A Framework of Software Testing Metrics – Part 1

Ljubomir Lazić, School of Electrical Engineering, Vojvode Stepe 283, Beograd, Serbia,

Nikos Mastorakis, Military Institutions of University Education, Hellenic Naval Academy, Terma Hatzikyriakou, 18539, Piraeus, Greece

Abstract:- Software testing needs to be measured in similar terms as overall software development process (SDP) in order to understand its true progress and make informed decisions. Basic considerations of **S**oftware **T**esting **M**etrics **F**ramework (STMF) and some commonly used testing metrics and where in testing process they apply are described in this, Part 1 article. Typically, software development is measured in terms of overall progress in meeting functional and business goals. By considering testing dimensions other than cost and schedule, managers and other team members can better understand and optimize the testing process, in effect opening the black box and managing testing more effectively is described in Part 2 article.
Key-Words:- software testing, testing metrics, size estimation, effort estimation, test effectiveness evaluation.

1 Introduction

Testing is often seen as a troublesome and uncontrollable process. As it is often performed, it takes too much time, costs too much, and does not contribute to product quality. Testing can become merely an exercise in identifying how bad the product will be in the field. However, with appropriate processes, it can be brought under control and can add significant value to the development process. Planning for testing on a software project is often challenging for program managers. Test progress is frequently unpredictable, and during software testing painful schedule and feature "surprises" typically occur. Software testing is often viewed as an obstacle—more as a problem and less as a vital step in the process. For this reason, testing is treated as a "black box" and addressed at the end of the schedule. While budget and time may be allocated for it, testing is not really managed in the same way as development. Typically, software development is measured in terms of overall progress in meeting functional and business goals. Software testing needs to be measured in similar terms to understand its true progress and make informed decisions. By considering testing dimensions other than cost and schedule, managers and other team members can better understand and optimize the testing process, in effect opening the black box and managing testing more effectively. In this way they can avoid costly and painful "surprises" late in the project.

Test metrics are an important barometer used to measure the effectiveness of the software testing process. Aim of this paper is to propose basic metrics of key software testing activities and artifacts in development processes that can be objectively measured, according to ISO 15939 – Software Measurement and SEI CMMI-SE/SW/IPPD/SS product suit [1-3] as a foundation for enterprise wide improvement of Integrated and Optimized Software Development / Testing Pocess (IOSTP) [7-11] i.e. **S**oftware **T**esting **M**etrics **F**ramework (STMF).

In testing we tend to focus on collecting internal IOSTP measures such as numbers of defects and innovation measures such as process improvement metrics. If we examine where our normal test metrics fit in the STMF we can see gaps in both quantitative and qualitative measures, which we may wish to address not only to focus on internal IOSTP results but to look at a balance between four measurable areas: financial measures such as profit and loss, customer measures such as market share and repeat business, internal measures such as numbers of defects in products and process violations, and innovation or learning measures such as number of new products developed and marketed. Our customers and managers may be interested in the financial impact of testing and customer satisfaction measures. Measure the past to predict the future. Software development is inherently a people-intensive enterprise, and software quality is influenced by many factors that vary tremendously among organizations. To achieve useful accuracy, software quality models must be calibrated for each specific development environment [1]. A case study acquires historical data on one or more projects. We construct models that could have been developed during the historical project, and calculate assessments that could have been made. The accuracy of those assessments is then evaluated against actual experience. This gives us confidence in predictions for a current project. Exploit your gold mines. Our approach to software quality modeling is aptly described as data mining, especially when operational faults are rare. Data mining is most appropriate when one seeks valuable bits of knowledge in large amounts of data collected for some other purpose, and when the amount of data is so large that manual analysis is not possible. Many software development organizations have very large databases for project management, configuration management, and problem reporting which capture data on individual events during development. For large systems or product lines, the amount of available data can be overwhelming. Manual analysis is certainly not

possible. However, we have found that these databases do contain indicators of which modules will likely have operational faults [11].

