Software Quality & Testing Metrics

LJUBOMIR LAZIĆ
State University of Novi Pazar
Vuka Karadžića bb, 36 300 Novi Pazar, SERBIA
llazic@np.ac.rs ; http://www.np.ac.rs

Abstract: - Software Quality Assurance without measures is like a jet with no fuel. Development of high quality software is very complicated and unreliable task, but the management of software development and testing process is much harder without appropriate software quality metrics and measurement establishment that assure process testing quantitative management in order to increase efficiency of software bugs detection and removal. Process improvement enables the same amount of software to be built in less time with less effort and fewer defects. Progress control tracks defects found during development in order to avoid premature delivery and to ensure the reliability goals are achieved. In-progress control and reporting of development progress gives continuous visibility to both developers and purchasers. This paper focuses on software testing and the measurements which allow for the quantitative evaluation of this critical software development process. This paper describes the Software Quality & Testing Metrics Model, which is a framework for establishing and interpreting in-process metrics in software development. The model has been validated and used on large sample of software projects in a mature software development organization. The central issue for in-process metrics, the concept and definition of the model, and its use are discussed. Examples of metrics real-life projects are provided.

Key-Words: - Software Quality, In-process Measurements, Testing Metrics, Quantitative Evaluation

1 Introduction

The business value of a software product results from its quality as perceived by both acquirers and end-users. Therefore, quality is increasingly seen as a critical attribute of software, since its absence results in financial loss as well as dissatisfied users, and may even endanger lives.

When examining troubled software projects, it always happens that the main reason for delay or termination is due to excessive volumes of serious defects. Conversely, large software projects that are successful are always characterized by excellence in both defect prevention and defect removal. It can be concluded that achieving state of the art levels of software quality control is the most important single objective of software process improvements.

Quality control is on the critical path for advancing software engineering from its current status as a skilled craft to become a true profession.

Our research [1][1] of innovative software producing units are committed to continuously improving both the software development process and the software product in order to remain competitive in today's global community. Software product quality and software process improvement commence with addressing the testing process in a quantitative manner. The continuous monitoring of the testing process allows for establishing an adequate level of confidence for the release of high quality software products and for the quantification of software risks, elements which traditionally have plagued the software industry.

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. Defect Distribution, Defect Density and Defect Type metrics allow for quantifying the quality of software modules, while Defect Age, Defect Detection Rates and Defect Response Time metrics allow for pinpointing software inspection and testing process shortcomings. Code coverage and testing effort measurements complement the defect metrics and provide additional software product as well as

[1] This work was supported in part by the Ministry of Education and Science of the Republic of Serbia under Grant No. TR-35026 entitled as:"Software Development Environment for optimal software quality design".

process quality indicators. The analysis of code coverage during system testing will also allow for identifying which code segments get exercized the most during the execution of business transactions. By identifying the code segments which get exercized the most, the testers can focus on most heavily traversed statements while designing and executing tests. Obviously, time limitations do not allow for complete code coverage and testers must maximize testing benefits versus the time allocated to them for testing.

Perhaps the most important testing measurement is the one which will provide an indication concerning the readiness for the software code to be released. Most project managers which want to release code based on testing measurements generate graphs of cummulative number of defects detected per some meaningful unit of time (i.e.hours, days or weeks depending on module size). The slope of the curve will steadily approach zero as the testing process concludes and the software code can be released. Such graphs, per each code module, are an important project management tool for monitoring the maturity of the respective modules regarding defect detection and correction. Software testing is a process area which traditionally needs improvement and the metrics presented in this paper allow for the quantification of both the product quality and of the respective software testing process. Software firms which have used all or even some of the presented metrics achieved significant improvements. This is achieved due to the fact that line management has visibility to the development process and decisions are not made based on intuition alone, but, with the sound interpretation of the available software testing metrics.

2 Related work

Assessing software quality is extremely hard and expensive, and the state of practice falls short of expectations. As noted, there are some inherently difficult problems with software quality assessments that defy solutions. However, all is not lost. The field of software assurance is advancing rapidly and a number of software quality/reliability groups both in the academic and commercial worlds are actively researching next generation techniques and tools to better create and assess software quality.

