UDC 004.413.4, DOI:10.2298/CSIS090923008L

Software Testing Optimization by Advanced Quantitative Defect Management

Ljubomir Lazić

State University of Novi Pazar, Vuka Karadzica bb,
36 300 Novi Pazar, Serbia
llazic@np.ac.rs

Abstract. Software testing provides a means to reduce errors, cut maintenance and overall software costs. Numerous software development and testing methodologies, tools, and techniques have emerged over the last few decades promising to enhance software quality. While it can be argued that there has been some improvement it is apparent that many of the techniques and tools are isolated to a specific lifecycle phase or functional area. This paper presents a set of best practice models and techniques integrated in optimized and quantitatively managed software testing process (OptimalSQM), expanding testing throughout the SDLC. Further, we explained how can Quantitative Defect Management Model be enhanced to be practically useful for determining which activities need to be addressed to improve the degree of early and cost-effective software fault detection with assured confidence is proposed. To enable software designers to achieve a higher quality for their design, a better insight into quality predictions for their design choices, test plans improvement using Simulated Defect Removal Cost Savings model is offered in this paper.

Keywords: Software Testing; Defect Management; Optimization.

1. Introduction

This paper presents some research results of ongoing project [5-7][1], designed to study software defect data as a means toward identifying where resources should be allocated most effectively to provide the highest quality of software product while reducing the overall cost of software testing. The software development industry spends more than half of its budget on maintenance related activities. Software testing provides a means to reduce errors, cut maintenance and overall software costs. The importance of software testing has been emphasized more and more, as the quality of software affects its

[1] This work has been done within the project 'Integrated and Optimized Software Testing and Maintenance Process', supported in part by the Ministry of Science and Technological Development of the Republic of Serbia under Project No. TR-13018.

benefit to companies significantly [1-4]. The identification and removal of software defects constitutes the basis of the software testing process a fact which inevitably places increased emphasis on defect related software measurements. Early in the history of software development, testing was confined to testing the finished code, but, testing is more of a quality control mechanism. Avoidable rework consumes a large part of development projects, i.e. 20-80 percent depending on the maturity of the organization and the complexity of the products [9]. High amounts of avoidable rework commonly occur when having many faults left to correct in late stages of a project. In fact, research studies indicate that the cost of rework could be decreased by up to 50 percent by finding more faults earlier [2, 5, 9]. Numerous software development and testing methodologies, tools, and techniques have emerged over the last few decades promising to enhance software quality. While it can be argued that there has been some improvement it is apparent that many of the techniques and tools are isolated to a specific lifecycle phase or functional area.

This paper focuses on software testing and the measurements which allow for the quantitative evaluation of this critical software development process. The software testing process requires practical measurements for the quantification of all software testing phases. Software product quality and software testing process (STP) improvement commence with addressing the testing process in a quantitative manner [7]. The continuous monitoring of the testing process allows for establishing an adequate level of confidence for the release of software products and for the quantification of software risks, elements which traditionally have plagued the software industry. The mechanism for this study is development of a series of simulation models to improve STP [7,8].

In this paper, Quality and Efficiency in Software Testing by our OptimalSQM framework is described and its components defined and exemplified. It also discusses practical applications of OptimalSQM and research model for investigating its antecedents and impacts is presented. OptimalSQM provide alignment between testing and development which has been raised as an issue for successful systems development. Missing however have been actionable how to methodologies for assessing and enhancing such alignment [12,13,16]. This paper attempts to fill this gap by describing a systematic methodology, a development-testing alignment (DTA) methodology which posits that such alignment leads to beneficial effects such as lower costs and shorter time of development, greater system quality, fewer errors and a better relationship between the corporate IT units.

Systematic mechanisms for tying testing strategy and capabilities to development strategies and capabilities are also discussed. This paper presents a set of best practice models and techniques integrated in optimized and quantitatively managed software testing process, expanding testing throughout the SDLC. It includes best practice from Design of Experiments, Modeling & Simulation, integrated practical software measurement, Six Sigma strategy, Earned (Economic) Value Management (EVM) and Risk Management (RM) methodology through simulation-based software testing scenarios at various abstraction levels of the SUT to manage stable (predictable and controllable) software testing process at lowest risk, at an affordable price and time.

To enable software designers to achieve a higher quality for their design, a better insight into quality predictions for their design choices, test plans improvement using Simulated Defect Removal Cost Savings model is offered in this paper. Much rather we aim to define a simulation method with which it is possible to assist the test manager in evaluating test plan alternatives and adjusting test process improvement decisions in a systematic manner.

2. Need for Research

Cost to an organization (both in dollars and in image) is significant when software defects are identified after installation at a client site. This project intends to identify areas where improvements in software testing resource allocations would provide added value to the organization.

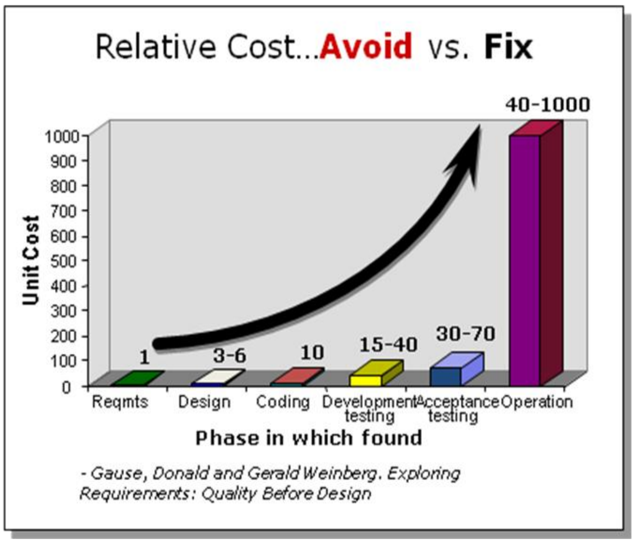


Fig. 1. Average Cost Of Defect Removal [10]

This paper proposes a development-testing alignment (DTA) methodology which posits that such alignment leads to beneficial effects such as lower costs and shorter time of development, greater system quality, fewer errors and a better relationship between the corporate IT units and customers in business functions who have commissioned new systems (see Fig. 2). Alignment models and measurements have been studied in other related contexts [16] but never within corporate IT units and specifically between the development and testing functions. The paper therefore decomposes DT alignment into a series of aspects for the purpose of assessing and analyzing each of the construct. These aspects are drawn from the overarching framework developed initially from prior literature [8,16]. The DTA methodology will allow IT managers to improve the effectiveness of testing and development by both synergistically integrating testing in the development process and by aligning the testing and development units in terms of strategy and execution capability.

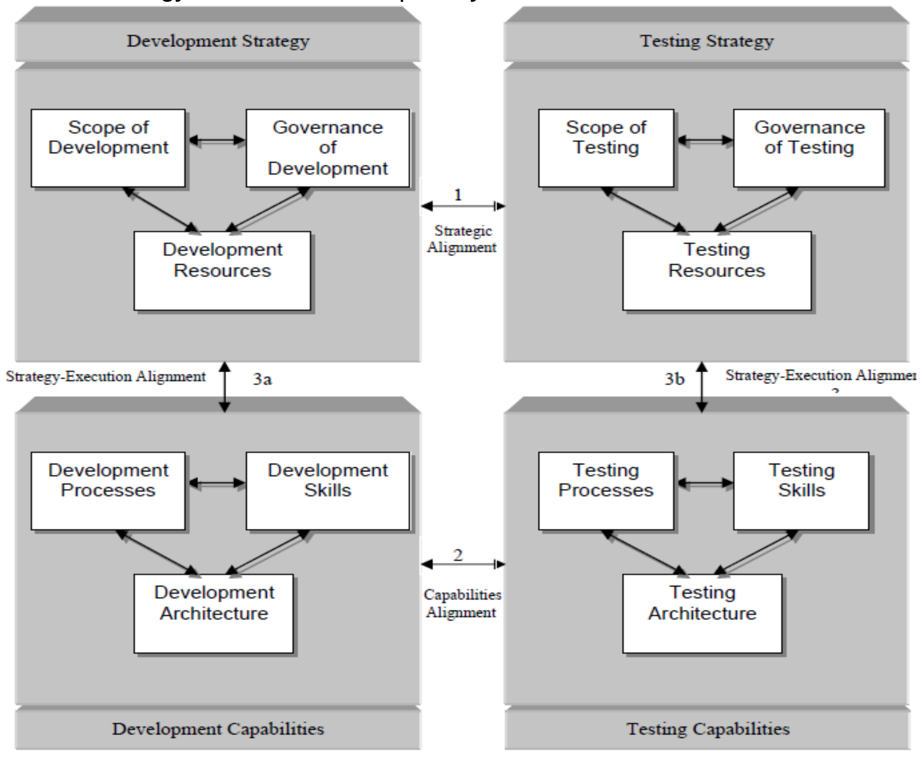


Fig. 2. Alignment model for testing and development (adapted from [13])

Organizations that engage in software development and testing benefit significantly if their management team has tools to assist them in determining the most effective use of financial resources that might result in the fewest software errors in delivered systems [2-10,20-24]. To be most effective, this tool needs to be developed after a thorough review of the specific organization's testing data [17,24]. Once developed, the tool will identify the specific phases and processes during the development life cycle where additional resources would provide the best return on investment and highest software quality. The use of this tool will provide a major reduction in the number and severity of software defects that exist after software testing. It will also reduce the overall cost of software testing by focusing on the appropriate process for a specific organizational environment [7,9,17-19,24].