One metric is not enough. Much of the literature on software metrics is aimed to demonstrate the value of individual metrics. However, this does not fulfill our purpose: to build industrial-strength quality models.

Our experience with modeling empirical data from industry has indicated that a model with one software metric as the only independent variable does not have useful accuracy and robustness. Lines of code is not enough. McCabe cyclomatic complexity is not enough. The metric that is most highly correlated to faults is not enough. Recent case studies have demonstrated that multiple independent variables give more accurate results than models with just one independent variable [4]. The cost of collecting many metrics from source code (or other software product), rather than just a few, is not a practical problem for conventional metrics, because a metric-analyzer software tool is capable of measuring many metrics in one pass. We have found it is more effective to begin with many metrics, and then to apply data mining techniques to choose those with statistically significant empirical relationships to faults. Code metrics are not enough. The development histories of modules often differ radically. For example, modules from early releases have been used or tested more than recently developed modules. A stable module may have been developed by only one person, while other modules may have been modified by many different programmers. Indicators of such variations can significantly improve model accuracy and robustness. For example, our case studies have shown that a simple indicator of reuse from a prior release can be a significant independent variable in both classification and regression models. A case study of the Automatic Target Tracking Radar System - ATTRS [9], showed that the likelihood of discovering additional faults during integration and test in a spiral life cycle can be usefully modeled as a function of the module history prior to integration. In other words, process-related measures derived from configuration management data and problem reporting data may be adequate for software quality modeling, without resorting to software product measurement tools and expertise. Empirical validation must be realistic. Due to the many human factors that influence software reliability, controlled experiments to evaluate the usefulness of empirical models are not practical. Therefore, we take the case study approach to demonstrate their usefulness in a real-world testing. To be credible, the software engineering community demands that the subject of an empirical study be a system with the following characteristics [5]: (1) developed by a group, rather than an individual; (2) developed by professionals, rather than students; (3) developed in an industrial environment, rather than an artificial setting; and (4) large enough to be comparable to real industry projects. Our case studies fulfill all of these criteria through collaborative arrangements with development organizations.

Test metrics and data gathering regarding the testing costs, testing failure costs, and defects are essential to manage and control testing function efficiently and effectively. Accurate data and relevant metrics provide information for decision making in relation to quality of products and processes. Otherwise the release decisions, further investments, and process changes are troublesome to justify without proper information. Hard data about the current situation also concretizes the true facts enabling to set up feasible and rational objectives.

By establishing appropriate metrics, an organization can balance the cost of testing with the benefits derived from that testing. In order for metrics to be effective, the data collected must allow an organization to understand clearly:

• When the cost of further testing would outweigh the risk to the business.

• The cost to fix defects at the various stages of a project life cycle.

• The potential risk and subsequent costs to the business if the amount of testing were to be reduced.

This information can then be used to provide the organization with an informed basis of decision and effective ways to:

• Estimate the testing budget/spend.

• Spend more efficiently for future projects.

• Potentially reduce the overall costs of testing, realizing maximum value.

• Reduce total development and production support costs.

During individual projects, project metrics can be compared with accumulated experience to provide an early indication of quality levels and the accuracy of estimates. This in turn enables effective management and cost control at a project management level.

In section 2, basic considerations of **S**oftware **T**esting **M**etrics **F**ramework (STMF) and some commonly used testing metrics and where in testing process they apply are described. Finally in section 3, some concluding remarks are given.

2 Why metrics specific to SW Testing are essential

Software measurement is a challenging but essential component of a healthy and highly capable software engineering culture. In this article, we describe some basic software measurement principles and suggest some metrics that can help you understand and improve the way your organization operates i.e. **S**oftware **T**esting **M**etrics **F**ramework (STMF). Plan your measurement activities carefully because they can take significant effort to implement and the payoff will come over time. Software projects are notorious for running over schedule and budget, yet still having quality

problems. Software measurement lets you quantify your schedule, work effort, product size, project status, and quality performance. If you don't measure your current performance and use the data to improve your future work estimates, those estimates will just be guesses. Because today's current data becomes tomorrow's historical data, it's never too late to start recording key information about your project. You can't track project status meaningfully unless you know the actual effort and time spent on each task compared to your plans. You can't sensibly decide whether your product is stable enough to ship unless you're tracking the rates at which your team is finding and fixing defects. You can't quantify how well your new development processes are working without some measure of your current performance and a baseline to compare against.

