Incorporating Metrics in an Organizational Test Strategy

Wasif Afzal and Richard Torkar
Blekinge Institute of Technology,
S-372 25 Ronneby, Sweden
{waf, rto}@bth.se

Abstract

An organizational level test strategy needs to incorporate metrics to make the testing activities visible and available to process improvements. The majority of testing measurements that are done are based on faults found in the test execution phase. In contrast, this paper investigates metrics to support software test planning and test design processes. We have assembled metrics in these two process types to support management in carrying out evidence-based test process improvements and to incorporate suitable metrics as part of an organization level test strategy. The study is composed of two steps. The first step creates a relevant context by analyzing key phases in the software testing lifecycle, while the second step identifies the attributes of software test planning and test design processes along with metric(s) support for each of the identified attributes.

1. Introduction

There is a need for cost effective and efficient software testing processes that are able to meet todays demands of delivering low cost and quality software. Low cost and high quality software is a perfect couple but achieving it is indeed a challenge due to constraints of schedules, time to market and work force limitations. Often is the case that software is released in a rush to stay in competition with market but a compromise is made on adequate testing of the software, making it more prone to failures.

The perplexity of the situation can be reduced by having an effective software testing process in place, governed by an organizational level testing strategy. Assessment of the effectiveness of the software testing process has to rely on appropriate measures. Measures that are embedded in the organizational level testing strategy makes the underlying testing process activities visible, enabling the managers and engineers to better acknowledge the connections among various process activities. In addition measures provide evidences about process inadequacies; thereby facilitating improvements. As is shown by Burnstein et al. [10], measurement helps in improving the software testing procedures, methods, tools and activities by providing objective evidence for evaluations and comparisons. In short, knowing and measuring what is being done is important for an effective testing effort [40].

If we categorize software testing into phases of planning, design, execution and review; then the current literature presents enough metrics based on the test execution phase and on the basis of number of faults found during this phase. There is an apparent gap when we are interested in metrics for test planning and test design processes. This is rather surprising because test planning and test design processes have a significant impact on actual test execution, therefore the improvements in these two processes is bound to impact the test execution phase. Moreover, the artifacts resulting from planning and design phase (See Sections 4.1 and 4.2) are critical to the success of the testing effort as a whole. Therefore, we explore the possible metrics in software test planning and test design processes at system testing level to contribute towards defining an organizational level testing strategy that incorporates metrics as a means to control the testing process. Also, an organization needs to have a reference set of measurable attributes and corresponding metrics that can be applied at various stages during the course of execution of an organization wide testing strategy. Therefore, we attempt to present the measurable attributes and the corresponding metrics of software test planning and test design processes in a consolidated form.

The rest of this paper is divided into the following sections. Section 2 presents the research methodology. Section 3 introduces the reader to relevant definitions, while Section 4 provides an insight into the software testing life cycle used as a baseline in this paper. Sections 5 and 6 cover the different attributes relevant to software test planning and test design processes respectively, while Sections 7 and 8 cover the metrics relevant for the respective attributes. Finally, we conclude this paper in Section 9 and cover future work in Section 10.

2. Research method

Selection of appropriate information resources and an unbiased search is fundamental in reviewing literature. The intention of this study is to perform a survey of relevant work into metric support for software test planning and test design processes and to deduce important conclusions. The search strategy used is limited to electronic databases, namely IEEE Xplore and ACM Digital Library, using a combination of multiple key terms to reduce the differences in terminology. In addition to electronic search, manual search was performed on the proceedings of International Software Metrics Symposium. The studies that were not related to software test planning and design phase at the system testing level were excluded. Other electronic databases were not searched as the authors preferred books and authentic web references to complement IEEE Xplore and ACM Digital Library results. The authors believe that searching within additional electronic databases like Compendex and ISI Web of Science would have increased the breadth of primary studies. The authors acknowledge the fact that additional work is needed to mitigate threats to external validity.

For each paper of interest, relevant material was extracted and compared against other relevant literature, specifically the abstract, introduction and the conclusion of each of the articles. The references at the end of each article proved to be a useful source of further investigation. The summary performed on research articles was complemented by textbooks written by well-known authors in the field of software metrics and software testing. It is expected that by referring to multiple perspectives on a relevant topic, performing an exhaustive search in two of the largest research databases and then following the references in all relevant papers, contributed towards reducing internal validity threats.