To summarize, the purpose of this research is to increase software quality and reduce overall costs of software testing by focusing resources where they provide the most value. According to Gartner [14], on average, 7% of software functionality that was paid for is actually used, while 85% of IT projects failing to meet objectives (32% being cancelled outright). Dhaliwal and Onita [13] posit that many of these development failures are a result of poorly executed development process. These employ either inadequate development models or flawed implementation due, in part, to the lack of proper testing and effective collaborative mechanisms between testers and developers. These issues have not yet found a proper solution, due, in part, to a lack of a methodology that would allow the analysis and correction of software development processes. A review of the testing and development literature reveals that relations between the development and testing functions are lacking for projects of medium and large magnitude, where testing is separate from the development activities [12,15].

2.1. Research Questions

Based on the outcome of the evaluation of related work conducted in the previous subsection, the our project has identified some challenges to address. The challenges can be broken down into five sub-questions to address in this paper. The initial main research question that was posed for the complete research in this project was:

RQ1 or Main Research Question: How can software testing be performed efficiently and effectively i.e. Optimal, that is, do we have a framework model targeted specific software testing domains or problem classes described below in RQ2 to RQ5?

To be able to address the main research question several other research questions needed to be answered first (RQ2–RQ5). The first question that needed an answer, after the main research question was formulated, was:

RQ2: Which metric or set of metrics can assess effectiveness of test detecting techniques and what is the potential in combining different software testing techniques with respect to effectiveness (and to some extent efficiency)?

The answer to this research question is to be found in Section 3 and 4 together with an analysis of how software testing is used in different types of projects. To begin with, the research aimed at exploring the factor of defect detection and removing effectiveness (DRE) during SDLC while later focusing on early aspects of software cost of quality. In order to examine if the current practice in software development projects was satisfactory for developing software with sufficient quality and budget constraint, RQ3 evolved into:

RQ3: Which metric or set of metrics can identify and prioritize software quality attributes, can assess cost of software quality management process in a specific project i.e. how to optimize software quality to pay off investment in STP improvement (ROI)?

To enable software designers to achieve a higher quality for their design, a better insight into quality predictions for their design choices that evolved to the RQ4:

RQ4: Which metric or set of metrics can identify and prioritize improvements to achieve early and cost-effective software fault detection, can assess the improvements potential of improving the degree of early and cost-effective software fault detection?

A systematic model which enables to minimize the cost of switching between test plan alternatives, when the current choice cannot fulfill the quality constraints, is proposed in this paper as one answer to the RQ4. This is main concern and contribution of this paper which is described in Section 5. New kinds of STP improvement is introduced and hence indirectly led to Research Question 5:

RQ5: How should a software development organization apply the metric(s) suggested above for assessing ongoing and finished projects with an Dynamic Control Model view i.e. What are optimality and stability criteria of very complex STP dynamics problem control?

Section 5 explain how can Quantitative Defect Management (QDM) Model be enhanced (as one solution to RQ4) to be practically useful for determining which activities need to be addressed to improve the degree of early and cost-effective software fault detection with assured confidence, optimality and stability criteria of very complex STP dynamics problem control which is partially solved in our paper [19].

To be able to implement all proposed solutions, one must choose a research methodology. Iterative approaches for improvement exist in the quality management area. The PDCA (plan-do-check-act) or "Shewhart Cycle", the WV (or zigzag) framework and the DMAIC (define-measure-analyze-improve-control) cycles are analogous methods to capture a generic framework for the improvement of a process or system [1,3,8]. A similar model, the "simulate-test-evaluate process" iterative experimentation cycle was developed by the office of the US Secretary of Defense, called the Simulation, Test and Evaluation Process (DoD STEP framework) to integrate M&S into the test and evaluation process of the system/software under test (SUT) [17]. Long design iteration loops with late feedback drive cost and schedule overruns in SDP-STP requires further research of this stability criteria of very complex STP dynamics problem control.

3. Integrated, quantitatively managed and optimized software testing process - OptimalSQM solution

When design and testing activities are not coupled, the information testing provides on product design is delivered at a wrong point in the process. This late information is either not useful any more or shows design problems too late, causing undesired late rework. Thus, iteration cycles should be kept short and rapid.

To address the research questions stated above, multiple studies have been conducted [5-8] about alignment between the development and testing functions which can be defined as the strategic and operational fit between the development and testing functions on components of strategy and capabilities [13-16]. Since systems development as well as systems testing is integral parts of the corporate technology acquisition strategy, they too have to be aligned to ensure business success. In many organizations, there is a gap, or misalignment, at the strategic and/or execution level, between the development and testing groups as well as between individual testers and developers. To correct these misalignments, this paper proposes a methodology, grouped under the DTA model [13] that draws upon the strategic alignment model initially proposed in [16]. This DTA model focuses on the fit between the development and testing functions. A high level of integration of business and IT plans facilitates communication and collaboration [16]. Integration represents the level of linkage between development and testing, while correspondence represents how closely their capabilities mirror and complement each other. Varying levels of alignment can either promote or hinder integration and correspondence. This is a common characteristic of all alignment models in the literature as verified by Dhaliwal, J. and Onita C. in their work [13]. Figure 2 details the key structural and flow components of the DT alignment model for development and testing within the corporate IT unit. This model decomposes the alignment of the development and testing functions along three key flow dimensions: 1) strategic alignment, 2) capabilities alignment, and 3) strategy-execution alignment.

Based on our study in [8], a methodology for achieving DT Alignment through Collaborative Techniques & Technology, enables OptimalSQM to be realised. The methodology is derived from a survey of the literature from Strategic Alignment [13-16] Testing [1-4], [8-10] to Project Management and Information Systems development methods [10-16]. To improve the reliability and validity of this methodology, alignment case studies and field studies were conducted and real life examples are given to improve the applicability of the methodology. A list of techniques is also mapped onto each step of the methodology.

3.1. Integrated and Optimized Software Testing Process (IOSTP) framework - OptimalSQM solution

To answer the main research question (**RQ1**) we applied DTA model, described above, in OptimalSQM framework which combine best practice from Design of Experiments, Modeling & Simulation, integrated practical software measurement, Six Sigma strategy, Earned (Economic) Value Management (EVM) and Risk Management (RM) methodology through simulation-based software testing scenarios at various abstraction levels of the software under test (SUT) to manage stable (predictable and controllable) software testing process at lowest risk, at an affordable price and time [8,9], [17,18] as depicted in Fig. 3. Unlike conventional approaches to software testing (e.g. structural and functional testing) which are applied to the software under test without an explicit optimization goal, the IOSTP with embedded Risk Based Optimized STP (RBOSTP) approach designs an optimal testing strategy to achieve an explicit optimization goal, given a priori [8,17]. This leads to an adaptive software testing strategy. A non-adaptive software testing strategy specifies what test suite or what next test case should be generated, e.g. random testing methods, whereas an adaptive software testing strategy specifies what testing policy should be employed next and thus, in turn, what test suite or test case should be generated next in accordance with the new testing policy to maximize test activity efficacy and efficiency subject to time-schedule and budget constraints.

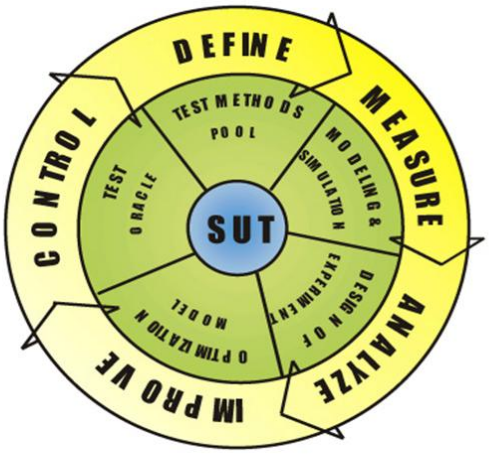


Fig. 3. Integrated and optimized software testing process (IOSTP) framework, core of OptimalSQM framework [17]

The use of state-of-the-art methods and tools for planning, information, management, design, cost trade-off analysis, and modeling and simulation, Six Sigma strategy significantly improves STP effectiveness as in Fig. 3 which graphically illustrates a generic IOSTP framework that makes core of the OptimalSQM framework [17].

The main components of IOSTP with embedded RBOSTP approach to STP:

- Integrate testing into the entire development process
- Implement test planning early in the life cycle via Simulation based assessment of test scenarios
- Automate testing, where practical to increase testing efficiency
- Measure and manage testing process to maximize risk reduction
- Exploit Design of Experiments techniques (optimized design plans, Orthogonal Arrays etc.)
- Apply Modeling and Simulation combined with Prototyping
- Continually improve testing process by pro-active, preventive (failure mode analysis) Six Sigma DMAIC model
- Continually monitor Cost-Performance Trade-Offs (Risk-based Optimization model, Economic Value and ROI driven STP).

In order to significantly improve software testing efficiency and effectiveness for the detection and removal of requirements and design defects in our framework of IOSTP, during 3 years of the IOSTP framework deployment to STP of embedded-software critical system such as Automated Target Tracking Radar System (ATTRS) [17], we calculated overall value returned on each dollar invested i.e. ROI of 100:1 .