2.1 Major components used for STMF definition

Metrics help you better control your software projects and learn more about the way your organization works. Specifically, the measurements described in this paper first answers the question of whether Software Testing is "doing the right thing" (effectiveness). Once there is assurance and quantification of correct testing, metrics should be developed that determine whether or not Software Testing "does the thing right" (efficiency) as we did during M&S of Optimized Software Testing model which combine Risk Management and Earned Value Management called RBOST [10]. You can measure many aspects of your software products, projects, and processes. The trick is to select a small and balanced set of metrics that will help your organization track progress toward its goals. Major components (depicted in Fig. 1) of proposed **S**oftware **T**esting **M**etrics **F**ramework are: 1) The Goal Question Metric (GQM) process, created by Victor Basili and his colleagues at the University of Maryland, is a good place to begin targeting the specific measurement needs of an organization [6,7], 2) Balanced Scorecard (BSC) that ensures set of measures providing coverage of all elements of performance and avoid hidden trade-offs and 3) Process Model Performance measures that are most meaningful with respect to selected areas of performance, prefere outcome then output measures over process and input measures.

The main emphasis of GQM is goal directed measurement. An organization usually starts with generic goals that must be refined. For example, "Reduce the number of failures found on a project". This is certainly a goal, but is it well enough refined? One technique to further refine goals, making them specific enough that they are applicable to the direction of the organization, is the SMART technique.

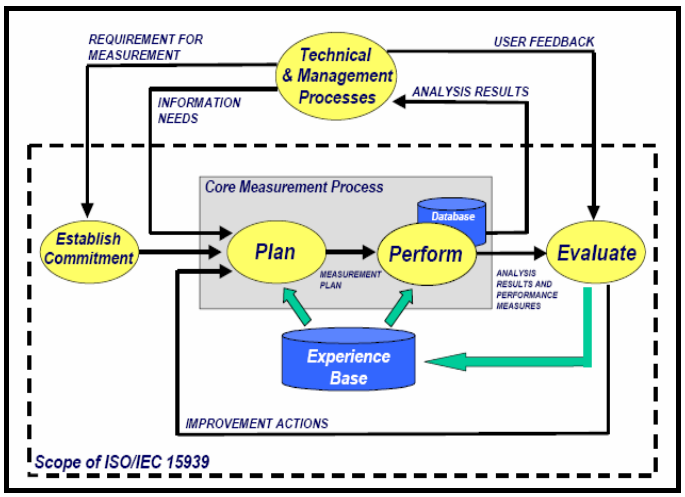

Fig. 1 Major components used for STMF definition

This consists of five parts:

Specific – Is the goal specific? Even for developers and testers working on the project, a percentage and timeframe should quantify the words "reduce" and "failures".

Measurable – Can the goal be gauged in comparison to other data? In this example, the answer appears to be "yes". What is lacking is why this quantity is being measured? What decisions are being made? What conclusions can be drawn? It is important to consider questions like these when refining goals.

Attainable – Is there agreement that this goal is achievable? Has consensus from the rest of the team been obtained? Are resources allocated to work on the goal?

Relevant – Is the goal impractical or imprecise? Is the goal scaled in the proper perspective? Is the goal within the scope of what you are responsible for and expected to accomplish?

Time-limited – Does the goal have a specific start and end date? Is there time in the project schedule allocated toward collecting data and tracking progress toward the goal? In the example above, this is not the case.