Authors in [1] discusses software test metrics and their ability to show objective evidence necessary to make process improvements in a development organization. When used properly, test metrics assist in the improvement of the software development process by providing pragmatic, objective evidence of process change initiatives.

S. H. Kan in the book [2], provids huge amount of quantitative examples of Metrics and Models in Software Quality Engineering, but just a few samples about software tseting metrics.

Defect management is one of the core demands for the success of the project and it is a fact that team performance influences the effective defect management [4-6].

Authors in [3] and [5] feel that software engineering has now taken a new perspective where project manager looks for professional management and software quality assurance methodologies during developmental activities.

Our investigation therefore confirmed that it is important to measure the quality of the product under development, and also, 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, as a foundation for enterprise wide improvement. This will bring in awareness on the efficiency of project manager in accurate estimation and resource allocation by indicating the impact analysis of these resources on the success of a project.

3. Software Quality Fundamentals

Agreement on quality requirements, as well as clear communication on what constitutes quality, require that the many aspects of quality be formally defined and discussed. Over the years, authors and organizations have defined the term “quality” differently.

3.1 Model and quality characteristics

Quality models for software have been developed to assess both software processes and software products. Software process engineering has made many advances in the last decade, and has introduced numerous quality models and standards concerned with the definition, implementation, measurement, management, change and improvement of the software engineering process itself. Assessing the quality of software products requires that the quality characteristics of software be defined and measured. The quality of a software product is determined by its properties. Some of these are inherent to the software product, while others are assigned, by priority, to the software product during its requirements definition process. Process quality is concerned with the technical and

managerial activities within the software engineering process that are performed during software acquisition, development, maintenance and operation.

In contrast, software product quality models describe the many quality attributes of software. Software quality means different things to different people. This makes the entire concept highly context-dependent. The IEEE, ISO, DoD and several other agencies and individuals have offered definitions for software quality. Some of these definitions "conformance to requirements", "meeting users' composite expectations", "value to some person", and "fitness for use" are useful but extremely vague because there is no definite way to state whether or not the final software product conforms to these definitions.

During the early years, the terminology for software product quality differed from one model of software quality to another. Each model had a different umber of hierarchical levels and number of characteristics, and especially confusing were the different naming conventions used. Based on these efforts, the International Organization for Standardization (ISO) normalized three software product quality models (i.e. internal quality, external quality and quality in use) (ISO9126-01) and accompanied them with a set of guides like ISO/IEC14598 which explains how to evaluate a software product (see Fig. 1).

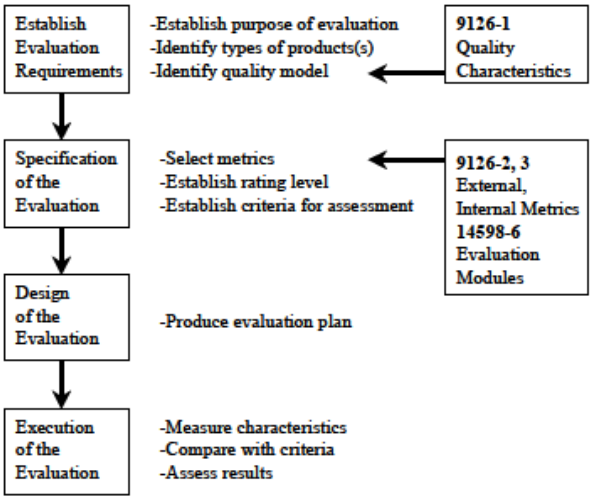

Fig. 1. ISO/IEC 14598 – Evaluation process

In an attempt to impart formalism and to provide a systematic definition for software quality, the ISO 9126 defines 21 attributes that a quality software product must exhibit. These attributes (shown in Table 1) are arranged in six areas: functionality, reliability, usability, efficiency, maintainability, and portability. Recent advances in software quality measurement techniques allow us to measure some of these attributes (These attributes are shown in bold in table 1).

