initial –> 2 Managed –> 3 Measured –> 4 Optimized –> 5 Integrated

Maturity levels represent an evolutionary path to regression testing process improvement. Each level is associated with certain maturity goals and actions (that are similar to TMMi specific goals and practices). Maturity goals describe process characteristics that must be present to satisfy particular maturity level, and actions describe what have to be done in order to achieve these characteristics.

Regression testing maturity model is depicted below.

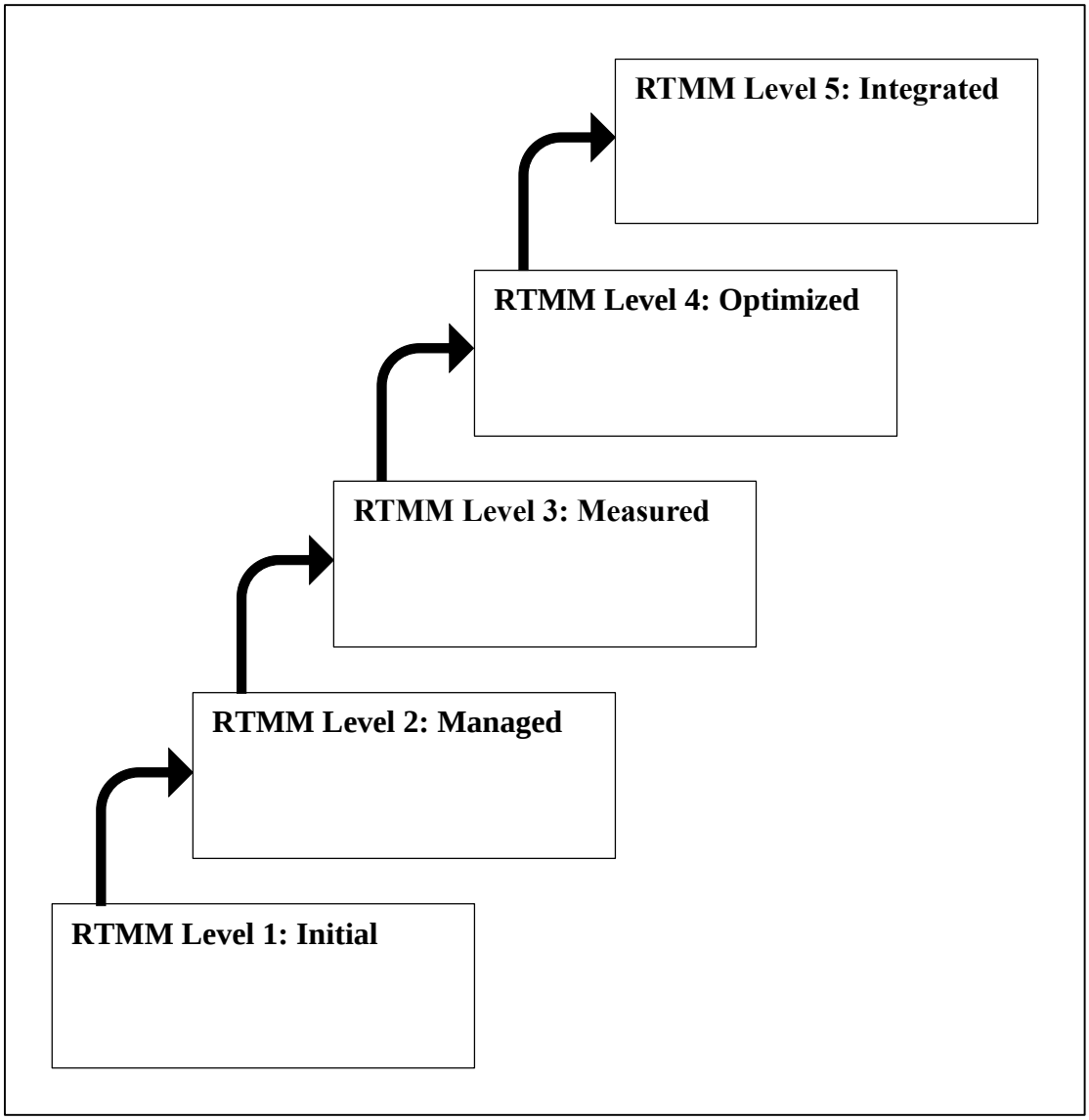

Figure 4. RTMM Maturity Levels

An overview of the maturity levels of the regression testing process is presented below.

3.2.1 RTMM Maturity Level 1: Initial

At RTMM Level 1 regression testing is chaotic:

- Not defined and not planned in advance.
- The approach is rather exploratory-experience based.
- Execution is mostly manual.
- The execution is not systematic and is triggered when necessary.

3.2.2 RTMM Maturity Level 2: Managed

At RTMM Level 2 regression testing is a managed process:

- The goals and purposes of the regression testing are defined.
- The strategy for regression testing is defined:
    - The approach is mostly systematic. Exploratory testing is a complementary method.
    - Test execution is still mostly manual. Automation can be partially implemented.
- The approach to regression testing is defined:
    - Selective testing approach is used.
    - Certain optimization techniques are used, mostly minimization.
    - Test cases are mostly selected based on error guessing, experience and knowledge of the system, or specific guidelines based on risk analysis and coverage criteria.
- Regression testing execution is rule-based and systematic.

3.2.3 RTMM Maturity Level 3: Measured

At RTMM Level 3 regression testing process is measured, analyzed and evaluated:

- Process performance is measured.
- Metrics are defined and applied to measure, assess and improve the regression testing process if necessary.

3.2.4 RTMM Maturity Level 4: Optimized

At RTMM Level 4 regression testing tended to be optimized:

- Based on the evaluation of the current process:
    - Employed optimization techniques and methods are revised and

improved if necessary;

  - More systematic optimization methods are preferred.

- The amount of automation increased.

3.2.5 RTMM Maturity Level 5: Integrated

At RTMM Level 5 regression testing is integrated into development process:

- Execution is mostly automated. Manual testing is done only in cases when automation is impossible, too hard or impractical. Exploratory testing is a complementary method.

- Regression testing is executed as a part of Continuous Integration (CI)*.

Maturity level 5 is considered to be optional since it strongly depends on external context – the software development methodology applied to a project, and the application of Continuous Integration practices in the software development process.

3.2.6 RTMM Maturity Goals

The summary of RTMM maturity goals per each maturity level is presented below.