4. Optimum DDTs combination selection and optimization study in OptimalSQM

The research question - **RQ2** divided the research, as presented in this paper, into two areas covering effectiveness in software testing techniques (defect detection techniques – DDT) and efficiency in software testing with development-testing alignment (DTA) methodology [5-9]. Such alignment leads to beneficial effects such as lower costs and shorter time of development, greater system quality, fewer errors and a better relationship between the corporate IT unit and customers in business functions who have commissioned new systems. To begin with **RQ2**, the research aimed at exploring the factor of defect detection and removing effectiveness (DRE) during SDLC is answered in our work [6]. Here is brief description of main ideas of **RQ2** answer.

4.1. Statement Of the Problem - Defect removal effectiveness

A key metric for measuring and benchmarking the software testing efficacy is by measuring the percentage of possible defects removed from the product at any point in time. Both a project and process metric – can measure effectiveness of quality activities or the quality of a all over project by:

$$DRE = E/(E+D) \qquad \textbf{(1)}$$

Where *E* is the number of errors found before delivery to the end user, and *D* is the number of errors found after delivery. The goal is to have *DRE* close to 100%. The same approach is applied to every test phase denoted wit *i* as shown on Fig. 4:

$$DRE_i = E_i / (E_i + E_{i+1}) \qquad \textbf{(2)}$$

Where E_i is the number of errors found in a software engineering activity *i*, and E_{i+1} is the number of errors that were traceable to errors that were not discovered in software engineering activity *i*. The goal is to have this DRE_i approach to 100% as well i.e., errors are filtered out before they reach the next activity. Projects that use the same team and the same development processes can reasonably expect that the *DRE* from one project to the next are similar. For example, if on the previous project, you removed 80% of the possible requirements defects using inspections, then you can expect to remove ~80% on the next project. Or if you know that your historical data shows that you typically remove 90% before shipment, and for this project, you've used the same process, met the same kind of release criteria, and have found 400 defects so far, then there probably are ~50 defects that you will find after you release. How to combine Defect Detection Technique to achieve high *DRE*, let say >85%, as a threshold for IOSTP required effectiveness [2-5], is explained in this section, which describe optimum combination of software defect detection techniques choices.

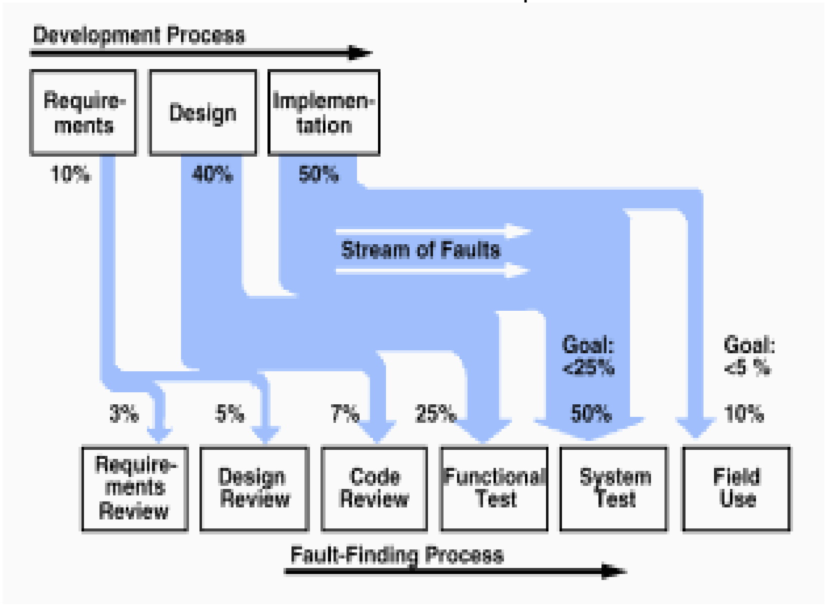


Fig. 4. Fault Injection and Fixing Model

Note that the defects discussed in this section include all severity levels, ranging from severity 1: *activity stoppers*, down to severity 4. Obviously, it is important to measure defect severity levels as well as recording numbers of defects.

4.2. The optimum combination of software defect detection techniques choices determination

Planning, managing, executing, and documenting testing as a key process activity during all stages of development is an incredibly difficult process. There is strong demand for software testing effectiveness and efficiency increases. Software/System testing effectiveness is mainly measured by percentage of defect detection and defect leakage (containment), i.e. late defect discovery. Software testing efficiency is mainly measured by dollars spent per defect found and hours spent per defect found. The first step of test strategy definition is to decide what to test in the SDP process as described in [3,20,23]. The requirements verification matrix method supports this goal by cross-referencing each product requirement with suitable verification methods (inspection, analysis, demonstration, or test), verification classes (design proof, first article, or production), and special verification procedures (e.g., Failure Mode and Effect Analysis, Design of Experiments, Finite Element Method, etc.). Product requirements with high priority are the critical requirements, where test planning has to concentrate its resources.

The process is a building block approach designed to build upon the strengths and minimize the weakness of each testing technique and available resources. Main task was to develop a versatile optimization model [8] for assessing the cost, duration and effectiveness of alternative test scenario through feasible series of experiments: software test method, field test, through simulation, or through a combination, which represent sequence of test events.

Such scenarios are invaluable for determining where testing resources should be spent at the beginning of software development project. With an optimized testing solution, you can create what-if scenarios to help users understand the impact of changing risks, cycle attributes and requirements as priorities change. This insight proves invaluable when a testing organization is trying to determine the best way to balance quality with cost and schedule. By understanding the impact of different factors on testing, IT managers can identify the right balance.

We applied the End-to-End (E2E) Test strategy in our IOSTP framework [6,17]. End-to-End Architecture Testing is essentially a "gray box" approach to testing - a combination of the strengths of white box and black box testing. In determining the best source of data to support analyses, IOSTP with embedded RBOSTP considers credibility and cost of each test scenario i.e. concept. Resources for simulations and software test events are weighed against desired confidence levels and the limitations of both the resources and the analysis methods.

Our study [6] focuses on rapid multidisciplinary analysis and evaluation-on-a-DRE maximum-basis for DDT combination choices selection for each test phase activities in an traditional SDP i.e. P1- software requirement, P2- High level design, P3- Low Level Design, P4- code under test, P5- integration test, P6- system under test and finally P7- Acceptance test. Recall section 4, in our work [6], for Different Defect Detection Strategy and Techniques options, together with critical STP variables performance characteristics (e.g. DRE, cost, duration), in which we are studied to optimize design, development, test and evaluation cost using orthogonal arrays for computer experiments. The optimum combination of software defect detection techniques choices were determined applying orthogonal arrays constructed for post mortem designed experiment with collected defect data of a real project. First, we applied adapted Borda voting method, on similar way, to rank all used Defect Detection Techniques through software development life cycle from most-to-least performance and quality characteristics of DDT in revealing software faults (bugs, errors). In this way we reduced huge possible number of DDTs, in particular, the DDT with the highest Borda Count is the best DDT according to testers Performance and Quality multi-criteria assessment [6], the DDT with the second highest count is the next DDT with highest score, and so forth to only three most ranked DDT. According to testers assessment of 5 most frequently used DDT in IOSTP [6,]: DDT1= Inspection – DBR, DDT2= PBR, DDT3= CEG+BOR+MI, DDT4= M&S, DDT5= Hybrid (Category Partition, Boundary value analysis, Path testing etc.) three of DDTs have the highest rank 0 i.e. DDT1=DDT2=DDT4=0, then DDT3= CEG+BOR+MI is next ranked and the last was DDT5. Because of that we will group those three DDT with highest rank 0, call them Static Test Techniques – TT1 and treat all three DDTs as one factor in optimization experiment applying Orthogonal Arrays as Optimization Strategy. Next high Borda ranked DDT4= CEG+BOR+MI we designate with TT2 and the last ranked DDT5 as TT3.

In this study, design of maximum DRE percentage of STP optimization problem solving with best DDT choice combination in each phase P1 to P7 as controlled variables values is determined by designed experiment plan using orthogonal arrays designed for this computer experiment. Seven major test phases - P1 to P7, for accounting maximum DRE percentage all over STP fault injection and removal model (see Fig. 4, 7 and 8 below) for DDT candidate selection in each test phase were determined. These were the Static Test Techniques – TT1 (consisting of three DDTs as one factor in optimization experiment applying Orthogonal Arrays as Optimization Strategy), the TT2 i.e. DDT4= CEG+BOR+MI and TT3 – Hybrid Detection Technique= DDT5 (consisting of Category Partition, Boundary value analysis, Path testing etc.). The objective of this investigation was then to determine the best combination of Test Techniques (*TTi* , *i*=1,2 and 3) options for the seven major test phase activities sections optimized for maximum DRE percentage under cost and time constraints [6,pages 1333-1335].

As the next step, least squares regression analysis is used to fit the second order approximation model given by equation (3) to the DRE data in terms of the seven design variables P_i , i=1 to 7. This parametric model accounts for the response surface curvature (square terms) and two factor interactions (cross terms):

$$\text{DRE (\%)} = 111.71 - 2.58 *P1 + 1.22*P2 -1.95*P3 - 7.61*P4 - 0.69*P5 + 0.94*P6 -13.04*P7 - 0.36*P2^2 + 1.46*P4^2 + 0.79*P5^2 - 0.36P6^2 + 3.15*P7^2 \quad (\mathbf{3})$$

Note that, in this response surface approximation model, the parameter values for P_i design variables are restricted to 1 (TT1), or 2 (TT2), or 3 (TT3). In Table 1, a Maximum DRE (%) value and corresponding Test Techniques choices (TT1,TT2 and TT2) per test phase solution is given.