In addition to "SMART", every valid goal should represent a "stretch". In other words, achievement of the goal is not something that will be accomplished without effort and focus. When the organization reaches the goal, there will be agreement that improvement has definitely occurred. For each of the fundamental issues there are key questions that the project manager must periodically ask to ensure that the project remains on course and under control. To answer these questions, specific categories of measurement data must be available to the project manager. The issues, key questions related to each issue, and categories of measures necessary to answer the questions are show in Table 1.

Table 1. The issues, key questions related to each issue, and categories of measures

Issue	Key Questions	Measurement Category
1. Schedule & Progress	Is the project meeting scheduled milestones?	1.1 Milestone Performance
	How are specific activities and products progressing?	1.2 Work Unit Progress
	Is project spending meeting schedule goals?	1.3 Schedule Performance
	Is capability being delivered as scheduled?	1.4 Incremental Capability
2. Resources & Cost	Is effort being expended according to plan?	2.1 Effort Profile
	Are qualified staffs assigned according to plan?	2.2 Staff Profile
	Is project spending meeting budget objectives?	2.3 Cost Performance
	Are necessary facilities and equipment available as planned?	2.4 Environment Availability
3. Growth & Stability	Are the product size and content changing?	3.1 Product Size & Stability
	Are the functionality and requirements changing?	3.2 Functional Size & Stability
	Is the target computer system adequate?	3.3 Target Computer Resource Utilization
4. Product Quality	Is the software good enough for delivery?	4.1 Defect Profile
	Is the software testable and maintainable?	4.2 Complexity
5. Development / Testing Performance	Will the developer be able to meet budget and schedules?	5.1 Process Maturity
	Is the developer efficient enough to meet current commitments?	5.2 Productivity
	How much breakage to changes and errors has to be handled?	5.3 Rework
6. Technical Adequacy	Is the planned impact of the leveraged technology being realized?	6.1 Technology Impacts

2.2 Basic software testing process metrics

By focusing data collection activities on measurement categories that answer the key issue questions the project can minimize resources devoted to the measurement process. Among many Goals and Problems identified in former SDP/STP, before IOSTP deployment [8,9], our focus for STP improvement for demonstration purpose in this paper were issues - Development/Testing Performance and Product Quality i.e. only to these sampled issues, key questions related to each issue, and categories of measures necessary to answer the questions are show in Table 2 to 6 and some graphical presentations in figures 2 to 4.

Table 2. Key questions related to each issue, and categories of measures

4. Product Quality	Is the software good enough for delivery?	4.1 Defect Profile
5. Development / Testing Performance	Is the developer efficient enough to meet current commitments?	5.2 Productivity

Measuring the impact and consequences of problems that arise during testing is a critical step in the process. This should include analysis of collected measurements and calculated metrics to find out how much of the software is affected by a given problem, at what point during testing a problem was found, and what kinds of problems regression tests are attempting to uncover. The idea is to generate questions about the goal that will lead to specific metrics. A few questions to consider are:

- Is this project similar enough to the previous project that this type of comparison makes sense?
- What are the causes of critical defects?
- What data about duration testing indicates that 20% more critical failures can be found using these techniques?
- In the last product, what was the percentage of "critical" failures found, for the corresponding time period, as compared to the total?

- How many critical defects are expected for the same period on the next project?
- What duration test suite is appropriate for this project?
- Does duration testing enable finding a higher percentage of critical defects than regular testing?

Once a list of valid questions are created, measurements are generated. When considering metrics, it is often helpful to list the raw data that must be collected. This raw data is sometimes referred to as "primitive metrics". In this example, some important raw data is:

- Number of critical defects with a severity level of three and four.
- Time in duration testing.
- Total number of defects found in duration testing time period.
- Number of critical defects found on the last project for the corresponding time period.
- Number of total defects on last project for the corresponding time period.

Table 3. Measurement Category and Specific Measures

Measurement Category	**Specific Measures**
4.1 Defect Profile	4.1.1 Problem Report Trends 4.1.2 Problem Report Aging 4.1.3 Defect Density 4.1.4 Failure Interval

Once the raw data is defined, more complex, or "computed" metrics are generated based on combinations of primitive metrics.