(RTMM) Level 2: Managed

- (RTMM) Goal 1: Establish regression testing process
  - (RTMM) Sub Goal 1.1: Establish regression testing goals and objectives
  - (RTMM) Sub Goal 1.2: Establish regression testing strategy
  - (RTMM) Sub Goal 1.3: Establish regression testing approach
- (RTMM) Goal 2: Establish regression testing process performance measurement and analysis process

(RTMM) Level 3: Measured

- (RTMM) Goal 3: Measure and analyze regression testing process performance
- (RTMM) Goal 4: Determine regression testing process improvements

(RTMM) Level 4: Optimized

- (RTMM) Goal 5: Implement regression testing process improvements
    - (RTMM) Sub Goal 5.1: Increase amount of automated regression testing

(RTMM) Level 5: Integrated

- (RTMM) Goal 6: Incorporate regression testing into development process

3.2.7 RTMM Maturity Actions

The actions required for the achievement of the desired maturity level of the regression testing process are presented in the context of TMMi model. There are several reasons for such approach:

- Regression testing, as part of the overall testing process, cannot be considered in isolation of it. Referring to TMMi can give a better overview of the master process. All activities related to test planning and execution can directly or indirectly influence the process of regression testing. Some activities, like test execution and defects reporting, are common, and do not require any special consideration in the context of regression testing.

- TMMi model contains related activities that can be either elaborated for regression testing process, or used as a basis for defining specific RTMM actions. Therefore, RTMM actions are defined within or based on related TMMi practices and corresponding RTMM levels are incorporated into the respective TMMi levels. The arrangement of RTMM levels within TMMi model is depicted below.

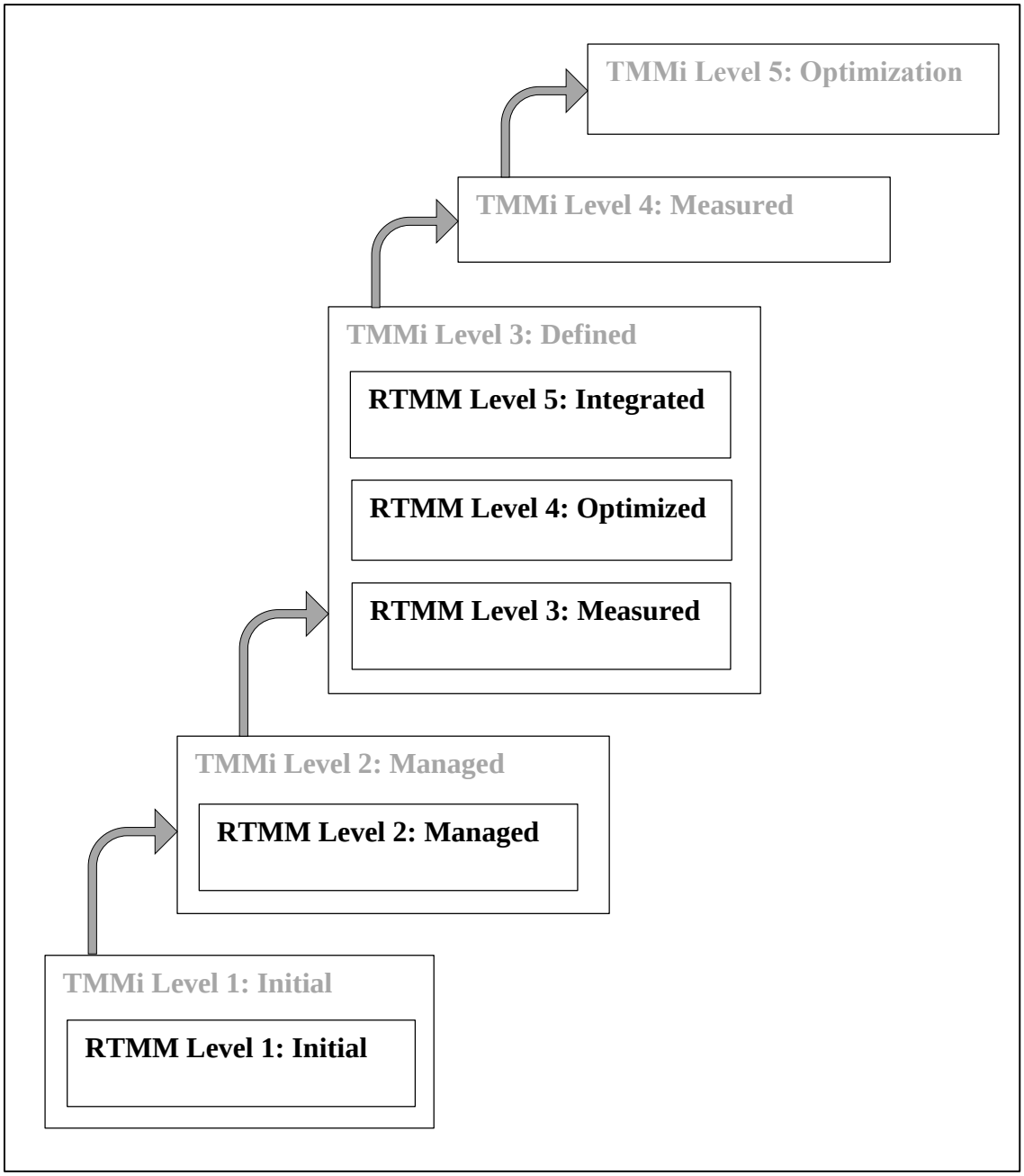

Figure 5. RTMM in the context of TMMi

RTMM actions and related TMMi levels and practices are presented below.

	TMMi	**RTMM**
Level	TMMi Level 2: Managed	RTMM Level 2: Managed

Process Area	PA 2.1 Test Policy and Strategy	
Goal	SG 1 Establish a Test Policy	(RTMM) Goal 1: Establish regression testing process (RTMM) Sub Goal 1.1: Establish regression testing goals and objectives
Practice	SP 1. 2 Define test policy	
RTMM Actions		
1. Define the risks addressed by regression testing. 2. Define the goals of regression testing, and / or the quality goals expected to be accomplished by employing regression testing.		

	TMMi	**RTMM**
Level	TMMi Level 2: Managed	RTMM Level 2: Managed
Process Area	PA 2.1 Test Policy and Strategy	
Goal	SG 2 Establish a Test Strategy	(RTMM) Goal 1: Establish regression testing process (RTMM) Sub Goal 1.2: Establish regression testing strategy
Practice	SP 2.2 Define test strategy	
RTMM Actions		
Define the strategy for regression testing. *Examples of topics to be addressed as part of regression testing strategy may include*		

the following:

- *Testing approach to be employed: systematic testing, exploratory testing, or combination of them;*
- *Testing approach to be employed: manual testing, automated testing, or combination of them;*
- *Automation strategy;*
- *Regression testing application policy:*
    - *Triggers for regression testing (e.g. system enhancements, bug fixes, system configuration changes, etc.);*
    - *Execution schedule:*
        - *Periodic execution (e.g. daily, weekly, monthly, etc.);*
        - *Rule-based execution (e.g. after any change, after changing critical parts only, after certain amount of changes, etc.).*

	TMMi	**RTMM**
Level	TMMi Level 2: Managed	RTMM Level 2: Managed
Process Area	PA 2.1 Test Policy and Strategy	
Goal	SG 3 Establish Test Performance Indicators	(RTMM) Goal 2: Establish regression testing process performance measurement and analysis process
Practice	SP 3.1 Define test performance	

	indicators	

RTMM Actions

1. Define the objectives of regression testing process performance measurement and analysis.

2. Define performance indicators for regression testing process.

   Examples of performance indicators for regression testing process may include the following:

   - *Direct metrics:*
     - *Total number of test cases in regression suite;*
     - *Total number of regression test cases executed;*
     - *Regression test coverage (overall, distributed across components);*
     - *Number of test cases in regression test suite that resulted in logging defects;*
     - *Number of valid regression defects found during regression testing;*
     - *Number of valid regression defects found during operational use of a software;*
   - *Indirect (computed) metrics:*
     - *Regression suite efficiency;*
     - *Regression testing effectiveness.*

3. Define how regression testing performance indicators will be obtained and stored

4. Define how regression testing performance indicators will be analyzed and

reported

	TMMi	RTMM
Level	TMMi Level 2: Managed	RTMM Level 2: Managed
Process Area	PA 2.2 Test Planning	
Goal	SG 2 Establish a Test Approach	(RTMM) Goal 1: Establish regression testing process (RTMM) Sub Goal 1.3: Establish regression testing approach
Practice	SP 2.2 Define the test approach	
RTMM Actions		

Define the approach to regression testing.

Examples of topics to be addressed as a part of regression testing approach may include the following:

- *At what level(s) regression testing is performed (e.g. component level, system level, business level);*
- *Types of testing to be performed for regression testing (e.g. functional testing, non-functional testing);*
- *Types of test cases to be used for regression testing (e.g. GUI test cases, functional test cases, system test cases, positive test cases, negative test cases, boundary / corner test cases, etc.);*
- *Regression testing approach to be employed: 'Retest all' or selective testing;*

- *The source of regression test cases (e.g. re-use existing test cases, create dedicated regression test cases);*
- *Regression testing optimization techniques to be applied: minimization, selection, prioritization;*
- *The methods for test suite minimization, test case selection and prioritization to be applied;*
    - *Manual / automatic;*
- *Regression test suite maintenance policy (triggers, actions, responsibilities);*
- *The scope and coverage of regression testing:*
    - *Items and features to be regression tested;*
    - *Criteria for the selection of test cases (modification-based, coverage-based, risk analysis-based, etc.);*
    - *Expected regression test coverage;*

	TMMi	**RTMM**
Level	TMMi Level 3: Defined	RTMM Level 3: Measured
Process Area	PA 3.1 Test Organization	
Goal	SG 4 Determine, Plan and Implement Test Process Improvements	(RTMM) Goal 3: Measure and analyze regression testing process performance
Practice	SP 4.1 Assess the organization's	

	test process	
RTMM Actions		
1. Obtain regression testing performance indicator data. 2. Analyze and interpret regression testing performance indicator data.		

	TMMi	**RTMM**
Level	TMMi Level 3: Defined	RTMM Level 3: Measured
Process Area	PA 3.1 Test Organization	
Goal	SG 4 Determine, Plan and Implement Test Process Improvements	(RTMM) Goal 4: Determine regression testing process improvements
Practice	SP 4.2 Identify the organization's test process improvements	
RTMM Actions		
Based on regression testing performance indicator data and on the assessment of the efficiency and effectiveness of the regression testing process determine problem areas and possible regression testing improvements. *Examples of possible improvement areas may include the following:* • *Revise employed regression testing optimization techniques. Consider using different / multiple techniques:* o *Minimization;* o *Selection;*		

- *Prioritization;*

- *Revise employed regression testing optimization methods;*
  - *Consider using advanced methods;*
  - *Consider using automatic methods;*
- *Consider increasing the amount of automated testing.*

	TMMi	**RTMM**
Level	TMMi Level 3: Defined	RTMM Level 4: Optimized
Process Area	PA 3.1 Test Organization	
Goal	SG 4 Determine, Plan and Implement Test Process Improvements	(RTMM) Goal 5: Implement regression testing process improvements
Practice	SP 4.4 Implement test process improvements	
RTMM Actions		
Continually implement regression testing process improvements.		

	TMMi	**RTMM**
Level	TMMi Level 3: Defined	RTMM Level 5: Integrated
Process	PA 3.3 Test Lifecycle and	

Area	Integration	
Goal	SG 2 Integrate the Test Lifecycle Models with the Development Models	(RTMM) Goal 6: Incorporate regression testing into development process
Practice	SP 2.1 Establish integrated lifecycle models	
RTMM Actions		
Integrate automated regression tests with Continuous Integration environment.		

4 Regression Testing Metrics Model

Further, a model is proposed for dynamic analysis of the regression testing process.

Dynamic analysis of regression testing process is proposed to be performed on the basis of the process performance indicator data obtained during the execution of the process. The proposed model represents a set of core metrics that can be used to measure the efficiency and effectiveness of regression testing process. It should be taken into account that the proposed metrics are of general nature. Any specific metrics should be defined and developed within a certain context, considering specific needs and concerns.

The proposed metrics are based on core testing metrics and developed with the consideration of various metrics identification methodologies. For that reason, key concepts of software measurement and analysis has been widely studied.

4.1 Introduction to Software Metrics

4.1.1 Software Measurement and Analysis

Nowadays software measurement and analysis has become an integral part of good software engineering practices. Measurement and analysis activities involve gathering of the quantitative data about products, processes and projects, and analysis of this data with the aim to get objective information, which enables to [2]:

- **characterize**, or gain insight into products, processes and projects;
- **evaluate**, or assess the current status of products, processes and projects against targets and benchmarks;
- **predict** values of the various attributes of products, processes and projects;
- **improve** the various attributes of products, processes and projects by identifying risks and problematic areas and applying corrective actions.

The results of software measurement and analysis serve as a basis for management decisions making.

4.1.2 Software Metrics

Software metric is a key concept in software measurement and analysis area. In different sources [2, 5], metric is referred to as a unit of measurement, measurement based technique, or a quantifiable measurement, which is applied to the different aspects of a software product, process or project with the aim to measure and quantify them. IEEE Standard for a Software Quality Metrics Methodology (1061-1998), defines Software Metric or Software Quality Metric as “a function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which software possesses a given attribute that affects its quality” [1].

Metrics can be classified into various categories [2, 3, 4] depending on their targets, purposes, or the ways of derivation and computation. The most notable of them are the following:

- In general, all metrics are divided into three main categories: process metrics, product metrics and project metrics. IEEE Standard for a Software Quality Metrics Methodology (1061-1998) gives the following definitions for process and product metrics: Process Metric is “a metric used to measure characteristics of the methods, techniques, and tools employed in developing, implementing, and maintaining the software system”, and Product Metric is “a metric used to measure characteristics of any intermediate or final product of the software development process” [1]. Project Metric can be defined as a metric used to measure various characteristic and execution of a software development project, like cost, schedule, and resources.