Table 1. Maximum DRE (%) value and corresponding Test Techniques choices per test phase solution [6, page 1335]

P1	P2	P3	P4	P5	P6	P7	DRE [%]
TT1	TT2	TT1	TT1	TT3	TT2	TT2	94.3

At these levels, the IOSTP DRE was predicted to be 94.03 % using a second order prediction model (3). As a next step, a verification analysis was performed. The DRE (%) of an IOSTP calculated from these test techniques choices, according to the post-mortem real project data using optimized DDT choices from Table 1, we computed DRE (%) to be 93.43 % . Difference is 0.6%=94.03%-93.43% that is acceptable to validate our prediction model for DRE (%) in equation (3) for optimal DDT combination choice given in Table 1.

Optimal combination of DDT choices per phase P given in Table 1 made increase of about 6 %, compared to un-optimized DDTs combination per each test phase we used in our real project in which we achieved DRE of 87.43 % in our case study.

5. Advanced Quantitative Defect Management (AQDM) Model

The investment in software quality, particularly in software testing, like any investment has an immediate cost, with an expected net payback. There is where Quality Cost Analysis could be used as effective tool to make them understand the ROI. In our paper [19], we defined techniques to analyze and interpret return on the testing investment (ROTI) values: Financial ROI and Schedule Benefits as one possible answer to **RQ3** based on our studies [5,18,19] i.e. which metric or set of metrics can identify and prioritize software quality attributes, can assess cost of software quality management process in a specific project i.e. how to optimize software quality?

In our work [19] we proposed a model that traces design decisions and the possible alternatives. With this model it is possible to minimize the cost of switching between design alternatives, when the current choice cannot fulfill the quality constraints.

5.1. Faults-Slip-Through (FST) Model

The partial answer to **RQ5** can be found in our work [19], too. In this section we explain how can Quantitative Defect Management (QDM) Model be enhanced (as answer to **RQ4**) to be practically useful for determining which activities need to be addressed to improve the degree of early and cost-effective software fault detection with assured confidence.

The main objective of the case study presented in this section was to investigate how fault statistics could be used for removing unnecessary rework in the software development process. This was achieved through a measure called Faults-Slip-Through (FST) [9, Section 2], i.e. the measure tells which faults that would have been more cost-effective to find in earlier phases.

5.2. The defect containment measure

An error in an activity of development phase P_i (i=1 to N) is made that causes a failure (see Fig. 10-13). The failure leads to a reported anomaly. When the reported anomaly is analyzed, the fault(s) causing the failure is found and corrected. *Rework* is about revising an existing piece of software or related artifact. Therefore, a typical rework activity is to correct reported anomalies. Rework can be divided into two primary types of corrective work [9]:

- *Avoidable rework* is work that would not have been needed if the previous work would have been correct, complete, and consistent. Such rework consists of the effort spent on detecting and fixing software difficulties that could have been discovered earlier or avoided altogether [2,5].
- *Unavoidable rewor*k is work that could not have been avoided because the developers were not aware of or could not foresee the change when developing the software, e.g. changed user requirements or environmental constraints [9].

This section describes the selected method for how to achieve the objectives stated in the previous section. The method can be divided into the following three steps:

1. Determine which faults that should have been avoided or at least found earlier,
2. Determine the average cost of finding faults in different phases,

3. Determine the improvement potential from the results in (1) and (2).

The three sub-sections below describe how to perform each of the three steps.

5.3. The raw defect containment data

This section is dedicated to a model for assessing a plan for SQA defect-removal effectiveness and cost. The model, a multiple filtering model as shown on Fig. 4, is based on data acquired from a survey of defect origins, percentages of defect removal achieved by various quality assurance activities, and the defect-removal costs incurred at the various development phases. The model enables quantitative comparison of quality assurance policies as realized in quality assurance plans. The application of the proposed model is based on three types of data, described under the following headings from [1, pages 135-142].

5.4. Defects removal improvement potential

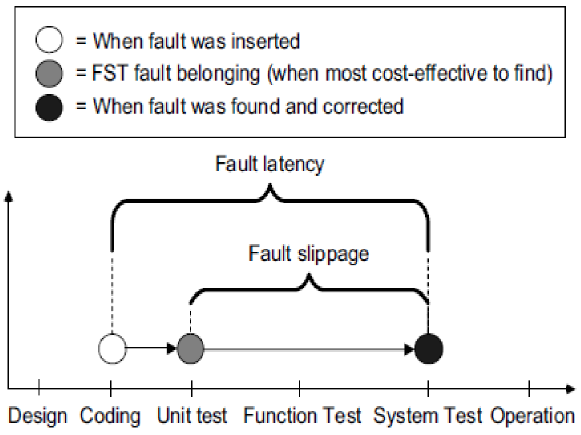

○ = When fault was inserted
● = FST fault belonging (when most cost-effective to find)
● = When fault was found and corrected
Fault latency
Fault slippage
Design
Coding
Unit test
Function Test
System Test
Operation

Fig. 5. Example of Fault Latency and FST

As previously mentioned, FST measurement was used for determining this, i.e. it evaluates whether each fault slipped through the phase where it should have been found or not. The main difference between FST measurement and other related measurements is when a fault is introduced in a certain phase but it is not efficient to find in the same phase. For example, a certain test technique might be required to simulate the behaviour of the function. Then it is not a fault slippage. Figure 5, further, illustrates this difference. A consequence of how FST is measured, a definition must be created to support the measurement, i.e. a definition that specifies which faults that should be found in which phase. To be able to specify this, the organization must first determine what should be tested in which phase. Therefore, this can be seen as test strategy work. Thus, experienced developers, testers and managers should be involved in the creation of the definition. The results of the case study in Section 5.5 further exemplify how to create such a definition.

When having all the faults categorized, the next step is to estimate the cost of finding faults in different phases. From the measure, the improvement potential of different parts of the development process is estimated by calculating the cost of the faults that slipped through the phase where they should have been found (see Fig. 12 and 13 below). The usefulness of the method was demonstrated by applying it on two completed development projects [1] and [2]. The results determined that the implementation phase had the largest improvement potential since it caused the largest FST cost to later phases, i.e. from 56 to 87 percent of the total improvement potential in the two studied project scenarios. It is assumed that the filtering effectiveness of accumulated defects of each quality assurance activity is not less than 40% (i.e., an activity removes at least 40% of the incoming defects). Typical average defect filtering effectiveness rates for the various quality assurance activities, by development phase, based on Galin, D. [1] and 11. Boehm, B. et al [11], are listed in Table 2.

Data collected about development project costs show that the cost of removal of detected defects varies by development phase, while costs rise substantially as the development process proceeds. For example, removal of a design defect detected in the design phase may require an investment of 2.5 working days; removal of the same defect may require 40 working days during the acceptance tests. Estimates of effectiveness of software quality assurance tools and relative costs of defect removal are provided by McConnell [10]. Although defect removal data are quite rare, professionals agree that the proportional costs of defect removal have remained constant since the surveys conducted in the 1970s and 1980s. Instead of average per phase defect removal cost we propose average relative defect-removal costs injected in phase $P_{i\ (i=1\ \text{to}\ 7)}$ and detected and removed latter in downstream phases P_j , $j>i$ up to the operation phase ($j=7$) as shown in Table 3.

The **improvement potential** (**IP**) is determined by calculating the difference between the cost of faults in relation to what the fault cost would have been if none of them would have had slipped through the phase where they were supposed to be found. Figure 6. provides the elements of matrix, with corresponding formulas, for making such a calculation and the improvement potential can be calculated in a two-dimensional matrix. The formulae for calculating the improvement potential for each cell **IPir** is:

$$IPir = No\_\ faults\_in\_P(i,r) * AverCDR(r,i) - No\_\ faults\_in\_P(i,r) * AverCDR(i,i)$$

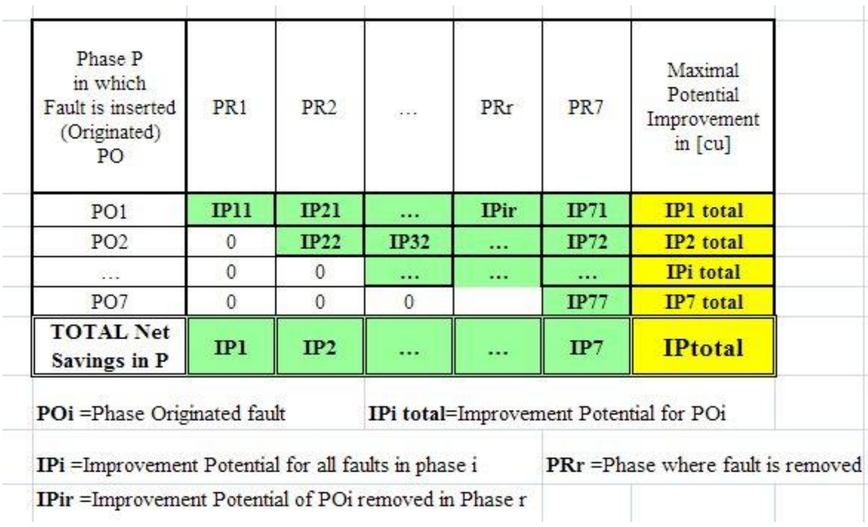
for $i=1$ to 7, and $r \geq i$ correspond to a cell in phase removed/originated matrix. *AverCDR(r,i)* is average cost of faults originated in *i* and removed in *r* as shown in Table 3. **IPi total** and **IPi** are calculated by summarizing the corresponding row/column. In order to demonstrate how to use and interpret the matrix, Figure 12 provides an example calculation by applying the previous formula on the values in the table in the Fig. 10. The most interesting cells are those in the rightmost column that summarizes the total cost of faults in relation to fault belonging and the bottom row that summarizes the total unnecessary cost of faults in relation to phase found. For example, the largest improvement potential is in the **LL Design** test phase, i.e. the phase triggered **30661 [cu]** of unnecessary costs in later phases due to a large FST from faults injected in Requirement phase. Therefore, the primary usage of the values is to serve as input to an expected ROI calculation when prioritizing possible improvement actions according to formula:

ROI= (CostToFixOld - CostToFixNew) /CostToFixOld=39.2%.