Table 1 – The ISO 9126 quality attributes

MAIN ATTRIBUTE	SUB ATTRIBUTE
Functionality	Suitability
	Accuracy
	Security
	Interoperability
Reliability	Maturity
	Fault-tolerance
	Recoverability
Usability	Understandability
	Learnability
	Operability
Efficiency	**Time behavior**
	Resource behavior
Maintainability	Analyzability
	Changeability
	Stability
	Testability
Portability	Adaptability
	Installability
	Conformance
	Replaceability

However, there still seems to be no straightforward means to measure the remaining attributes, let alone derive a metric for overall quality based on these attributes. Without clear methods and measures, we are back to square one, with no means to say anything quantitative about the final software product's quality.

In the end, we have only a vague idea of how to define software quality, but nothing concrete. We have some idea of measuring it, but no clearly defined methods. What we are left with is a grab bag of diverse standards and methods. Faced with a bewildering array of definitions and standards, software projects, already under heavy burden of schedule and budget pressures, are faced with a dilemma: the dilemma of how to create and assess the quality of software products.

3.2 Definition of quality metrics

Quality requirements of software products are often described in vague and broad terms. As a consequence it makes it difficult for software engineers to determine how quality influences their assignment and it is almost impossible for test engineers to evaluate the quality of the software product as no concrete and quantitative reference, of

ISBN: 978-960-474-304-9

what quality in that context means, exists. In order to implement software process and product quality, the software engineer must choose and document quality characteristics at the outset of the projects during the requirements definition process. This paper describes a possible way: Measuring software product quality during in-process testing, to define and measure software product quality as described in Fig. 2.

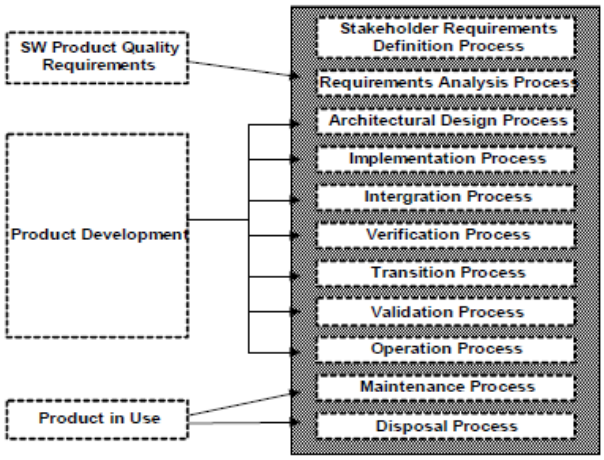

Fig. 2 Product life cycle mapping to the technical process life cycle

We have already stated that both functional and non-functional requirements are captured using the requirements definition process [8]. Functional requirements specify what a product should do. Nonfunctional requirements, also called quality requirements, place constraints on the functional requirements. The quality requirements need to refer to a quality model like the one described in ISO/IEC 9126-1 to be well understood. Quality requirements address important issues of quality for software products. Software product quality requirements are needed for:

• *Specification* (including contractual agreement and call for tender)
• *Planning* (e.g. feasibility analysis and translation of external quality requirements into internal quality requirements)
• *Development* (early identification of quality problems during development)
• *Evaluation* (objective assessment and certification of software product quality)

Judging software quality is very difficult task without technical process life cycle as described above. Experience shows tendency to associate the quality of the software with the number of defects encountered during testing and after release is high. This approach is flawed because: Any problem in the software gets recorded as a defect ranging from GUI Error to application failure.

If we continue to measure software quality just by the number of defects there will never be able to make any software release. Any software is bound to have defects. The business of the company need not depend on the number of defects in the software. A new perspective of looking at the quality of the software is by **Assessing the risks** associated with the software release. When a user uses the system, what are the likely risks associated with it and what it means to the user.

4 Measuring and Managing In-process Software Quality by Testing

Quality could be improved, first, by removing defects as early as possible in the development life cycle using techniques such as reviews, walkthroughs and inspections (IEEE-1028), and second, by putting in place defect prevention and defect detection activities.

4.1 In-process metrics

Using in-process metrics to determine the quality status of a software project under development is easier said than done. How can you interpret a test-phase defect curve correctly to reflect the true quality status of the project? If the defect curve is below a given baseline, is this a positive sign? What if the lower curve is due to slow progress in testing? Likewise, how does one establish meaningful metrics for design reviews and code inspections and interpret them correctly? How about metrics for stability and reliability?