- From observation or measurement perspective, metrics can be classified into primitive (direct) and computed (indirect / derived) metrics. Primitive or direct metrics are those that can be directly observed / measured and no other attributes or entities are required for computation., for example Number of Defects. Computed or indirect / derived metrics are those that cannot be directly observed but are computed in some manner from other measurements and / or metrics, for example Defect Density, which is derived from the number of defects divided by

the size of the software.

- From objectivity perspective, metrics can be classified into objective and subjective metrics. Objective metrics should provide objective information, and always result in identical values, even if measured by two or more qualified observers. Subjective metrics may result in different values as subjective judgment of qualified observers is involved in derivation of measured value.

In order for a software metric to be useful – be capable to describe a product or a process, indicate the current status of a product or a process, or predict product or process parameters – it has to meet several quality requirements [4]:

- Metric must be simple and precisely definable. It should be clear how it can be evaluated;
- Metric must be objective to the greatest extent possible;
- Metric must be easily obtainable;
- Metric must be reliable and valid. It should measure exactly what it is intended to measure;
- Metric must be robust, relatively insensitive to insignificant changes in the product or process.

4.1.3 Software Measurement and Analysis Frameworks and Standards

In order to define useful metrics – both in terms of applicability to a particular context and in terms of compliance with the quality requirements – several software measurement and analysis frameworks and standards have been proposed [1, 6, 7, 8, 15]. The main purposes of these standards are the following:

- To define and standardize software measurement and analysis processes – activities and tasks that are necessary to successfully identify, define, select, apply and improve measurement within a software development project [15];
- To propose software metrics identification methodologies – systematic approaches to establishing quality requirements and identifying, implementing,

analyzing and validating the process and product software quality metrics for a software system [1].

The most well-known of them are:

- [Process] Process Improvement and the Capability Maturity Model (CMM), A Support Process Area at Maturity Level 2, Measurement and Analysis;
- [Process] ISO International Standards, Systems and software engineering - Measurement process: ISO/IEC 15939:2007;
- [Methodology] Goal-Question-Metrics (GQM) Approach;
- [Methodology] IEEE Standard for a Software Quality Metrics Methodology: 1061-1998.

Mentioned standards do not prescribe specific metrics, but provide a guideline on how to define, implement and evaluate them. Despite the differences in details, the given approaches have much in common:

- The process of software measurement and analysis involves the following activities [6]:
    1. Specifying the objectives of measurement and analysis such that they are aligned with identified information needs and objectives;
    2. Specifying the measures, analysis techniques and mechanism for data collection, data storage, reporting and feedback;
    3. Implementing the collection, storage, analysis and reporting of the data;
    4. Providing objective results that can be used in making informed decisions and taking appropriate corrective actions.
- The approach for identification of appropriate software metrics involve the following three steps:
    1. On conceptual level, identification of quality requirements for a software product and / or process;

2. On operational level, identification of the ways of how the respective quality requirements can be measured and assessed;

3. On quantitative level, definition of measures and metrics.

4.1.4 Software Testing Metrics

Software testing metrics are a subset of overall software metrics. While software metrics are applicable to the whole development process, from initiation, when cost must be estimated, to monitoring the reliability of the end product in the field, software metrics are concentrated on the testing phase and act as indicators of software quality and fault proneness [9].

As mentioned above, testing metrics can be divided into two categories: test process metrics and test product metrics.

Test process metrics provide information about preparation for testing, test execution and test progress. They are utilized to monitor the progress of testing, status of design and development of test cases and outcome of test cases after execution. They don't provide information about the test state of the product and are primarily of use in measuring progress of the test phase. Process metrics describe the effectiveness and quality of the processes that produce the software product. Examples of test process metrics:

- Number of test cases designed;
- Number of test cases executed;
- % of test cases executed;
- % of test cases passed;
- % of test cases failed;
- Total actual execution time / total estimated execution time;
- Average execution time of a test case. [2]

Test product metrics provide information about the test state and testing status of a

software product and are generated by test execution. By using these metrics, it is possible to measure the products test state and the indicative level of quality, useful for product release decisions. Examples of test product metrics:

- Estimated time for testing;
- Actual testing time;
- % of time spent = (Actual time spent / Estimated testing time) * 100;
- Average time interval between failures;
- Maximum and minimum failures experienced in any time interval;
- Average number of failures experienced in time intervals;
- Time remaining to complete the testing. [2]

The importance and purposes of software testing metrics are manifested in the following [2, 10]:

- Testing metrics assist to understand the current position of a project;
- Testing metrics aid in prioritizing activities to reduce the risk of schedule over-runs on software releases;
- Testing metrics provide a basis for estimation and facilitates planning for closure of the performance gap;
- Testing metrics provide a means for control / status reporting;
- Testing metrics enable to identify risk areas that require more testing;
- Testing metrics enable to identify potential problems and areas of improvement;
- Testing metrics provide an objective measure of the effectiveness and efficiency of testing;
- Testing metrics enable to set quality benchmarks for several tasks and processes involved in the development.

There are many various software testing metrics, and a lot of literature is available where different kinds of testing metrics are compiled, described, analyzed, new metrics proposed, etc. [2, 3, 5, 9, 11, 12, 13, 14]. Some of the most common and widely used software testing metrics / metric types are presented in Table1.

Table 2. Software Testing Metrics

Metric / Metric Type	**Description**
Test Coverage	In general, two main conceptions of Test Coverage exist: Test Design Coverage – shows the extent to which the functionality of the product is covered by test cases. Expressed as percentage or ratio of covered requirements (or other items of interest) to the total number of requirements. The respective metric is used as an indicator of quality of test design and used to improve test coverage. Test Execution Coverage – shows the extent to which the functionality of the product has been covered during testing. Expressed as percentage or ratio of the requirements (or other items of interest) covered during testing to the total number of requirements. The respective metric is used as an indicator of the completeness of testing, and can serve as a criterion to stop or continue testing. Coverage could be with respect to requirements, business flows, use cases, etc.
Number of Defects / Defect Discovery Rate	The respective metric shows the total number of defects found across a given period of time.
Defect Distribution Metrics	The respective metrics show the distribution of existing defects by various criteria: category, severity, priority, state, component, etc.

Metric / Metric Type	Description
Defect Density	Defect Density can be calculated against different aspects: As the ratio of valid defects to the total size of software product under test, expressed in number of (thousands) Lines of Code (KLOC), Functional Points (FP), requirements, etc. This metric indicates the quality of the software. It can be used as a basis for estimating defects to be addressed in the next phase or the next version. As the ratio of valid defects to the number of executed test cases. This metric indicates the stability of the software.
Test Case Execution Statistics	The respective metrics provide an overall summary of test execution activities. For example: % of executed / not completed / passed / failed / blocked / etc. test cases Actual execution time vs. estimated execution time
Test Case Efficiency / Test Case Defect Density	The respective metric is calculated as the ratio of the number of test cases that resulted in logging defects to the total number of executed test cases. This metric indicates the effectiveness of the test cases and the stability of the software.