3. Relevant definitions and concepts

Software measurement is integral in improving software development [35]. There can be two applications of software metrics, tactical and strategic. The tactical use of metrics is useful in project management whereas the strategic application of software metrics is in software process improvement [19].

An entity is an object (e.g. programming code, software under test) or an event (e.g. test planning phase) in the real world [16]. An attribute is the feature, property or characteristic of an entity which helps us to differentiate among different entities.

There can be three categories of measures depending upon the entity being measured, namely process, product and resource measures [16]. Process metrics are used to measure the characteristics of methods, techniques and tools employed in the development, implementation and maintenance of the software system [26].

4. Software testing lifecycle

The traditional approach to testing was seen as executing tests [13]; but a modern testing process is a lifecycle approach that includes different phases (for a variety of proposed software testing life cycle stages please see e.g. [13, 23, 25, 38, 44, 45]). Consolidating the different views leads us to categorize software testing into test planning, test design, test execution and test review phases.

4.1. Test planning

The goal of test planning is to take into account the important issues of testing strategy, resource utilization, responsibilities, risks and priorities. Also identification of methodologies, techniques and tools is part of test planning which is dependent on the type of software to be tested, the test budget, the risk assessment, the skill level of available staff and the time available [25]. The output of the test planning is the test plan document.

4.2. Test design

The test design process is very broad and includes critical activities like determining the test objectives (i.e. broad categories of things to test), selecting test case design techniques, preparing test data, developing test procedures, setting up the test environment and supporting tools. Determination of test objectives is a fundamental activity which leads to the creation of a testing matrix reflecting the fundamental elements that needs to be tested to satisfy an objective. The objectives are transformed into a list of items that are to be tested under an objective. The test objectives are then used to create test cases. The test cases then become part of a document called the test design specification [27]. After the test design specification is produced, a test case specification is constructed, which describes how the tests will be run. A test procedure describes a switch over from one test to another [45].

4.3. Test execution

The test execution phase involves allocating test time and resources, running all or selected test cases, collecting execution information and measurements and observing results to identify system failures [13, 27]. At system test level, normally testers are involved but developers and end users may also participate. For our purposes the above description

is sufficient (we are focusing on the software test planning and test design processes in this paper).

4.4. Test review

The purpose of the test review process is to analyze the data collected during testing to provide feedback to the test planning, test design and test execution activities. Different assessments can be performed as part of test review which includes reliability analysis, coverage analysis and overall defect analysis.

5. Attributes for test planning process

The attributes to be measured are dependent on the time and phase in the software development life cycle, new business needs and ultimate goal of the project [32]. Useful attributes for measurement during software test planning are identified in the next section.

5.1. Progress

Suspension Criteria. The suspension criteria for testing establish conditions for suspending testing. **Exit Criteria.** The exit criteria for testing needs to be based on metrics so that an exit criterion is flagged after meeting a condition. **Scope of Testing.** The scope of testing helps to answer how much of the software is to be tested and metrics helps answering this question. **Monitoring of Testing Status.** The testing status needs to be monitored as on time and in budget completion of testing tasks is essential to meet schedule and cost goals. **Staff Productivity.** Measures for testers productivity should be established at the test planning phase to help a test manager learn how a software tester distributes time over various testing activities [12]. **Tracking of Planned and Unplanned Submittals.** The test planning progress is affected by the incomplete or incorrect source documentation as submittals [19]. The tracking of planned and unplanned submittals therefore becomes important so that a pattern of excessive or erratic submittals can be assessed.

5.2. Cost

Testing Cost Estimation. Metrics supporting testing budget estimation are required as the test planning phase also has to establish the cost estimates. **Duration of Testing.** Metrics assisting creation of a testing schedule are required as a testing schedule is one of the important elements of test planning. **Resource Requirements.** The test planning activity has to estimate the number of testers required for carrying out the system testing activity. **Training Needs of Testing Group and Tool Requirement.** Metrics indicating the need of training for testing group and tool requirement needs are to be established.