5.5. Qunatitative Defect Removal Model

The model is based on the following assumptions:

■ The development process is linear and sequential, following the waterfall model of CMM Level 5. Software size is aproximately 100FP (1 injected defect/FP) i.e. for Java implementation about 50KLOC of source code [4].



Phase P in which Fault is inserted (Originated) PO	PR1	PR2	…	PRr	PR7	Maximal Potential Improvement in [cu]
PO1	**IP11**	**IP21**	**…**	**IPir**	**IP71**	**IP1 total**
PO2	0	**IP22**	**IP32**	**…**	**IP72**	**IP2 total**
…	0	0	**…**	**…**	**…**	**IPi total**
PO7	0	0	0		**IP77**	**IP7 total**
TOTAL Net Savings in P	**IP1**	**IP2**	**…**	**…**	**IP7**	**IPtotal**

POi =Phase Originated fault **IPi total**=Improvement Potential for POi

IPi =Improvement Potential for all faults in phase i **PRr** =Phase where fault is removed

IPir =Improvement Potential of POi removed in Phase r

Fig. 6. Matrix Formula for Calculation of Improvement Potential

■ A number of “new” defects are introduced in each development phase. For their distributions, see Fig. 8 and 9.

■ Review and test software quality assurance activities serve as filters, removing a percentage of the entering defects and letting the rest pass tothe next development phase. For example, if the number of incoming defects is

30, and the filtering efficiency is 60%, then 18 defects will beremoved, while 12 defects will remain and pass to be detected by the next quality assurance activity. Typical filtering effectiveness rates for the Standard quality assurance activities are shown in Table 2.

Table 2. Average filtering (defect removal - DR) effectiveness by Standard quality assurance activities plan, adapted from [1, pages 136-138]

No.	Quality assurance activity	Defect removal effectiveness	Average Cost of removing a detected defect (cost units)
1	Requirement specification review	50%	1
2	Design review	50%	2.5
3	Unit test – code	50%	6.5
4	Integration test	50%	16
5	Documentation review	50%	16
6	System test	50%	40
7	Operation phase	100%	110

■ At each phase, the incoming defects are the sum of defects not removed by the former quality assurance activity together with the "new" defects introduced (created) in the current development phase.

■ The cost of defect removal is calculated for each quality assurance activity by multiplying the number of defects removed by the relative cost of removing a defect (see Table 3, 3rd column).

■ The remaining defects, unfortunately passed to the customer, will be detected by him or her. In these circumstances, full removal entails the heaviest of defect-removal costs. In this model, each of the quality assurance activities is represented by a filter unit, as shown for Design in Fig. 7. The model presents the following quantities:

■ POD = Phase Originated Defects (from Fig. 8)

■ PD = Passed Defects (from former phase or former quality assurance activity)

■ %FE = % of Filtering Effectiveness (also termed % screening effectiveness) (from Table 2)

■ RD = Removed Defects

■ CDR = Average Cost of Defect Removal (from Table 2)

■ TRC = Total Removal Cost: TRC = RD ×CDR.

The illustration in Fig. 8 of the model applies to a standard quality assurance plan ("standard defects filtering system") that is composed of six quality assurance activities (six filters), as shown in Table 2.

Table 3. Representative average relative defect-removal costs and fixing multiplier because FST

No.	Quality assurance activity	Average Cost of DR [cost units]	Fixing multi-plier (CM) P1→ P7	Fixing multi-plier (CM) P2→ P7	Fixing multi-plier (CM) P3→ P7	Fixing multi-plier (CM) P4→ P7	Fixing multi-plier (CM) P5→ P7	Fixing multi-plier (CM) P6→ P7
1	Requir. specification review	1	1					
2	Design review	2.5	5	1				
3	Unit test – code	6.5	10	2	1			
4	Integration test	16	50	10	5	1		
5	Documenta-tion review	16	130	26	13	3	1	
6	System test	40	368	64	37	7	3	1
7	Operation phase	110	400	75	40	20	15	10

A comprehensive quality assurance (QA) plan ("comprehensive defects filtering system") achieves the following:

(1) Adds two quality assurance activities, so that the two are performed in the design phase as well as in the coding phase;

(2) Improves the "filtering" effectiveness of other quality assurance activities. The comprehensive quality assurance plan can be characterized as shown in Table 4.

The main conclusions drawn from the comparison are:

(1) The standard plan successfully removes only 57.6% (28.8 defects out of 50) of the defects originated in the requirements and design phase, compared to 92.0% (46 defects out of 50) for the comprehensive plan, before coding begins.

(2) The comprehensive plan, as a whole, is much more economical than the standard plan as it saves 41% of total resources invested in defect removal, compared to the standard plan.

(3) Compared to the standard plan, the comprehensive plan makes a greater contribution to customer satisfaction by drastically reducing the rate of defects detected during regular operations (from 6.9 % to 3 %).

The comparison also supports the belief that additional investments in quality assurance activities yield substantial savings in defect removal costs. Alternative models dealing with the cumulative effects of several qualityassurance activities are discussed by [2,5,9] as described below. A process-oriented illustration of the comprehensive quality assurance plan and model of the process of removing 100 defects is provided in Fig. 9. A comparison of the outcomes of the standard software quality plan versus the comprehensive plan is revealing as shown in Table 5.

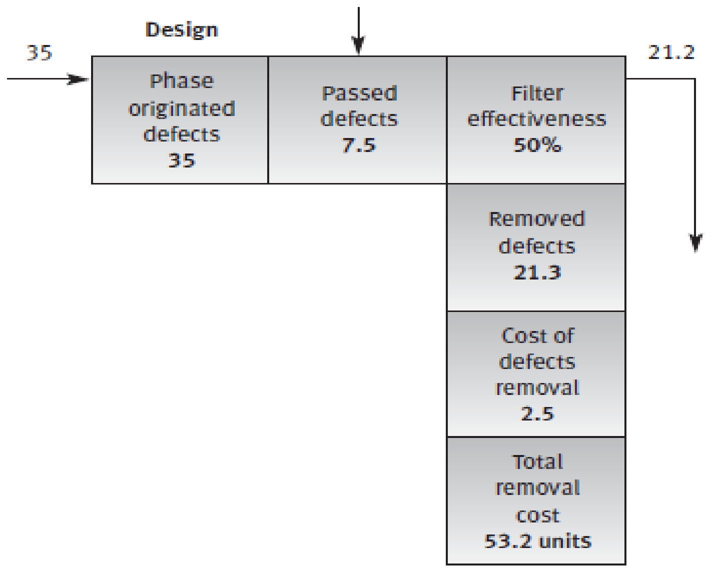

Design
35
Phase originated defects 35
Passed defects 7.5
Filter effectiveness 50%
21.2
Removed defects 21.3
Cost of defects removal 2.5
Total removal cost 53.2 units

Fig. 7. A filter unit for defect-removal effectiveness: example (100 defects) from [1]

Table 4. Average filtering (defect removal) effectiveness by Comprehensive quality assurance activities plan [1, page 140]

No.	Quality assurance activity	Defect removal effectiveness	Cost in [cu] of removing a detected defect
1	Requirement specification review	60%	1
2	Design inspection	70%	2.5
3	Design review	60%	2.5
4	Code inspection	70%	6.5
5	Unit test – code	40%	6.5
6	Integration test	60%	16
7	Documentation review	60%	16
8	System test	60%	40
9	Operation phase	100%	110

Table 5. Comparison of the standard and comprehensive QA plans [1, page 142]

No.	Quality assurance activity	Standad plan		Comprehensive plan	
		Percentage of removed defects	Cost of removing a detected defect (cost units)	Percentage of removed defects	Cost of removing a detected defect (cost units)
1	Requirement specification review	7.5%	7.5	9%	9
2	Design inspection	-	-	28.7%	71.8
3	Design review	21.3%	53.2	7.4%	18.5
4	Code inspection	-	-	24.4%	158.6
5	Unit test – code	25..6%	166.4	4.2%	27.3
6	Integration test	17.8%	284.8	9.8%	156.8
7	Documentation review	13.9%	222.4	9.9%	158.4
8	System test	7.0%	280	4%	160
	Total for internal QA activities	**93.1%**	**1014.3**	**97.4%**	**760.4**
	Defects detected during operation	6.9%	759	2.6%	286
	Total	**100.0%**	**1773.3**	**100.0%**	**1046.4**

5.6. Simulation results of AQDM improvement

Unlike conventional approaches to software testing which are applied to the software under test without an explicit optimization goal, as described above, the OptimalSQM approach designs an optimal testing strategy to achieve an explicit optimization goal, given a priori [5,6]. Improvement of original project data from [2] given in Fig. 10 (Note: original Defect Removal Efficiency [%], shown on Fig. 11 is less then Standard quality assurance activities plan, Scenario 1 in Table 2).