Metric / Metric Type	Description
Test Effectiveness / Defect Removal Efficiency and Defect Leakage	Test Effectiveness / Defect Removal Efficiency metric shows the efficiency of removing defects by internal testing before delivering to customer. It is calculated as the percentage of defects caught by internal testing team vs. the total number of defects reported against the product, both during the testing life cycle and post-delivery to end-customer. Test Efficiency= (DT / (DT + DU)) * 100 This metric indicates the effectiveness of testing, and serves as an indirect indicator of the quality of the product. Defect Leakage metric, in contradistinction to Defect Removal Efficiency, measures the percentage of defects missed during internal testing. Defect Leakage = (DU / (DT + DU)) * 100 This metric indicates the efficiency of internal testing. Where, DT = Number of valid defects identified during testing. DU = Number of valid defects identified by user after release of application.
Test Execution Productivity	Test Execution Productivity metric determines the number of test cases that can be executed per person-day of effort. This can be used for estimating future testing activities.

4.2 Regression Testing Metrics

As proposed by various metric identification methodologies [1, 7], the indication of the right metrics starts from the definition of quality requirements to the software product or process. The basic requirements to regression testing process are the following:

- Regression testing should ensure that no regression defects have been introduced into previously working software after changes.
- Regression testing should effectively reveal regression defects;

The fulfillment of the first requirement can be assessed through:

- Evaluation of the amount of functionality covered by regression test suite;

The fulfillment of the second requirement can be assessed through:

- Evaluation of the capability of regression test suite to reveal regression defects;
- Evaluation of the ratio of defects found during regression testing to the regression defects found during the operational use of the software.

The assessment of the capability to reveal regression defects will answer the question: "Do we select the right test cases for regression testing?". The assessment of the amount of functionality covered by regression testing will answer the question: "Do we do enough of regression testing?".

The main purposes of the proposed regression testing metrics are the following:

- Assess the efficiency of regression testing in terms of coverage and the capability of regression test sets to reveal regression defects;
- Assess the effectiveness of regression testing in reducing regression defects.

The proposed regression testing metrics are presented in the following section.

4.2.1 Regression Test Coverage

Name	**Regression Test Coverage (RTC)**
Description	The metric shows the extent to which software functionality is covered by regression test suite. Measured as the ratio of items of interest covered by the regression test suite to the total number of the respective items. Coverage could be with respect to requirements, use

	cases, business flows, etc.
Impact	If minimum target coverage is defined for regression testing, the metric would indicate the extent to which the respective requirement is satisfied. If the actual test coverage is less than the expected one, corresponding corrective actions must be taken in order to improve the situation. For example, revise regression test suite: replace inefficient (from coverage point of view) test cases with the ones that provide better coverage; extend regression test set with more test cases, etc. The respective metric may serve as indirect indicator of the quality of maintenance of regression test suites. It might indicate that regression test suites are out of sync with the ongoing development, especially when static regression test suites are used for regression testing.
Measures	• Total number of items of interest (TI); • Number of items of interest, covered by the regression test set (CI). Items of interest: requirements, use cases, business flows, etc. Measurements can be made both on the overall product level and on component level.
Computation	ORTC = (CI / TI) * *100%*

4.2.2 Regression Test Suite Efficiency

Name	**Regression Test Suite Efficiency (RTSEffcy)**
Description	The metric shows the efficiency of a regression test suite in terms of how many effective test cases it contains (how many test cases within a test suite are capable to reveal regression defects). Measured as the ratio of test cases resulted in logging valid regression defects to the

	total number of executed test cases. Relevant only in conjunction with Regression Testing Effectiveness (RTE) metric (the overall effectiveness of regression testing)**.**
Impact	The metric is a direct indicator of the quality of regression test suites and the effectiveness of test selection and minimization methods employed. Low efficiency of a regression suite might indicate that the suite contains irrelevant or redundant test cases. If overall effectiveness of regression testing is normal or high: • And there are no strict requirements to test coverage, the respective metric might indicate that: o The regression suite contains redundant test cases. o Unnecessary effort is spent on regression testing. If overall effectiveness of regression testing is low: • The respective metric might indicate that wrong (irrelevant) test cases are selected for regression testing. If any of these issues are valid employed test minimization and selection methods need to be revised.
Measures	• Number of (effective) test cases in the regression test suite that resulted in logging of valid regression defects (ETC); • Total number of executed test cases in the regression test suite (TTC).
Computation	RTSEffcy = (ETC / TTC) * *100%*

4.2.3 Regression Test Suite Effectiveness

Name	**Regression Test Suite Effectiveness (RTSEffness)**
Description	The metric shows the effectiveness of a regression suite in detecting regression defects (the capability of a regression test suite to reveal regression defects). Measured as the ratio of the number of valid defects detected during regression testing to the number of executed test cases. Relevant only in conjunction with Regression Testing Effectiveness (RTE) metric**.**
Impact	The metric is a direct indicator of the quality of regression test suites and the effectiveness of test selection and minimization methods employed. Low effectiveness of a regression suite might indicate that the suite contains irrelevant test cases or insufficient amount of test cases (insufficient regression test coverage). If overall effectiveness of regression testing is low: • And the effectiveness of a regression suite is low, then the respective metric might indicate that the regression suite contains irrelevant test cases and / or insufficient amount of test cases (insufficient regression test coverage). • And the effectiveness of a regression suite is normal or high, then the respective metric might indicate that the regression suite contains insufficient amount of test cases (insufficient regression test coverage). If any of these issues are valid employed test minimization and selection methods need to be revised.
Measures	• Total number of valid regression defects detected during

	regression testing (TD); • Total number of executed test cases in the regression test suite (TTC).
Computation	RTSEffness = (TD / TTC) * *100%*

4.2.4 Regression Testing Effectiveness

Name	**Regression Testing Effectiveness (RTE)**
Description	The metric shows the effectiveness of regression testing in reducing regression defects. Measured as the ratio of the number of valid regression defects detected during regression testing to the total number of valid regression defects.
Impact	This metric is the main indicator of the effectiveness of regression testing. If the effectiveness of regression testing is low, then it is necessary to review the whole process (strategies, approaches, optimization techniques and approaches, etc.).
Measures	• Total number of valid regression defects detected during regression testing (TDR); • Total number of valid regression defects (TD).
Computation	RTE = (TDR / TD) * *100%*

5 Practical Application

The following section contains analysis and evaluation of a real regression testing practice based on the proposed models. The objectives of this case study are the following:

- To analyse and evaluate a real regression testing practice with the aim of identifying areas for improvement.
- To examine the application of the proposed models to the real world context;

5.1 Background

Regression testing practice taken as an example for the examination of the proposed models is a real regression testing process that requires thorough analysis and improvements.

The regression testing practice under consideration adopted in a *Project*, which is carried out by IT Organization belonging to a *Company*. The IT Organization is responsible for development of enterprise software applications, and the *Project* is one of the largest projects within the *Company* initiated for developing a core business *Application*.