5.3. Quality

Test Coverage. A testing group wants to measure percentage of code, requirements and design covered by a test set. **Effectiveness of Smoke Tests.** Metrics establishing the effectiveness of smoke tests are required to be certain that application is stable enough for testing. **Quality of Test Plan.** The quality of the test plan produced needs to be assessed for attributes that are essential for an effective test plan. **Fulfillment of Process Goals.** It is useful to measure the extent to which the test planning activity is meeting the process goals expected of it.

5.4. Improvement trends

Faults Prior to Testing. The count of faults (fault content) captured prior to testing during requirements analysis and design phases identify resource training needs and process improvement opportunities. **Expected Number of Faults.** An estimate of an expected number of faults helps gauging the quality of the software. **Bug Classification.** The test planning activity needs to classify bugs into types and severity levels.

Table 1 shows the attribute classification in different categories.

Table 1. Classification of test planning attributes.

Progress	Suspension criteria for testing
	Exit criteria
	Scope of testing
	Monitoring of testing status
	Staff productivity
	Tracking of (un)planned submittals
Cost	Testing cost estimation
	Duration of testing
	Resource requirements
	Training needs and tool requirements
Quality	Test coverage
	Effectiveness of smoke tests
	Quality of test plan
	Fulfillment of process goals
Improvement trends	Count of faults prior to testing
	Expected number of faults
	Fault classification

6. Attributes for test design process

After having defined the attributes of the software test planning process, we are interested in identifying the attributes of the software test design process. Identification of attributes leads to selection of measures, which is an important step in establishing a measurement program [5].

6.1. Progress

Tracking Testing Progress. Tracking (or monitoring) the testing progress gives early indication if the testing activity is behind schedule and to flag appropriate measures to deal with the situation. **Tracking Testing Defect Backlog.** A large number of unresolved faults prior to test design negatively impact the test progress [28]. Therefore, it is useful to track the testing defect backlog over time. **Staff Productivity.** Measurement of staff productivity in developing test cases helps in estimating cost and duration of incorporating change [16].

6.2. Cost

Cost Effectiveness of Automated Tools. When a tool is selected to be used for testing, it is beneficial to come up with metrics that evaluate the cost effectiveness of tool.

6.3. Size

Estimation of Test Cases. If an initial estimate of number of test cases required to fully exercise the application is available, then the activity of test case development can progress well. **Number of Regression Tests.** The number of regression tests is an important factor to ensure that any changes and bug fixes have been correctly implemented and does not negatively affect other parts of the software. **Tests to Automate.** It is important to take the decision about which test cases to automate as to balance the cost of testing as some tests are more expensive to automate than if executed manually [14].

6.4. Strategy

Sequence of Test Cases. Since not all test cases are of equal importance, they need to be prioritized according to a particular rationale. **Identification of Areas for Further Testing.** As exhaustive testing is rarely possible, the test design process needs to identify high-risk areas that require more testing [13]. These areas might be e.g. critical parts of a system or code that is being exercised often. **Combination of Test Techniques.** Pertaining to focus on system testing, it is important to know what combination of test techniques to use that discovers more faults. **Adequacy of Test Data.** Adequate test data needs to be prepared to expose as many faults as possible.

6.5. Quality

Effectiveness of Test Cases. Test effectiveness is the fault-finding ability of the set of test cases [23]. Measures that establish the quality of test cases produced are required to determine how good a test case is being developed. **Fulfillment of Process Goals.** As with test planning, it is useful to measure the extent to which the test design activity is meeting the expected process goals. **Test Completeness.** As the test cases are developed, it is imperative that the requirements and code are covered properly [40]. This is to ensure the completeness of tests [19].

The classification of attributes in different categories is presented in Table 2.

Table 2. Classification of test design attributes.

Progress	Tracking testing process
	Tracking testing defect backlog
	Staff productivity
Cost	Cost effectiveness of automated tools
Size	Estimation of test cases
	Number of regression tests
	Tests to automate
Strategy	Sequence of test cases
	Identification of areas for further testing
Quality	Combination of test techniques
	Adequacy of test data
	Effectiveness of test cases
	Fulfillment of process goals
	Test completeness

7. Metrics for test planning attributes

We proceed to address available metrics for each attribute within the individual categories of progress, cost, quality and improvement trends identified for software test planning.

7.1. Metrics support for progress

The category of progress included attributes named as suspension criteria for testing, the exit criteria, scope of testing, monitoring of testing status, staff productivity and tracking of planned and unplanned submittals. Monitoring of testing status is discussed in combination with tracking testing progress attribute while staff productivity is dis-

cussed with the same attribute being identified as part of software test design attributes.