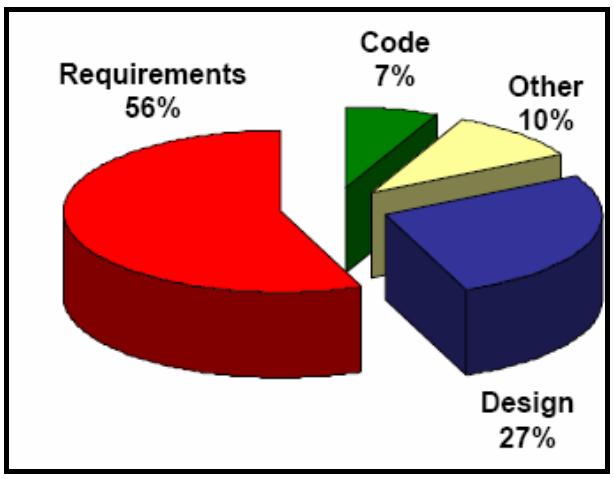

Fig. 2 Typical Distribution of Bugs

Deriving measurements from raw data and translating that data into something useful to managers and/or developers is essential in tracking real progress towards a goal. Important computed metrics in this example are:

- Number of critical failures found in duration testing time period / Total number of failures found in duration testing time period.
- Number of critical failures (severity 3&4) found in corresponding time period on previous project / Total number of failures found in corresponding time period on previous project.

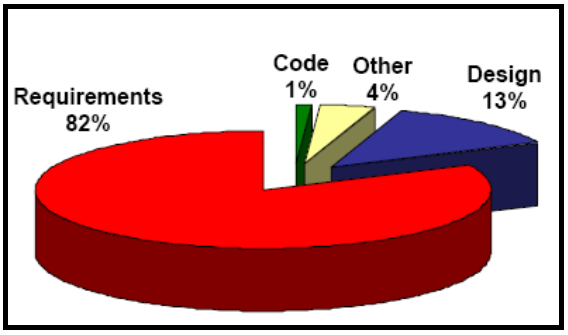

Fig. 3 Typical Distribution of Effort to Fix Bugs

After collection and analysis phase statistical methods and tools are used to identify and confirm root causes of defects. Not only must analysis of the data be performed, but also an in depth analysis of the process to ensure an understanding of how the work is actually being done must be performed to identify inconsistencies or problem areas that might cause or contribute to the problem. Deliverables of this phase are: data and process analysis, root cause analysis, quantifying the gap/opportunity and checkpoints for completion is to identify gaps between current performance and the goal performance.

Status of Open Deficiencies in Delivered Projects

Severity Levels	Number of Deficiencies That Have Been Open x Days				Totals
	x < 30	30 < x ≥ 60	60 < x ≥ 90	x > 90	
Severity 1	2	1			3
Severity 2	3	1	1		5
Severity 3	3	2	1	1	7
Severity 4	4	3	3	2	12
Severity 5	8	6	3	3	20
Totals	20	13	8	6	47

Fig. 4 Typical time to Fix Bugs vs severity levels

Root Cause Analysis should be done to:

- Generate list of possible causes (sources of variation).
- Segment and stratify possible causes (sources of variation).
- Prioritize list of 'vital few' causes (key sources of variation).
- Verify and quantify the root causes of variation.