5.1.1 Software Application

Application is a complex core business application. From functional point of view, it consists of three major functional domains. Two of them encompass functionality related to the two main businesses of the *Company*, and one is a cross-functional domain, which encompasses functionality common to these businesses. Each functional domain consists of a number of functional areas (e.g. 'Customer Maintenance' in cross-functional domain, and in total cross-functional domain consists of ca. 15 functional areas), and each functional area in its turn consists of a number of functional requirements and use cases (e.g. 'Create Customer').

From technical point of view, *Application* is a GUI client/server application based on Java technologies. *Application* is a part of large enterprise system – it is integrated with multiple external enterprise systems and applications.

The *Application* is intended to replace the currently used application. The new *Application* is being built using modern technologies and expected to provide extended coverage of business processes.

5.1.2 Development Methodology

Application is developed using incremental Waterfall model.

5.1.3 Release Schedule

The development of the *Application* is divided into four major releases. Each major release is delivered to production and includes a considerable amount of functionality.

Major releases are normally divided into twelve sub-releases. The first six sub-releases are part of development phase, during which the *Application's* functionality is incrementally implemented, and after which corresponding major release is delivered to production. The next six releases are part of maintenance phase, during which change requests are implemented and delivered to production.

The development of a next major release and the maintenance of the one which is currently in production are done in parallel.

Each sub-release may have multiple bug-fix and hot-fix releases.

Currently the second major release is in production, and the third one is at the end of its development phase.

5.1.4 Test Organization

A *Test Team* of the *Project* is responsible for system test phase.

In general test strategy can be defined as classical systematic scripted testing with clearly defined test processes and procedures. Test process consists of test planning, test preparation and test execution phases. During test preparation phase detailed test cases (GUI, functional, system, integration) are developed. Test execution phase includes testing of new functionality and regression testing. Defect management process is also defined.

5.1.5 Regression Testing

Regression testing is a continuous activity within the considered test process. It is executed for each sub-release (including bug-fix and hot-fix releases) of a major release which is currently in production, and for each sub-release of a major release which is at the end of its development phase. According to the statistical data, regression testing consumes up to 65% of the overall test effort.

Besides the amount of time and effort required for regression testing, there is also a human factor, which also plays an important role. Due to the fact that regression suites are relatively static and execution is 100% manual, the respective activity is considered by *Test Team* as demotivating.

A survey was conducted among the members of the *Test Team* to determine the problems they experience with the current approach to regression testing. Most of the team members have agreed in opinion that:

- Regression suites are not maintained properly:
    - Regression suites include outdated and duplicated test cases;
    - Regression test cases marked as requiring correction are not corrected for a long period of time.
- Running the same test cases all the time is very demotivating, people stop noticing problems, ignore things that worked before.

Due to all above mentioned facts, regression testing is recognized by the management and members of the *Test Team* as a problematic area in the test process that requires improvements.

Thereby the considered regression testing practice has been selected as a subject for the present case study:

- It requires thorough analysis and improvements;
- It allows to examine and evaluate the proposed models in terms of their applicability to the real world context.

5.2 Analysis

5.2.1 Process Static Analysis

Static analysis of the regression testing process under consideration is performed based on RTMM model, the results are presented below.

Table 3. Regression Testing Practice Analysis using RTMM Model

RTMM Goal	Current Practice	Status / Comments
(RTMM) Goal 1: Establish regression testing process	**Regression testing is a defined process**	**YES**
(RTMM) Sub Goal 1.1: Establish regression testing goals and objectives	**The goal of regression testing is defined** Regression testing goal is declared as 'Ensure that previous tested application components are still working correctly and no new defects have been introduced through changes in application itself or its environment (new features, defect fixes, interface changes or external systems). Ensure high test coverage at the end of test execution of each release'.	**YES**

(RTMM) Sub Goal 1.2: Establish regression testing strategy	**The strategy for regression testing is defined** • 100 % systematic scripted testing; exploratory testing not employed. • 100% manual testing. • Automation strategy is defined. • Application policy: rule-based execution (every sub-release (including bug-fix and hot-fix releases) of a major release which is currently in production, and every sub-release of a major release which is at the end of its development phase).	**YES** Automation strategy is defined, and to some extent automated testing is implemented. But because of the specifics of the selected approach, the results of automated testing are unstable and unreliable. Due to this fact they are never taken into account during regression testing. Test cases covered by automated tests are still executed manually. Nevertheless, one of the goals for regression testing defined in the test strategy is 'to reach a high automation rate'.
(RTMM) Sub Goal 1.3: Establish regression testing approach	**The approach to regression testing is defined** • Test types to be performed are defined in the regression test strategy: 'Regression testing comprises almost all test types. It will be defined during test execution depending on problem	**YES** • Although test selection technique is declared in the regression test strategy, in most cases it is not

	areas of SUT and functionalities where code changes have been implemented'. Regression suites normally include various test types – GUI, functional, integration. • Selective testing approach is employed. • Regression test cases are selected among the test cases created during test preparation phase. • Optimization techniques employed: minimization, selection. • Optimization methods employed: o Test selection approach is two-tiered; o At the first stage minimization technique is employed. A representative regression test suite is compiled from all existing test cases. Representative regression suite includes the most basic test cases of various types, and covering all functional areas. Test cases in representative regression suite are marked as 'regression test candidates'. Normally there are two	applied. Entire representative regression test suites are used for regression testing. Selection is applied only in case of hot-fix releases, and it is mostly experience-based and is done manually. • Although it is implied that every member of the *Test Team* is responsible for maintaining regression test cases, nobody actually follows this rule. As a result, regression test suites are of poor quality.

	representative regression suites – one extended for major release regression testing, another with limited scope for bug-fix and hot-fix releases. Representative regression suites are static and recompiled only for the subsequent major releases. o At the next stage selection technique is employed. Among the test cases contained in a representative regression test suite, a test suite for regression testing session is compiled. Selection is modification-based, coverage-based and risk analysis-based. o Selection is supported by an internal automatic tool; • Test maintenance policy is not strictly defined. It is implied that every member of the *Test Team* is responsible for maintaining regression test cases – deleting obsolete and duplicated test cases from representative test suites, updating and improving outdated test cases, etc. • Test coverage criteria defined:	

	○ All functional areas should be covered with regression tests; ○ Regression test suites should contain 40% of total number of test cases.	
(RTMM) Goal 2: Establish regression testing process performance measurement and analysis process	**Regression testing process performance measurement and analysis process is not defined**	**NO**
(RTMM) Goal 3: Measure and analyze regression testing process performance	**Regression testing process performance is not measured**	**NO**

(RTMM) Goal 4: Determine regression testing process improvements	**Regression testing improvements are not defined and not implemented**	**NO**
(RTMM) Goal 5: Implement regression testing process improvements	**Regression testing improvements are not defined and not implemented**	**NO**
(RTMM) Sub Goal 5.1: Increase amount of automated regression testing	**The amount of test automation is increased**	**YES** Automated tests are developed, and automation coverage constantly increases, but as it was previously mentioned the results of automated regression testing are not taken into account, and the same test cases are still executed manually.