This paper describes the Effort/Outcome Model, which is a framework for establishing and interpreting inprocess metrics in software development. The model has been validated and used on large scale software projects in a mature software development organization. The central issue for in-process metrics, the concept and definition of the model, and its use are discussed. Examples of metrics real-life projects are provided.

Defect data during the testing phase of the development process is perhaps the most widely used source of data for in-process metrics. Metrics formed using test defect data can be the overall number of defects, the overall defect density, or the defect arrival curve. To measure the status of in-process quality validly, one has to interpret the in-process metrics and data correctly. Take for example the defect arrival curve, at least several characteristics have implications to the quality status of the project: the shape and level of the tail end of the curve, the time the curve peaks relative to when the product will be shipped, and release to release

ISBN: 978-960-474-304-9

(or "like" product) comparisons. Just the defect metrics alone, however, may not be able to do the job (providing an accuracy status of in-process quality) adequately. What if the decline of the defect curve is due to poor testing effort?

4.2 Effort / Outcome Model Testing

In order to have good confidence in interpreting test defect arrival metrics, one have to rely upon the test effort related information. Indicators like testing effort, testing effectiveness and testing progress are related to the effort we extended into testing, and defect rates, defect volumes, or defect arrival curves indicate the resultant outcome based on testing effort. If we take a closer look at most in-process metrics, we can classify them into two groups: those that measures the effectiveness or effort, and those that indicate the outcome as presented in Fig.3. We call the two groups the effort indicators (e.g., test effectiveness assessment, test progress S curve, CPU utilization during test, inspection effort) and the outcome indicators (e.g., defect arrivals during testing-- total number and arrivals pattern, number of system crashes and hangs, mean time to unplanned initial program load (IPL), inspection defect rate), respectively.

5 Practical Testing Measurements of the Study

Measurement of a test process is a required competence for an effective software test manager for designing and evaluating a cost effective test strategy. Effective management of any process requires quantification, measurement and modeling. Software Metrics provide quantitative approach to the development and validation of the software process models. Metrics help organization to obtain the information it needs to continue to improve its productivity, reduce errors and improve acceptance of processes, products and services and achieve the desired Goal. This paper, focusing on metrics lifecycle, various software testing metrics, need for having metrics, evaluation process and arriving at ideal conclusion have also been discussed in the next part of this paper [9].

Based on the types of testing performed, following are the types of software testing metrics:

1. Manual Testing Metrics
2. Performance Testing Metrics
3. Automation Testing Metrics

Following part of this paper shows different software testing metrics formulas and examples of their calculation.

Development Phase	Effort Indicators	Outcome Indicators
Requirements	• ***Coverage*** **– req. analysis and reviews** • ***Coverage*****- Customer validation**	• **equirements problem/issue rate**
Design	• *Design reviews coverage* • *Design reviews* - inspector-hour per review • *Design reviews progress metrics* **-** Design defect rate**,** Design rework indicators	• Design defect rate • Design rework indicators
Code	• *Code inspection coverage* • *Code inspection progress metrics*- static code analysis coverage	• Inspection defect rate • Rework indicators
Unit Test/Integration /Build	• *Unit test plan coverage* • ***T****est progress metrics* • ***S****tatic code analysis coverage* • *Effectiveness*/ function coverage of build verification test	• Defect metrics
FVT (Functional Verification Test)	• *Test plan function coverage/usage* • *Test Progress metrics*	• Overall test defect rate • **Test defect arrival** pattern • Defect backlog metrics
SVT (System Verification Test)	• Same as FVT • CPU utilization • **O**ther indicators for system stress • Coverage of business/customer usage scenarios	• Same as FVT • Metrics for crashes and hangs • Mean time between outage
Beta	• Relative program progress • % of customers on production Vs testing • Customer usage of functions	• Customer problem rate • Cust. satisfaction

Fig. 3 Examples of Effort and Outcome Indicators along the Phases of the Development Process

A. Manual Testing Metrics

1) Test Case Productivity (TCP)