Also, comprehensive quality assurance plan (Scenario 2) which is realised through feasible series of experiments: software test method, field test, through simulation, or through a combination, represent new test sequence determined by Simulated Defect Removal Cost Savings model.

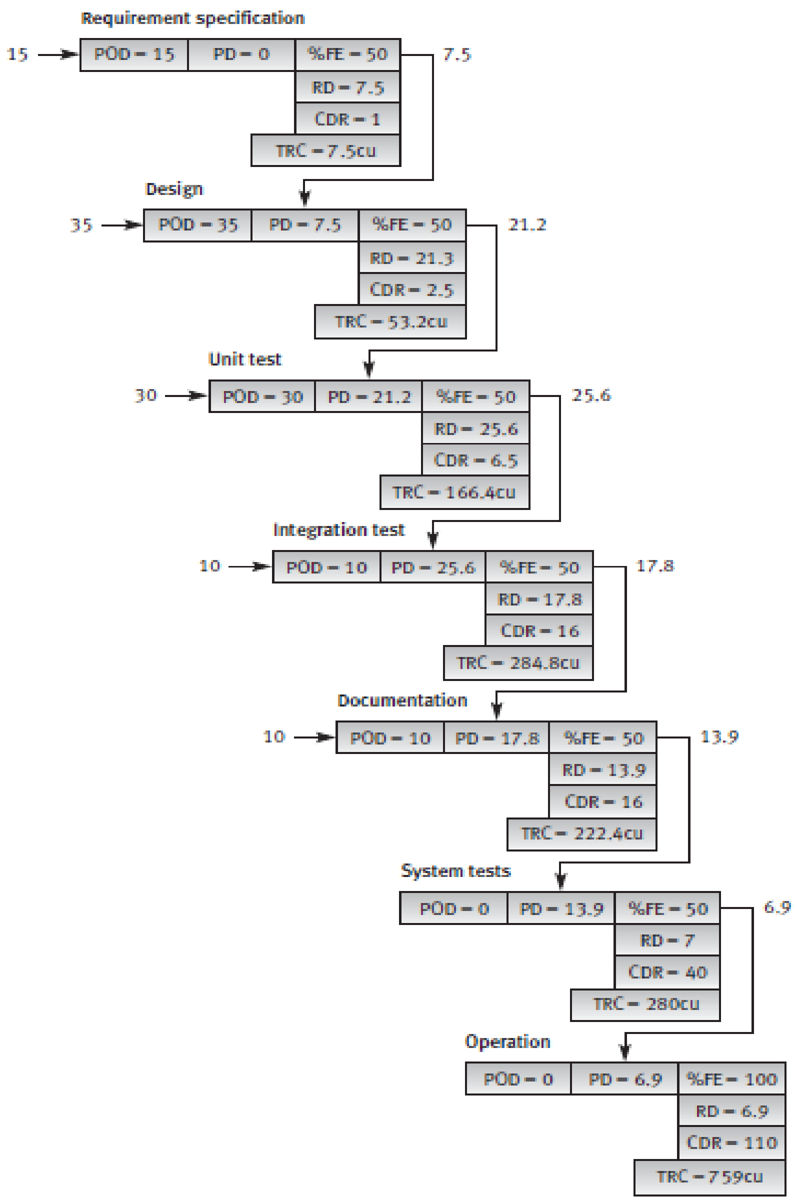
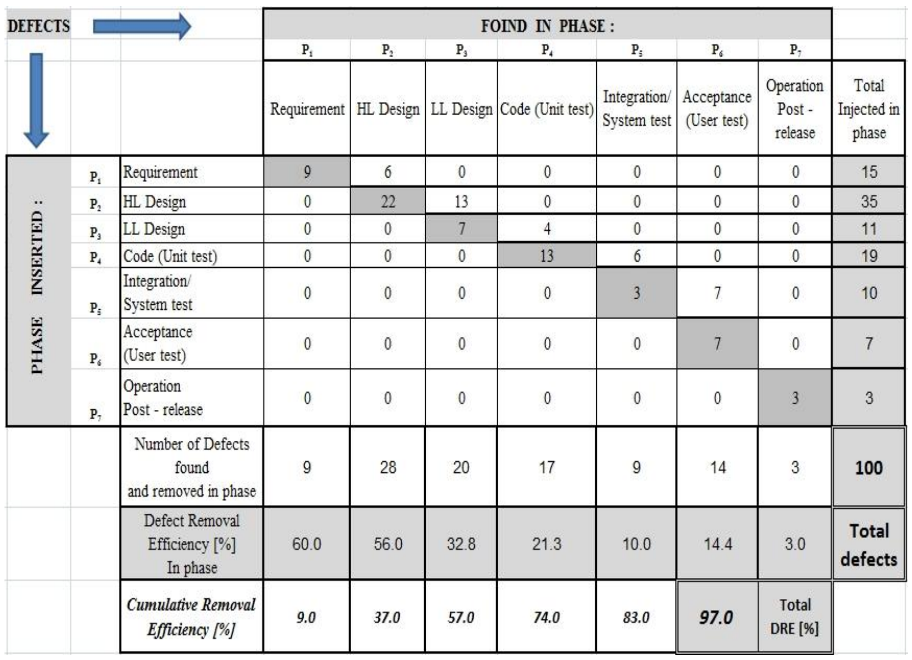


Fig. 8. DRE and costs of Standard QA plan and model of the process of removing 100 defects [1, page 139]



DEFECTS		FOIND IN PHASE :							
		P_1	P_2	P_3	P_4	P_5	P_6	P_7	
		Requirement	HL Design	LL Design	Code (Unit test)	Integration/ System test	Acceptance (User test)	Operation Post - release	Total Injected in phase
PHASE INSERTED : P_1	Requirement	9	6	0	0	0	0	0	15
P_2	HL Design	0	22	13	0	0	0	0	35
P_3	LL Design	0	0	7	4	0	0	0	11
P_4	Code (Unit test)	0	0	0	13	6	0	0	19
P_5	Integration/ System test	0	0	0	0	3	7	0	10
P_6	Acceptance (User test)	0	0	0	0	0	7	0	7
P_7	Operation Post - release	0	0	0	0	0	0	3	3
	Number of Defects found and removed in phase	9	28	20	17	9	14	3	**100**
	Defect Removal Efficiency [%] In phase	60.0	56.0	32.8	21.3	10.0	14.4	3.0	**Total defects**
	Cumulative Removal Efficiency [%]	*9.0*	*37.0*	*57.0*	*74.0*	*83.0*	*97.0*	Total DRE [%]	

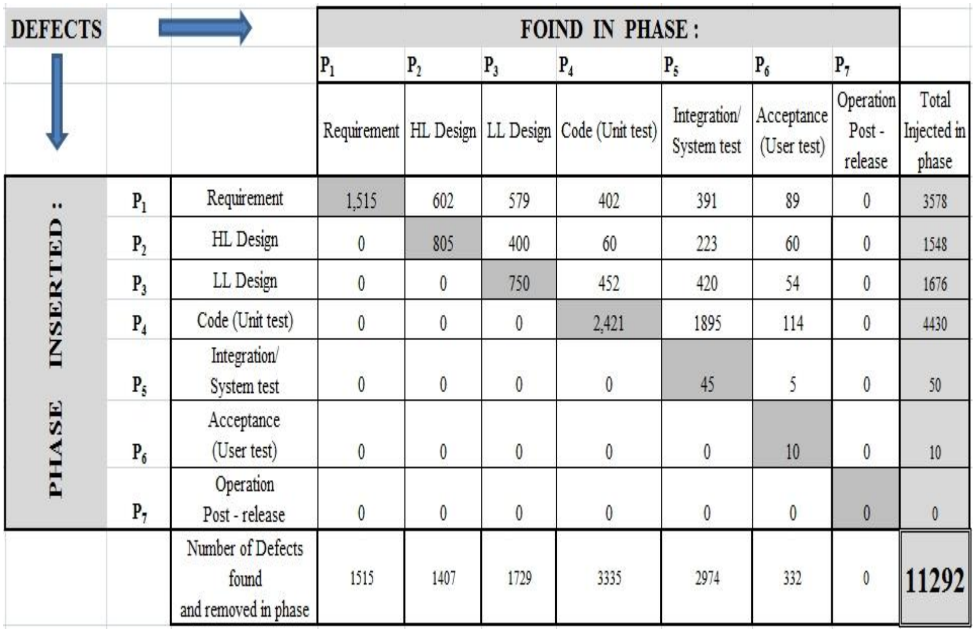
Fig. 9. DRE of Comprehensive QA plan and model of the process of removing 100 defects [1]

The Simulated Defect Removal Cost Savings model, uses net savings approach that is calculated using this formula:

$$NS = IP_{r \to r+1} = FST_{\mathrm{Pr} \to \mathrm{Pr}+1} * (CM_{r \to r+1} - CM_{r \to r}),\ r=1..6.$$