(RTMM) Goal 6: Incorporate regression testing into development process	**Automated regression tests are not integrated with Continuous Integration environment**	**NO**

5.2.2 Process Dynamic Analysis

Dynamic analysis of the regression testing process under consideration using the proposed RT Metrics Model has not been performed. The main reason is that the process is hardly measurable (see Table 3, Regression testing process performance measurement and analysis process is not defined). In this situation it is almost impossible to assess the effectiveness of the overall regression testing process (see Chapter 4.2.4 Regression Testing Effectiveness), because defects are not being distinguished as regression and non-regression. Without knowing the actual effectiveness of the process it is impractical to assess, analyse and evaluate the efficiency and effectiveness of regression test suites.

5.3 Evaluation

5.3.1 Process Evaluation

As a result of static analysis of the regression testing process under consideration conducted using RTMM model the following conclusions regarding the maturity of the process were made:

- The respective regression testing process is a well-defined process with clearly defined goals, strategies and approaches.
- It is on the second level of the process maturity scale according to RTMM (see Chapter 3.2.2 RTMM Maturity Level 2: Managed):
    - The process is managed;
    - The goals and purposes of the regression testing are defined;
    - The strategy for regression testing is defined:
        - The approach is systematic;
        - Test execution is mostly manual. Automation is partially implemented.
    - The approach to regression testing is defined:

        - Selective testing approach is used;
        - Certain optimization techniques are used: minimization and selection.
        - Test cases are selected based on experience and knowledge of the system, and specific guidelines based on risk analysis and coverage criteria.
    - Regression testing execution is rule-based and systematic.
- In order to improve its maturity level at least the following further actions need to be taken:
    - Establish and implement the process to measure and analyse the performance of regression testing;
    - Increase the proportion of automated testing.
- Certain problem areas and areas for improvement have been detected.

The following areas for improvement are detected:

- Regression testing automation;
- Optimization techniques and methods;
- Regression test suite maintenance.

The following proposals for improvement are made:

- Establish and implement the process to measure and analyse the performance of regression testing;
- Revise automation strategy, make it more stable and reliable. Increase the proportion of automated testing for regression testing. Replace manual effort with automated testing.
- Revise methods for test case selection. Make them more systematic, frequent and automated as possible.

- Establish regression test suite maintenance policy. Make it more systematic – define responsibilities and application policy.

- Incorporate exploratory testing into regression testing process. This can help to improve the motivation of the *Test Team* and overcome the well-known pesticide paradox issue, which consists in that repeating the same test cases over and over again makes them inefficient in finding defects.

5.3.2 Model evaluation

As a result of analysis of the regression testing process under consideration conducted using the proposed models the following conclusions regarding the applicability of these models to the real world context were made:

- RTMM facilitates analysis of a regression testing process – it allows systematically examine the process and reveal its strengths and weaknesses. This, in its turn, allows to determine areas for improvement. RTMM proposes general solutions to improve the process and increase the level of its maturity.

- Application of RTMM model to the regression testing process under consideration revealed problem areas that reflect the negative experience of the *Test Team* members mentioned in the survey (see Chapter 5.1.5 Regression Testing). Solving these problems would improve the overall motivation of the *Test Team*.

- Inability to apply RT Metrics Model to the regression testing process under consideration pointed to the importance of establishing a proper measurement program for regression testing.

6 Summary

The main goal of this thesis was to develop systematic methods for the analysis and evaluation of the process of regression testing. In order to achieve this goal an extensive review on regression testing was carried out: the most common regression testing methodologies and approaches were reviewed and systemized, various researches on regression testing were analysed and interpreted, also, in order to develop metrics for the assessment of regression testing process, key concepts of software measurement and analysis were studied.

As a result of this work two models for the analysis, evaluation and further improvement of the maturity, efficiency and effectiveness of regression testing process were introduced. One of the proposed models is intended for the analysis, evaluation and improvement of the level of process maturity, which is based on the evaluation of various process characteristics, like strategies and approaches employed. The other model represents a set of core metrics for the analysis and evaluation of the efficiency and effectiveness of regression testing. Both models can serve as reference and guideline models for regression testing process establishment, analysis, evaluation and further improvement.

The maturity model (RTMM) was applied for the analysis and evaluation of a real regression testing practice. As a result, key areas for improvement in the respective regression testing process have been detected and corresponding proposals for improvements were made. Also the applicability of the proposed models was examined. It was determined that the maturity model (RTMM) facilitates analysis, evaluation and improvement of a regression testing process – allows systematically examine the process, reveal its strengths and weaknesses, identify problem areas and possible solutions for improvement.

References

1. IEEE Standard for a Software Quality Metrics Methodology: IEEE Std 1061-1998 (R2009), IEEE Standards, 1998

2. S. U. Farooq, S. M. K. and N. Ahmad. Software measurements and metrics: role in effective software testing. Internal Journal of Engineering Science and Technology, Vol. 3, no. 1, 2011, pp. 671-680

3. Ming-Chang Lee, To Chang. Software Measurement and Software Metrics in Software Quality. International Journal of Software Engineering and Its Applications, Vol. 7, No. 4, July, 2013

4. Everald E. Mill. Software Metrics: SEI Curriculum Module SEI-CM-12-1.1. Carnegie Mellon University, Software Engineering Institute. December, 1988

5. Mr. Premal B. Nirpal, Dr. K. V. Kale. A Brief Overview of Software Testing Metrics. International Journal on Computer Science and Engineering, Vol. 3, No. 1, Jan 2011

6. CMMI for Development: Version 1.2. Carnegie Mellon University, Software Engineering Institute. August, 2006

7. Victor R. Basili, Gianluigi Caldiera, H. Dieter Rombach. The Goal Question Metric Approach. 1994

8. Measurement Frameworks and Standards, [WWW] http://www.dcs.qmul.ac.uk/~norman/papers/qa_metrics_article/section_7_standards.html (19.11.2015)

9. Kaur Arvinder, Suri Bharti, Sharma Abhilasha. Software Testing Product Metrics - A Survey. Proceedings of National Conference on Challenges & Opportunities in Information Technology (COIT-2007) RIMT-IET, Mandi Gobindgarh. March 23, 2007

10. Pusala, Ramesh. Operational Excellence through efficient Software Testing Metrics. Infosys, 2006

11. Testing Metrics, [WWW] http://www.mindlance.com/documents/test_management/testing_metrics.pdf (20.11.2015)

12. Important Software Test Metrics and Measurements, [WWW] http://www.softwaretestinghelp.com/software-test-metrics-and-measurements/ (20.11.2015)

13. Realizing Efficiency & Effectiveness in Software Testing through a Comprehensive Metrics Model, Infosys, [WWW] http://www.infosys.com/engineering-services/white-papers/Documents/comprehensive-metrics-model.pdf (20.11.2015)