Table 4. Focus question and specific measure

4 PRODUCT QUALITY	
Are difficult problems being deferred?	**4.1.2 Problem Report Aging**
Are reported problems being closed in a timely manner?	**4.1.2 Problem Report Aging**
Do report arrival and closure rates support the scheduled completion date of integration and test?	**4.1.1 Problem Report Trends**
FOCUS QUESTION	**SPECIFIC MEASURE**
How long does it take to close a problem report?	**4.1.2 Problem Report Aging**
How many problem reports are open? What are their priorities?	**4.1.1 Problem Report Trends**
How many problems reports have been written?	**4.1.1 Problem Report Trends**
How much code is being reused?	**4.2.6 Depth Of Inheritance**
How often will software failures occur during operation of the system?	**4.1.4 Failure Interval**
How reliable is the software?	**4.1.4 Failure Interval**
What components are candidates for rework?	**4.1.3 Defect Density**
What components have a disproportionate amount of defects?	**4.1.3 Defect Density**
What components require additional testing or review?	**4.1.3 Defect Density**
What is the program's expected operational reliability?	**4.1.4 Failure Interval**
What is the quality of the software?	**4.1.3 Defect Density**

In order to quantify the Gap/Opportunity answering the questions:

- What is the cost of poor quality as supported by the team's analysis?
- Is the process severely broken such that a re-design is necessary?
- What are the revised rough order estimates of the financial savings/opportunity for the improvement project?
- Have the problem and goal statements been updated to reflect the additional knowledge gained from the analyze phase?
- Have any additional benefits been identified that will result from closing all or most of the gaps?
- What were the financial benefits resulting from any 'ground fruit or low-hanging fruit' (quick fixes)?
- What quality tools were used to get through the analyze phase?

Table 5. Measurement Category and Specific Measures

Measurement Category	**Specific Measures**
5.2 Productivity	5.2.1 Product Size/Effort Ratio 5.3.2 Functional Size/Effort Ratio 5.8.1 Tracking Defect Containment

In proposed STMF our focus is on software Error and Defect Root Cases Analysis applying Defect Classification scheme as described in our paper about Software Testing Process Improvement to achieve a high ROI of 100:1 [7].

Table 6. Focus question and specific measure

FOCUS QUESTION	**SPECIFIC MEASURE**
How efficiently is software being produced?	**5.2.1 Product Size/Effort Ratio**
What is Phase Defect Detection effectiveness?	**5.8.1.1 Phase Containment Effectiveness**
What is Defect Escape effectiveness?	**5.8.1.2 Defect Containment Effectiveness**
What is Post-Released Defect number?	**5.8.1.3 Total Containment Effectiveness**
How much effort was expended on fixing defects in the software product?	**5.3.2 Rework Effort**
Is product being developed at a rate to be completed within budget?	**5.2.1 Product Size/Effort Ratio**
Is the amount of rework impacting cost or schedule?	**5.3.2 Rework Effort**
Is the amount of rework impacting the cost and schedule?	**5.3.1 Rework Size**
Is the planned software productivity rate realistic?	**5.2.1 Product Size/Effort Ratio**
What software development activity required the most rework?	**5.3.2 Rework Effort**
What was the quality of the initial development effort?	**5.3.1 Rework Size**

This information is needed to monitor the overall progress of the software through testing and to make informed decisions about software release. Once initial measurements are defined using the GQM paradigm, it

is essential to verify that the metrics align with the departments (teams) that make up the organization. So by combining all of the different perspectives of schedule, functionality, code, and problem resolution, it is possible to understand and manage software testing, rather than treating it as a black box as we explained in our paper of proposed STMF, Part 2 [12].

3 Conclusion

Although it is important to measure the quality of the product under development, it is equally important to measure the effectiveness and efficiency of Software Testing itself as an activity – not a service. We proposed basic metrics of key software testing activities and artifacts in development processes that can be objectively measured, according to ISO 15939 – Software Measurement and SEI CMMI-SE/SW/IPPD/SS product suit [1-3] as a foundation for enterprise wide improvement of Integrated and Optimized Software Development / Testing Pocess (IOSTP) [7-11] i.e. **S**oftware **T**esting **M**etrics **F**ramework (STMF). Specifically, the measurements described in this paper first answers the question of whether Software Testing is "doing the right thing" (effectiveness). Once there is assurance and quantification of correct testing, metrics should be developed that determine whether or not Software Testing "does the thing right" (efficiency). By measuring effectiveness and efficiency, a Software Testing organization can better communicate its own importance using factual information. Often, there are early warning signs that testing is going to have problems. These show up in the details of the analysis and design phases of the tests themselves. They appear in the form of incomplete or deferred work due to missing information, improperly managed problems recorded against key functionality, and other "small" indicators accumulating over time. If these indicators are spotted far enough ahead of time by managers, developers, and the testers themselves, work can be done to head problems off while they are still small. This in turn ensures that the testing group is better prepared for the software and that the software is better prepared for testing. Measuring the impact and consequences of problems that arise during testing is a critical step in the process. So by combining all of the different perspectives of schedule, functionality, code, and problem resolution, it is possible to understand and manage software testing, rather than treating it as a black box.