This metric gives the test case writing productivity based on which one can have a conclusive remark.

$$TotalCase\,\mathrm{Pr}oductivity = \frac{TotalTestSteps}{EffortsInHours}[steps(s)/hour] \quad (1)$$

Example 1:

From the past project database, in the next table, five test cases and their step nombers are given:

Test Case Name	Raw Steps
XYZ_1	30
XYZ_2	32
XYZ_3	40
XYZ_4	36
XYZ_5	45
Total Raw Steps	183

According to formula (1) we can calculate efforts took for writing 183 steps is 8 hours.
TCP=183/8=22.8, than, *Test case productivity = 23 steps/hour.*
One can compare the Test case productivity value with the previous release(s) and draw the most effective conclusion from it.

2) Defect Acceptance (DA)
This metric determine the number of valid defects that testing team has identified during execution.

$$DefectAcceptance = \frac{NumberofValidDefects}{TotalNumberofDefects} * 100.[\%] \quad (2)$$

The value of this metric can be compared with previous release for getting better picture.

3) Defect Rejection (DR)
This metric determine the number of defects rejected during execution.

$$Defect\ Rejection = \frac{NumberofDefects\ Rejected}{TotalNumberofDefects} * 100.[\%] \quad (3)$$

This metric gives the percentage of the invalid defect the testing team has opened and one can control, if required, in future.

4) Bad Fix Defect (B)
Defect whose resolution give rise to new defect(s) are bad fix defect.
This metric determine the effectiveness of defect resolution process.

$$BadFixDefect\% = \frac{NumberofBadFixDefect(s)}{TotalNumberofValidDefects} * 100.[\%] \quad (4)$$

This metric gives the percentage of the bad defect resolution which needs to be controlled.

5) Test Execution Productivity (TEP)
This metric gives the test cases execution productivity which on further analysis can give conclusive result.

$$TestExecution\ Productivity = \frac{NumberofTCexecuted\ (Te)}{ExecutionEffort\ in\ Hours} * 8.[Exec./Day] \quad (5)$$

Where *Te* is calculated as explained below, where:
Base Test Case = No. of TC executed at least once.
T (1) = No. of TC Retested with 71% to 100% of Total TC steps,
T (0.66) = No. of TC Retested with 41% to 70% of Total TC steps
T (0.33) = No. of TC Retested with 1% to 40% of Total TC steps

6) Test Efficiency (TE)
This metric determine the efficiency of the testing team in identifying the defects. It also indicated the defects missed out during testing phase which migrated to the next phase.

$$TestEfficiency = \frac{DT}{DT + DU} * 100.[\%] \quad (6)$$

Where,
DT = Number of valid defects identified during testing.
DU = Number of valid defects identified by user after release of application. In other words, post-testing defect.

7) Defect Severity Index (DSI)
This metric determine the quality of the product under test and at the time of release, based on which one can take decision for releasing of the product i.e. it indicates the product quality.

$$DefectSeverityIndex = \frac{\sum(SeverityIndex * No.OfValidDefect(s)ofSI}{TotalNumberOfValidDefects} \quad (7)$$

Where, *No. of Valid Defect(s) ofSI* means: Number of valid defects for this severity.
One can divide the Defect Severity Index in two parts:
a) DSI for All Status defect(s):
This value gives the product quality under test.
b) DSI for Open Status defect(s):
This value gives the product quality at the time of release. For calculation of DSI for this, only open status defect(s) must be considered.

$$DSI(open) = \frac{\sum(SeverityIndex * No.OfOpenDefect(s)ofSI}{TotalNumberOfValidDefects} \quad (8)$$

B. Performance Testing Metrics
1) Performance Scripting Productivity (PSP)
This metric gives the scripting productivity for performance test script and have trend over a period of time.

$$PerformanceScripting\ Productivity = \frac{\sum OperationsPerformed}{EffortinHours}[Oper.(s)/hour] \quad (9)$$

Where Operations performed is:
1. No. of Click(s) i.e. click(s) on which data is refreshed.
2. No. of Input parameter
3. No. of Correlation parameter
Above evaluation process does include logic embedded into the script which is rarely used.
Example 2:

ISBN: 978-960-474-304-9

Operation Performed	Total
No. of clicks	10
No. of Input Parameter	5
No. of Correlation Parameter	5
Total Operation Performed	20

Efforts took for scripting = 10 hours.
Performance scripting productivity =20/10=2 operations/hour.