7.1.1 Measuring suspension criteria for testing.

One way to suspend testing is when there are a specific number of severe faults or total faults detected by testing that makes it impossible for the testing to move forward. In such a case, suspension of testing acts as a safeguard for precious testing time-scales. Also in case of dependencies among the testing activities, if there is an activity on a critical path, the subsequent activities need to be halted until this task is complete [13]. Any incompleteness in the different aspects of test environment (hardware, communications, interfaces, documentation, software, supplies, personnel and facilities [13]) may also prevent testing of the application.

7.1.2 Measuring the exit criteria.

The metrics that help clarifying the exit criteria for testing includes rate of fault discovery in regression tests, frequency of failing fault fixes and fault detection rate.

7.1.3 Measuring scope of testing.

The factors influencing the decision regarding testing scope are driven by situational circumstances including number of available testing resources, amount of available testing time, areas that are more prone to faults, areas that are changed in the current release and areas that are most frequently used by the customer.

7.1.4 Tracking of planned and unplanned submittals.

Planned submittals are those that are required to be submitted according to a schedule and unplanned submittals are those that are resubmitted due to the problems in the submitted submittals. If any of these submittals are incomplete or incorrect, it can lead to situations where these submittals are to be resubmitted again so that testing activities can progress. Therefore, tracking of planned and unplanned submittals can identify potential problems in the process leading towards testing. Grady gives an example of tracking planned and unplanned submittals for a project on monthly basis [19].

7.2. Metrics support for cost

The category of cost included attributes named as testing cost estimation, duration of testing, resource requirements and training needs of testing group and tool requirement. The training needs of testing group and tool requirement are discussed in combination with tests to automate attributes identified as part of software test design attributes.

7.2.1 Measuring testing cost estimation, duration of testing and testing resource requirements.

We will use the term cost estimation here as a common term that includes predictions about the likely amount of effort time, budget and staffing levels required. Formal estimates regarding the time and resources for testing milestones are important as they allow planning for risks and contingencies in time [3, 13]. However, the important question is, is it possible to objectively estimate testing time? According to [14], estimation formulas that make use of complex equations are not only difficult to use, they are also rarely precise. Hence, what follows are some examples (found in [14, 24, 32]) of some simple to use models for estimating time and resources.

First of all, the development ratio method makes use of software development effort estimation as a basis for estimating the resource requirements for the testing effort, thereby helping to outline an estimated testing schedule. Another similar method estimating the tester to developer ratio makes use of heuristics. Project staff ratio method makes use of historical metrics by calculating the percentage of testing personnel from overall allocated resources planned for the project. The test procedure method uses the size measures of testing artifacts like the planned number of test procedures for estimating the number of person hours of testing and number of testers required. The task planning method makes use of the historical records of number of personnel hours expended to perform testing tasks to estimate a required level of effort. Expert opinion involves true experts making the effort estimates if size estimates are available to be used as benchmark data. Experts use work and activity decomposition and system decomposition to make estimations. Finally, estimation by analogy makes use of analogs (completed projects that are similar) and use of experience and data from those to estimate the new project. This method can both be tool-based or manual.

7.3. Metrics support for quality

The category of quality included attributes named as test coverage, effectiveness of smoke tests, the quality of test plan and fulfillment of process goals. Test coverage is discussed in combination with test completeness attribute being identified as part of software test design attributes.

7.3.1 Measuring effectiveness of smoke tests.

There are some established best practices when it comes to measurement of the effectiveness of smoke tests. These practices recommend smoke tests to be broad in scope as opposed to depth, smoke tests should exercise the system from end-end, they should cover the most frequently used and basic operations, smoke tests need to be automated

and made part of the regression testing suite, the number of smoke tests are to increase in size and coverage as the system evolves, a system test environment is required for smoke testing and smoke tests need to take advantage of historical database of fault-finding subset of test suites that have proved valuable over other projects.