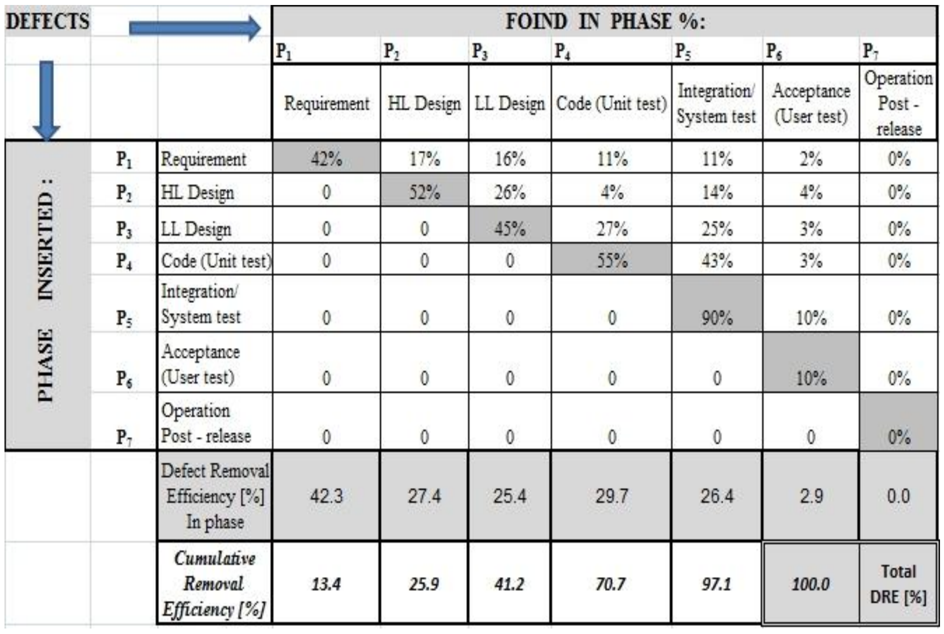
For the given large (~11300 FP, Java implementation about 600KLOC of source code) project example from [2], and for original data of process defect removal effectiveness given in Fig. 11, and simulated calculations of two Scenarios 1 and 2 are shown in Fig. 12, Fig.13 and Fig. 14.

Calculated Matrix of Improvement Potential calculation for Scenario 2 is given in Fig 12. From Fig. 13 and 14, we can easy find **maximal improvement potential point to be in phase P3**.



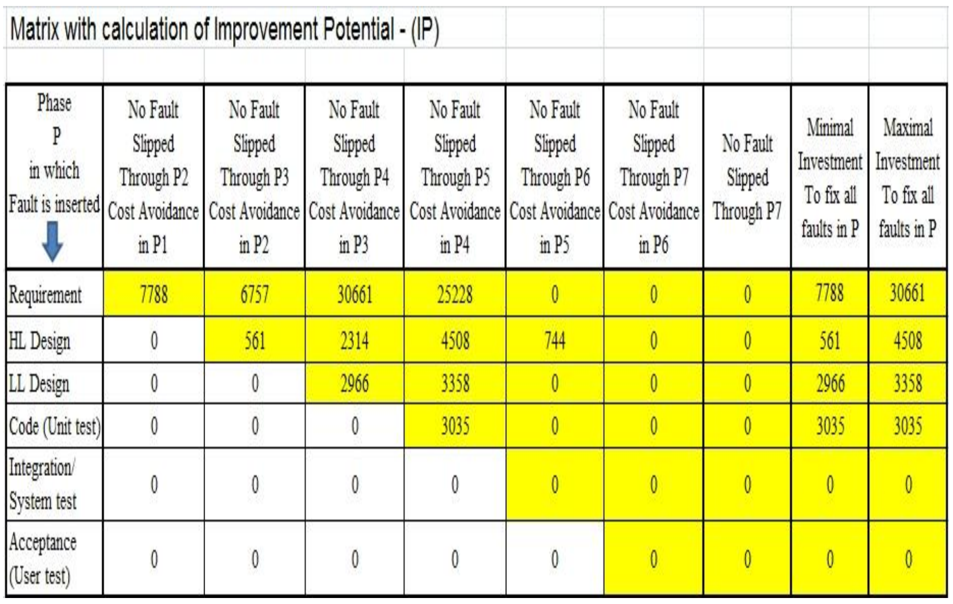
DEFECTS			FOIND IN PHASE :							
			P_1	P_2	P_3	P_4	P_5	P_6	P_7	
			Requirement	HL Design	LL Design	Code (Unit test)	Integration/ System test	Acceptance (User test)	Operation Post - release	Total Injected in phase
PHASE INSERTED :	P_1	Requirement	1,515	602	579	402	391	89	0	3578
	P_2	HL Design	0	805	400	60	223	60	0	1548
	P_3	LL Design	0	0	750	452	420	54	0	1676
	P_4	Code (Unit test)	0	0	0	2,421	1895	114	0	4430
	P_5	Integration/ System test	0	0	0	0	45	5	0	50
	P_6	Acceptance (User test)	0	0	0	0	0	10	0	10
	P_7	Operation Post - release	0	0	0	0	0	0	0	0
		Number of Defects found and removed in phase	1515	1407	1729	3335	2974	332	0	**11292**

Fig. 10. Original Software Process Defect Containment Matrix [2]



DEFECTS			FOIND IN PHASE %:						
			P_1	P_2	P_3	P_4	P_5	P_6	P_7
			Requirement	HL Design	LL Design	Code (Unit test)	Integration/ System test	Acceptance (User test)	Operation Post - release
PHASE INSERTED :	P_1	Requirement	42%	17%	16%	11%	11%	2%	0%
	P_2	HL Design	0	52%	26%	4%	14%	4%	0%
	P_3	LL Design	0	0	45%	27%	25%	3%	0%
	P_4	Code (Unit test)	0	0	0	55%	43%	3%	0%
	P_5	Integration/ System test	0	0	0	0	90%	10%	0%
	P_6	Acceptance (User test)	0	0	0	0	0	10%	0%
	P_7	Operation Post - release	0	0	0	0	0	0	0%
		Defect Removal Efficiency [%] In phase	42.3	27.4	25.4	29.7	26.4	2.9	0.0
		Cumulative Removal Efficiency [%]	***13.4***	***25.9***	***41.2***	***70.7***	***97.1***	***100.0***	**Total DRE [%]**

Fig. 11. Software Defect Containment Percentage Matrix – PCE



Matrix with calculation of Improvement Potential - (IP)

Phase P in which Fault is inserted	No Fault Slipped Through P2 Cost Avoidance in P1	No Fault Slipped Through P3 Cost Avoidance in P2	No Fault Slipped Through P4 Cost Avoidance in P3	No Fault Slipped Through P5 Cost Avoidance in P4	No Fault Slipped Through P6 Cost Avoidance in P5	No Fault Slipped Through P7 Cost Avoidance in P6	No Fault Slipped Through P7	Minimal Investment To fix all faults in P	Maximal Investment To fix all faults in P
Requirement	7788	6757	30661	25228	0	0	0	7788	30661
HL Design	0	561	2314	4508	744	0	0	561	4508
LL Design	0	0	2966	3358	0	0	0	2966	3358
Code (Unit test)	0	0	0	3035	0	0	0	3035	3035
Integration/ System test	0	0	0	0	0	0	0	0	0
Acceptance (User test)	0	0	0	0	0	0	0	0	0

Fig. 12. Matrix with Calculation of Improvement Potential for Scenario 1

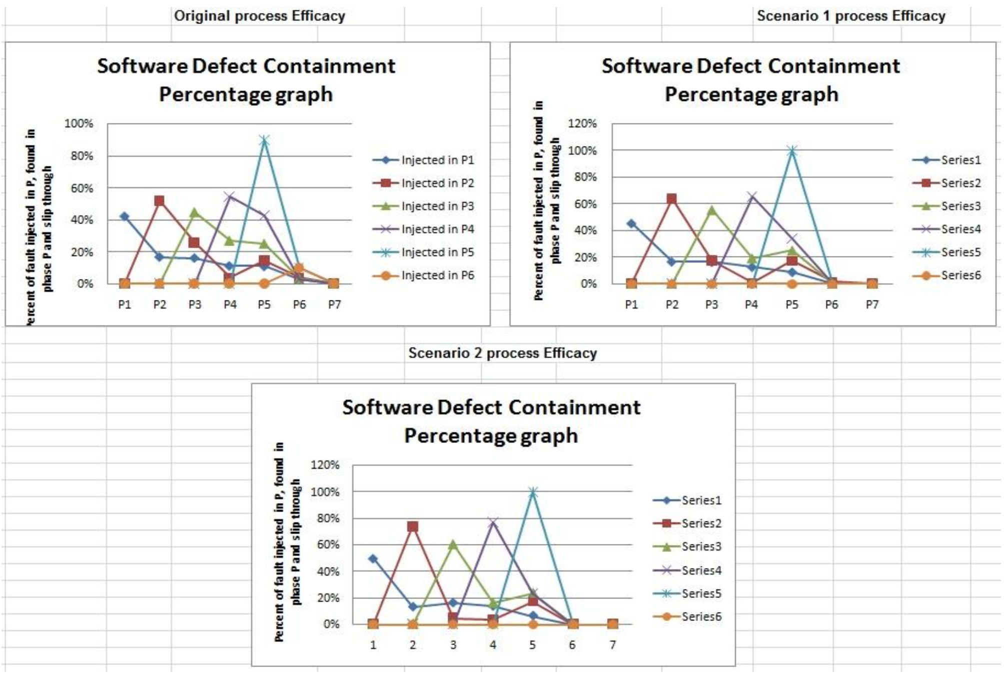


Fig. 13. Calculated Defect Containment Percentage Matrix – for original and anlized scenarios 1 and 2