2) Performance Execution Summary
This metric gives classification with respect to number of test conducted along with status (Pass/Fail), for
various types of performance testing.
Some of the types of performance testing: -
1. Peak Volume Test.
2. Endurance/Soak Test.
3. Breakpoint/Stress Test.
4. Failover Test

3) Performance Execution Data - Client Side
This metric gives the detail information of Client side data for execution.
Following are some of the data points of this metric
1. Running Users
2. Response Time
3. Hits per Second
4. Throughput
5. Total Transaction per second
6. Time to first byte
7. Error per second

4) Performance Execution Data - Server Side
This metric gives the detail information of Server side date for execution.
Following are some of the data points of this metric:
1. CPU Utilization
2. Memory Utilization
3. HEAP Memory Utilization
4. Database connections per second

5) Performance Test Efficiency (PTE)
This metric determine the quality of the Performance testing team in meeting the requirements which can be
used as an input for further improvisation, if required.

$$PerformanceTestEfficiency = \frac{\text{Requirement}DuringPT}{(\text{Requirement}DuringPT + \text{Requirement}AfterSignoffOfPT} * 100.[\%]$$

(10)

To evaluate this one need to collect data point during the performance testing and after the signoff of the performance testing.

Some of the requirements of Performance testing are:
1. Average response time.
2. Transaction per Second.
3. Application must be able to handle predefined max user load.
4. Server Stability
Example 3:
Consider during the performance testing above mentioned requirements were met. In production, average response time is greater than expected, then:
Requirement met during PT = 4;
Requirement not met after Signoff of PT = 1;
PTE = (4 / (4+1)) * 100 = 80%
Performance Testing Efficiency is 80%

6) Performance Severity Index (PSI)
This metric determine the product quality based performance criteria on which one can take decision for releasing of the product to next phase i.e. it indicates quality of product under test with respect to performance.

$$PerformanceSeverityIndex = \frac{\sum(SeverityIndex * No.Of\ \text{Req.}NotMetSI)}{TotalNo.Of\ \text{Req.}NotMetSI} * 100$$

(11)

If requirement is not met, one can assign the severity for the requirement so that decision can be taken for the product release with respect to performance.
Example 4:
Consider, Average response time is important requirement which has not met, then tester can open defect with
Severity as Critical.
Then Performance Severity Index = (4 * 1) / 1 = 4 (Critical)

C. Automation Testing Metrics
1) Automation Scripting Productivity (ASP)
This metric gives the scripting productivity for automation test script based on which one can analyze and
draw most effective conclusion from the same.

$$AutomationScripting\ \text{Pr}oductivity = \frac{\sum OperationsPerformed}{EffortinHours}[Oper.(s)/hour]$$

(12)
Where Operations performed is:
1. No. of Click(s) i.e. click(s) on which data is refreshed.
2. No. of Input parameter
3. No. of Checkpoint added
Above process does include logic embedded into the script which is rarely used.
Example 5:
Automation scripting productivity = 2.5 operations/hour.

2) Automation Test Execution Productivity (AEP)
This metric gives the automated test case execution productivity.

$$AutomationTestExecution\,\text{Pr}oductivity = \frac{TotalNo.ofTCexecuted(ATe)}{ExecutionEffort\,\text{sin}\,Hours} * 8.[Exec./Day] \quad (13)$$

Where Te is calculated as,

$$ATe = BaseTestCase + (T(0.33) * 0.33 + T(0.66) * 0.66) + T(1))$$

Evaluation process is similar to Manual Test Execution Productivity.

3) Automation Coverage
This metric gives the percentage of manual test cases automated.

$$AutomationCoverage = \frac{TotalNo.ofTCAutomated}{TotalNo.ofManualTC} * 100.[\%] \quad (14)$$

Example 6:
If there are 100 Manual test cases and one has automated 60 test cases then Automation Coverage = 60%.