References

[1] S. H. Kan. *Metrics and Models in Software Quality Engineering*, Second Edition, Addison-Wesley, 2003.

[2] CMM Product Development Team. Capability Maturity Model for Software (CMM), Version 1.1, CMU/SEI-93-TR-24, ESC-TR-93-177. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA; 64 pp., 1993.

[3] CMMI Product Development Team. Capability Maturity Model Integration for Software Engineering (CMMi), Version 1.1, CMU/SEI-2002-TR-028, ESC-TR-2002-028. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA; 707 pp., 2002.

[4] T. M. Khoshgoftaar, E. B. Allen, W. D. Jones, and J. P. Hudepohl. Data mining for predictions of software quality. *International Journal of Software Engineering and Knowledge Engineering*, 1999.

[5] Nagappan, N., Williams, L., Vouk, M., Osborne, J. "Early Estimation of Software Quality Using In-Process Testing Metrics: A Controlled Case Study", Third Software Quality Workshop, co-located with the International Conference on Software Engineering (ICSE 2005), pp. 46-52, May 2005.

[6] V. R. Basili, G. Caldiera, H. D. Rombach. *The Goal Question Metric Approach*, Encyclopedia of Software Engineering, volume 1, John Wiley & Sons, 1994, pp. 528-532

[7] Lj. Lazić, N. Mastorakis. Software Testing Process Improvement to achieve a high ROI of 100:1, 6th WSEAS Int. Conf. On MATHEMATICS AND COMPUTERS IN BUSINESS AND ECONOMICS (MCBE'05), March 1-3, Buenos Aires, Argentina 2005.

[8] Lj. Lazić, D. Velašević, N. Mastorakis. A Framework of Integrated and Optimized Software Testing Process, WSEAS Conference, August 11-13, Crete, Greece, 2003 also in WSEAS TRANSACTIONS on COMPUTERS, Issue 1, Volume 2, January 2003.

[9] Lj. Lazić, D. Velašević. Applying Simulation and Design of Experiments to the Embedded Software Testing Process", SOFTWARE *TESTING, VERIFICATION AND RELIABILITY*, Volume 14, Number 4, p 257-282, John Willey & Sons, Ltd., 2004.

[10] Lj. Lazić, Mastorakis, N. RBOSTP: Risk-based optimization of software testing process Part 2", *WSEAS TRANSACTIONS on INFORMATION SCIENCE and APPLICATIONS*, Issue 7, Volume 2, p 902-916, July 2005, ISSN 1790-0832.

[11] Lj. Lazić, N. Mastorakis. "Faster, Cheaper Software Error Detection with Assured Confidence – Part 1", 5th WSEAS International Conference on APPLIED COMPUTER SCIENCE (ACOS '06), Hangzhou, China, Sponsored by WSEAS and WSEAS Transactions, Co-Organized by WSEAS and Zhejiang University of Technology, April 16-18, 2006.

[12] Lj. Lazić, N. Mastorakis. A Framework of Software Testing Metrics – Part 2, 11h WSEAS CSCC (CIRCUITS-SYSTEMS-COMMUNICATIONS-COMPUTERS) Multiconference, Agios Nikolaos, Crete Island, Greece, July 23-28, 2007.