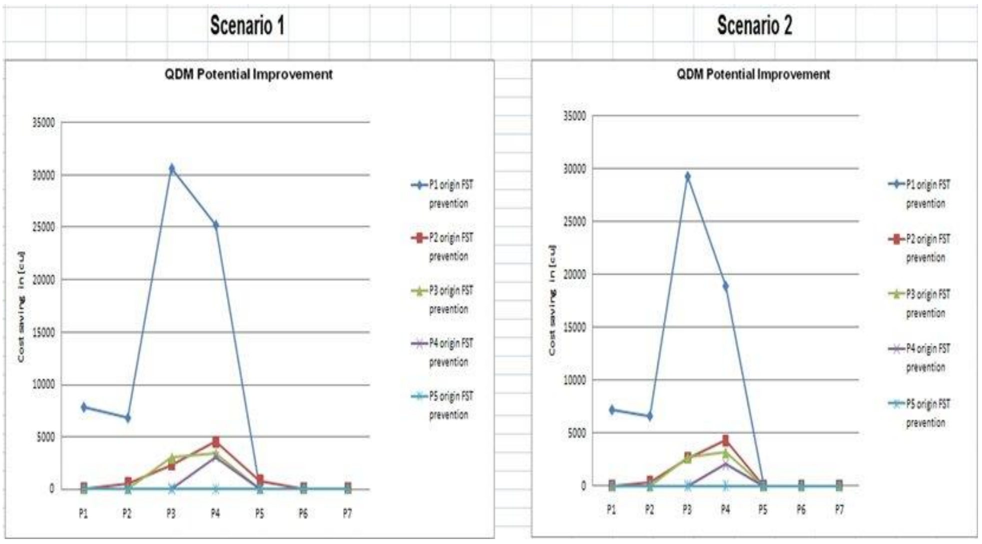


Fig. 14. Graph with calculation of Improvement Potential in [cu] for Scenario 1 and 2

We described ind this section, as answer to the RQ4 and RQ5, a Software Quality Optimization strategy of OptimalSQM framework, which is a continuous, iterative process throughout the application lifecycle resulting in zero-defect software that delivers value from the moment it goes live, with Simulated Defect Removal Cost Savings model.

6. Conclusion

The initial main research question that was posed for the complete research in this project was: How can software testing be performed efficiently and effectively i.e. **Optimal**, that is, do we have a framework model targeted specific software testing domains or problem classes described in the paper? To be able to address the main research question several other research questions needed to be answered first (RQ2–RQ5). Thus, since this project is based upon the main research question, it was worthwhile taking the time to examine the current practice in different projects and see how software quality is measured and, especially, software testing was practiced [1-8] as we described in Section 2. In Section 3 and 4 we described our OptimalSQM framework which presents a set of best practice models and techniques integrated in optimized and quantitatively managed software testing process (OptimalSQM), expanding testing throughout the SDLC. In Section 5, we explained how can Advanced Quantitative Defect Management (AQDM) Model be enhanced (as answer to RQ4 and RQ5) is practically useful for determining which activities need to be addressed to improve the degree of early and cost-effective software fault detection. To enable software designers to achieve a higher quality for their design, a better insight into quality predictions for their design choices, test plans improvement using Simulated Defect Removal Cost Savings model is offered in paper.

Finally, this paper presents and validates a method for measuring the efficiency of the software test process to achieve early and cost-effective software fault detection. That is, it determines how fault statistics can be used for assessing a test process and then quantify the improvement potential of changing the process. The described method assesses a software development organization through the following three steps:

1. Determine which faults that could have been avoided or at least found earlier, i.e. FST.
2. Determine the average cost of faults found in different phases.
3. Determine the improvement potential from the metrics obtained in (1) and (2), i.e. measure the cost of not finding the faults in the right phase.

The practical applicability of the method was determined by applying it on two industrial software development projects. In the studied projects, potential improvements were foremost identified in the the largest improvement potential is in the **LL Design** test phase, i.e. the phase triggered **30661 [cu]** of unnecessary costs in later phases due to a large FST from faults injected in Requirement phase. Therefore, the primary usage of the values is to serve as input to an expected ROI calculation according to given formula, when prioritizing possible improvement actions we can improve DRE cost for ROI=39.2%. That is, the **LL Design** phase inserted, or did not capture faults present at least, too many faults that slipped through to later phases.

References

1. Galin, D.: Software Quality Assurance:From theory to implementation, Pearson Education Limited, ISBN 0201 70945 7, pp. 135-142. (2004)
2. Frost and, A., Campo,M.: Advancing Defect Containment to Quantitative Defect Man, *CrossTalk*, December. pp. 24-28, (2007)
3. Lim, S. So, Y, Cha, S. D. and Kwon, Y. R.: “An Empirical Study on Software Error Detection: Voting, Instrumentation, and Fagan Inspection,” Proceedings of the Asia-Pacific Software Engineering Conference, IEEE Computer Society Press, Dec. (1995)
4. Jones, C.: Estimating Software Costs. 2nd edition. McGraw-Hill, New York.(2007)
5. Lazić, Lj.,Mastorakis, N.:Cost Effective Software Test Metrics, WSEAS TRANSACTIONS on COMPUTERS , Issue 6, Volume 7, June. pp. 599-619, (2008)
6. Lazić, Lj., Mastorakis, N.:Orthogonal Array application for optimal combination of software defect detection techniques choices, WSEAS TRANSACTIONS on COMPUTERS, Issue 8, Volume 7, pp.1319-1336, August.(2008)
7. Lazić, Lj., Mastorakis, N.:Optimizing Test Process Action Plans by Simulated Defect Removal Cost Savings, 11th WSEAS Int.Conf. on AUTOMATIC

CONTROL, MODELLING & SIMULATION (ACMOS'09), Istanbul, Turkey, May 30,pp. 280-287, June 1.(2009)

8. Lazić, Lj.: The Integrated and Optimized Software Testing Process. PhD Thesis, School of Electrical Engineering, Belgrade, Serbia.(2007)
9. Lars-Ola D.: Early and Cost-Effective Software Fault Detection, PhD Thesis, Blekinge Institute of Technology, SWEDEN, Section 2.(2007)
10. McConnell, S. :Professional Software Development, Addison Wesley, ISBN 0-321-19367-9.(2004)
11. Boehm, B. , Abts, C., Brown, A., Chulani,S., Clark, B.,Horowitz,E., Madachy,R., D. Reifer,D., Steece,B.: Software Cost Estimation with COCOMO II, Prentice Hall.(2000)
12. Cohen, C. F., Birkin, S. J., Garfield, M. J. and Webb, H. W.:Management Conflict in Software Testing. Communications of the ACM, 47(1),pp. 76-81. (2004)
13. Dhaliwal, J., Onita C., A framework for aligning Testing and Development, Proceedings of the Workshop on Advances & Innovations in Systems Testing. (2007)
14. Hunter and Blosch:Managing the New IT Risks". Gartner.(2003)
15. Pettichord, B. :Testers and Developers Think Differently: Understanding and Utilizing the Diverse Traits of Key Players on your Team. Software Testing & Quality Engineering, 2(1), pp. 42-45. (2000)
16. Sabherwal, Hirschheim and Goles:Information systems – business strategy alignment: The dynamics of alignment: Insights form a punctuated equilibrium model. Strategic information management: Challenges and strategies in managing information systems, (Galliers and Leidner, Eds), Butterworh-Heinemann, Oxford pp 311-346. (2003)
17. Lazić Lj., Velasević, D.:Applying simulation and design of experiments to the embedded software testing process. STVR, Volume 14, Issue 4, John Willey & Sons, Ltd., p257-282. (2004)
18. Popovic S, and Lazic, Lj.:Orthogonal Array And Virtualization As A Method For Improvement Configuration Testing, Proceedings of 1st IEEE Eastern European Regional Conference on the Engineering of Computer Based Systems - ECBS-EERC 2009, Novi Sad, pp.148-149. (2009)
19. Lazić, Lj. , Kolašinac, A., Avdić, Dž.:The Software Quality Economics Model for Software Project Optimization. WSEAS TRANSACTIONS on COMPUTERS, Issue 1, Volume 8, pp.21-47.(January 2009)
20. Morasca, S., Serra-Capizzano, S.: On the analytical comparison of testing techniques. In: International Symposium on Software Testing and Analysis (ISSTA '04). Association for Computing Machinery, Boston, Massachusetts, USA, pp. 154–164. (2004)
21. Houston, D., Keats, B.:Cost of Software Quality: A Means of Promoting Software Process Improvement, Quality Engineering, 10:3, pp. 563-573, March. (1998)
22. Müller, M.:About the Return on Investment of Test-Driven Development", ICSE'03, Portland, Oregon. (2003)
23. Chen, T., Kuo, F, Merkel,R.: On the statistical properties of testing effectiveness measures. The Journal of Systems and Software 79,pp. 591–601. (2006)
24. Wang, Q. et al.:Estimating Fixing Effort and Schedule based on Defect Injection Distribution, Softw. Process Improve. Pract.; 13: pp.35–50. (2008)

Ljubomir Lazić is an assistant professor of software engineering and computer science at State University of Novi Pazar, Serbia. He received the Ph.D. degree in electrotechnics from School of Electrical Engineering, Belgrade University in 2007. He was a Post-Doctoral Researcher at The WSEAS (The World Scientific and Engineering Academy and Society) of computer science from 2009 to 2010. So far, he has authored over 50 research papers. His current research interests are optimal software project management.

Received: September 23, 2009; Accepted: May 26, 2010.