4) Cost Comparison
This metrics gives the cost comparison between manual testing and automation testing. This metrics is used to have conclusive ROI (return on investment).
Manual Cost is evaluated as:
*Cost (M) =Execution Efforts (hours) * Billing Rate*
Automation cost is evaluated as:
*Cost (A) =Tool Purchased Cost (One time investment) + Maintenance Cost + Script Development Cost + (Execution Efforts (hrs) * Billing Rate)*
If Script is reused the script development cost will be the script update cost.
Using this metric one can have an effective conclusion with respect to the currency which plays a vital role in IT industry.

A. Effort Variance (EV)
This metric gives the variance in the estimated effort.

Operation Performed	Total
No. of clicks	10
No. of Input Parameter	5
No. of Checkpoint added	10
Total Operation Performed	25

Efforts took for scripting = 10 hours.
ASP=25/10=2.5

$$EffortVariance = \frac{ActualEffort - EstimatedEffort}{EstimatedEffort} * 100.[\%] \quad (15)$$

B. Schedule Variance (SV)
This metric gives the variance in the estimated schedule i.e. number of days.

$$ScheduleVariance = \frac{ActualNo.ofDays - EstimatedNo.ofDays}{EstimatedNo.ofDays} * 100.[\%] \quad (16)$$

C. Scope Change (SC)
This metric indicates how stable the scope of testing is.

$$ScopeChange = \frac{TotalScope - \text{Pr}eviousScope}{\text{Pr}eviousScope} * 100.[\%] \quad (17)$$

Where,
Total Scope = Previous Scope + New Scope, if Scope increases.
Total Scope = Previous Scope - New Scope, if Scope decreases.

6 Conclusion

Metrics are gaining importance and acceptance in corporate sectors as organizations grow, mature and strive to improve enterprise qualities. Measurement of a test process is a required competence for an effective software test manager for designing and evaluating a cost effective test strategy. Effective management of any process requires quantification, measurement and modeling.

In this paper we described the effort/outcome model for in-process software testing metrics and quality management. The model goes beyond the traditional way of interpreting metrics and assessing quality when a software development project is underway. We illustrated the use of the model for the testing phase as well as for the front end of the development process with regard to design and code inspections.

Improvement paths were discussed and possible effort and outcome indicators that can be used along the phases of software development life cycle were provided. The effort/outcome model testing and the metrics thus established, used, and interpreted, are clearly an effective way for quality management.

Future work will take several directions. More data will be collected to further validate our approach, the refinement of measurement process. cost and effectiveness of test metrics model.

References

[1] Lj. Lazić, N. Mastorakis. "Cost Effective Software Test Metrics", WSEAS TRANSACTIONS on COMPUTERS , Issue 6, Volume 7, June 2008, p599-619.

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

[3] J. Capers, *Applied Software Measurement*, 3rd edition; McGraw-Hill, New York, 2008.

[4] Suma, V., and Gopalakrishnan Nair, T.R. Defect management strategies in software development, book on Recent Advances in Technologies, Intecweb Publishers, Vienna, Austria, ISBN 978-953-307-017-9, 2009, p379-404.

[5] Lj. Lazić, "Software Testing Optimization by Advanced Quantitative Defect Management", COMPUTER SCIENCE AND INFORMATION SYSTEMS, vol. 7, No. 3, p. 459-487, 2010.

[6] B. Boehm and V. Basili, Software Defect Reduction Top 10 List, IEEE Computer, IEEE Computer Society, Vol. 34, No. 1, January 2001, pp. 135-137.

[7] Bennett, Ted L., and Paul W. Wennberg. "Eliminating Embedded Software Defects Prior to Integration Test."Dec. 2005.

[8] Cusumano, M., MacCormack, A., Kemerer, C., Grandall, B.(2003). Software development worldwide: the state of the practice, *IEEE Software*, 20(6): 28–34, 2003.

[9] G. Rothermel. and M.J. Harrold, "Analyzing Regression Test Selection Techniques", IEEE Transactions on Software Engineering, vol. 22, no. 8, pp. 529 – 551, 1996.