14. Gregory Jose, Jusha Joseph. Test Metrics and KPI's, [WWW] http://www.ust-global.com/en/images/stories/pdf/Test_Metrics_and%20KPI_s.pdf (22.11.2015)

15. ISO International Standards, Systems and software engineering - Measurement process: ISO/IEC 15939:2007, 01.08.2007

16. Dorothy Graham, Erik Van Veenendaal, Isabel Evans, Rex Black. Foundations of Software Testing, ISTQB Certification, ISTQB Glossary

17. Waterfall model, [WWW] https://en.wikipedia.org/wiki/Waterfall_model (07.01.2016)

18. Agile Methodology, [WWW] http://agilemethodology.org/ (11.01.2016)

19. Regression Testing with an Agile Mindset, [WWW] http://blog.xebia.com/regression-testing-with-an-agile-mindset/ (11.01.2016)

20. David Parsons, Teo Susnjak, Manfred Lange. Influences on Regression Testing Strategies in Agile Software Development Environments. [WWW] https://www.academia.edu (05.02.2016)

21. Talby, D, Keren, A, Hazzan, O, & Dubinsky, Y. (2006). Agile software testing in

a large-scale project. IEEE Software, 23(4), 30 – 37

22. Test Maturity Model integration (TMMi): Release 1.0. TMMi Foundation, 2012

23. Inder P Singh. What is software regression? [WWW] http://inderpsingh.blogspot.com.ee/2011/04/what-is-software-regression.html (28.02.2016)

24. Emelie Engstrom, Per Runeson. A Qualitative Survey of Regression Testing Practices. In M. Ali Babar, Matias Vierimaa, Markku Oivo (Eds.), Product-Focused Software Process Improvement, LNCS, vol. 6156, Springer, Berlin, Heidelberg, 2010, pp. 3-16.

25. Zachman Framework. [WWW] https://en.wikipedia.org/wiki/Zachman_Framework (02.03.2016)

26. Maiqin Cui and Chengyao Wang, 2015. Cost-Benefit Evaluation Model for Automated Testing Based on Test Case Prioritization. Journal of Software Engineering, 9: 808-817.

27. Error guessing. [WWW] https://en.wikipedia.org/wiki/Error_guessing (20.03.2016)

28. Exploratory testing. [WWW] https://en.wikipedia.org/wiki/Exploratory_testing (20.03.2016)

29. E. Engström, P. Runeson and M. Skoglund (2010). A systematic review on regression test selection techniques. Information and Software Technology, 52, 14-30

30. S. Yoo, M. Harman (2012), Regression Testing Minimization, Selection and Prioritization: A Survey. Software Testing, Verification and Reliability, 22(2), 67-120

31. R. Miranda, O. Gomez, G. Rodriguez (2015), “15 Years of Software Regression Testing Techniques: A Survey”, International Journal of Software Engineering and Knowledge Engineering

32. R. P. Gorthi, A. Pasala, K. K. Chanduka and B. Leong, "Specification-Based Approach to Select Regression Test Suite to Validate Changed Software", Software Engineering Conference, 2008. APSEC '08. 15th Asia-Pacific, Beijing, 2008, pp. 153-160

33. M. Salehie, S. Li, L. Tahvildari, R. Dara, S. Li and M. Moore, "Prioritizing Requirements-Based Regression Test Cases: A Goal-Driven Practice", Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on, Oldenburg, 2011, pp. 329-332.

34. A. S. A. Ansari, K. K. Devadkar and P. Gharpure, "Optimization of test suite-test case in regression test", Computational Intelligence and Computing Research (ICCIC), 2013 IEEE International Conference on, Enathi, 2013, pp. 1-4.

35. M. Puleio, "How not to do agile testing," Agile Conference, *2006*, Minneapolis, MN, 2006, pp. 7 pp.-314.

36. M. Gittens, H. Lutfiyya, M. Bauer, D. Godwin, Y. W. Kim, P. Gupta (2002). An empirical evaluation of system and regression testing. Proceedings of the 2002 conference of the Centre for Advanced Studies on Collaborative research (CASCON ‘02), 3

37. H. Svensson and M. Host, "Introducing an Agile Process in a Software Maintenance and Evolution Organization", Software Maintenance and Reengineering, 2005. CSMR 2005. Ninth European Conference on, 2005, pp. 256-264.

38. Lars-Ola Damm, Lars Lundberg, David Olsson, Introducing Test Automation and Test-Driven Development: An Experience Report, Electronic Notes in Theoretical Computer Science, Volume 116, 19 January 2005, pp 3-15

39. T. L. Graves, M. J. Harrold, J. Kim, A. Porters and G. Rothermel, "An empirical study of regression test selection techniques", Software Engineering, 1998. Proceedings of the 1998 International Conference on, Kyoto, 1998, pp. 188-197.

40. P. Runeson, E. Engström, “Regression Testing in Software Product Line Engineering”, Advances in Computers, Volume 86, 2012, Pages 223-263

41. E. D. Ekelund and E. Engström, "Efficient regression testing based on test history: An industrial evaluation", Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on, Bremen, 2015, pp. 449-457

42. Jung-Min Kim and A. Porter, "A history-based test prioritization technique for regression testing in resource constrained environments", Software Engineering, 2002. ICSE 2002. Proceedings of the 24rd International Conference on, Orlando, FL, USA, 2002, pp. 119-129

43. Di Nardo, D., Alshahwan, N., Briand, L., *and* Labiche, Y. *(*2015*),* Coverage-based regression test case selection, minimization and prioritization: a case study on an industrial system. Softw. Test. Verif. Reliab., 25, 371–396

44. Yanping Chen, Robert L. Probert, D. Paul Sim, “Specification-based Regression Test Selection with Risk Analysis”, CASCON '02 Proceedings of the 2002 conference of the Centre for Advanced Studies on Collaborative research, 2012

45. Wasiur Rhmann, Taskeen Zaidi, Vipin Saxena, “Test Cases Minimization and Prioritization Based on Requirement, Coverage, Risk Factor and Execution Time”, British Journal of Mathematics & Computer Science 01/2016; 14(1):1-9

46. L. Naslavsky, H. Ziv and D. J. Richardson, "A model-based regression test selection technique", Software Maintenance, 2009. ICSM 2009. IEEE International Conference on, Edmonton, AB, 2009, pp. 515-518

47. TMMi Foundation. [WWW] http://www.tmmi.org/ (09.04.2016)

48. Capability Maturity Model Integration. [WWW] https://en.wikipedia.org/wiki/Capability_Maturity_Model_Integration (09.04.2016)

49. Anti-Regression Approaches: Impact Analysis and Regression Testing Compared and Combined: Part III: Regression Testing. [WWW] http://gerrardconsulting.com/?q=node/553 (17.04.2016)

50. Continuous integration. [WWW] https://en.wikipedia.org/wiki/Continuous_integration (19.04.2016)