7.3.2 Measuring the quality of the test plan.

A test plan can only be evaluated in terms of functions that it intends to serve [2]. According to the test plan evaluation model presented by [2], a test plan needs to possess the quality attributes of usefulness, accuracy, efficiency, adaptability, clarity, usability, compliance, foundation and feasibility. These quality attributes are to be assessed using heuristics i.e. accepted rules of thumb reflecting experience and study. Berger describes a multi-dimensional qualitative method of evaluating test plans using rubrics [7]. Rubrics take the form of a table, where each row represents a particular dimension, and each column represents a ranking of the dimension as excellent, average and poor.

7.3.3 Measuring fulfillment of process goals.

Test planning makes a fundamental maturity goal at the Testing Maturity Model (TMM) level 2 [10]. The goals at TMM level 2 support testing as a managed process. A managed process is planned, monitored, directed, staffed and organized. The setting of goals is important because they become the basis for decision making. The test plan includes both general testing goals and lower level goal statements. An organization can use a checklist to evaluate the process of test planning.

7.4. Metrics support for improvement trends

The category of improvement trends included attributes such as count of faults prior to testing, expected number of faults and bug classification. Count of faults prior to testing and expected number of faults are discussed in combination with identification of areas for further testing attribute which was identified in the software test design process.

7.4.1 Bug classification.

There are different ways to classify the faults found in the application during the course of system testing. One such classification categorizes the faults as algorithmic and processing, control, logic and sequence, typographical, initialization, data flow, data, module interface, code documentation and external hardware/software interfaces defects [10]. Gray [21] classifies the faults as being deterministic (permanent) and transient (temporary) faults. Vaidyanathan and Trivedi [47] extend this classification to include aging related faults that refer to potential fault conditions that accumulate gradually over time, leading to performance degradation.

8. Metrics for test design attributes

We proceed to address available metrics for each attribute within the individual categories of progress, quality, size, cost and strategy identified for software test planning.

8.1. Metric support for progress

The category of progress included attributes named as tracking testing progress, tracking testing defect backlog and staff productivity.

8.1.1 Tracking testing process.

The testing activity being monitored can be projected using graphs that show trends over a selected period of time. For the testing milestone of execution of all planned system tests, the data related to number of planned system tests currently available and the number of executed system tests at this date should be available. Number of requirements or features to be tested, number of equivalence classes identified, number of equivalence classes actually covered, number of degree of requirements or features actually covered, number of features actually covered/total number of features are some of the metrics for monitoring the coverage at the system testing level [10]. For monitoring of test case development at the system testing level, useful measures are number of planned test cases, number of available test cases and number of unplanned test cases [10]. The test progress S-curve compares the test progress with the plan to indicate corrective actions in case the testing activity falls behind schedule [28].

8.1.2 Tracking testing defect backlog.

Defect backlog is the number of defects that are outstanding and unresolved at one point in time with respect to the number of defects arriving. Then there are defects that are not fixed due to resource constraints. The defect backlog metric is to be measured for each release of a software product for release to release comparisons [28]. The defects that affect and halt the testing progress should be given priority in fixing. Another way to prioritize the reduction in defect backlog is on the basis of impact on customer which can help the developers to focus efforts on critical problems [36].

8.1.3 Staff productivity.

Brooks observed that adding more people on a software project will make it even more late. This law is believed to hold not as strongly in testing as in other areas of software engineering [8]. In spite of the fact that productivity measures for testers are not extensively reported, a test manger is interested in learning about the productivity of their staff and how it changes as the project progresses. Time spent in test planning and test case design, number of test cases developed and number of test cases developed/unit time are useful measures of testers productivity [10]. The evaluation of individual testers is multi-dimensional, qualitative, multi-source, based on multiple samples and individually tailored. The primary sources of information in this case are not numeric, rather specific artifacts are reviewed, specific performances are assessed and the work of testers is discussed with others to come up with an evaluation [29].

8.2. Metric support for quality

The category of quality included attributes named as effectiveness of test cases, fulfillment of process goals and test completeness.

8.2.1 Measuring effectiveness of test cases.

Common methods available for verifying test case specifications include inspections and traceability of test case specifications to functional specifications. However, there needs to be an in-process evaluation of test case effectiveness [11] so that problems are identified and corrected in the testing process before the system goes into production. Chernak describes a simple in-process test case effectiveness metric [11] defining test case effectiveness as the ratio of faults found by test cases to the total number of faults reported during a function testing cycle. Defect removal efficiency is another powerful metric for test effectiveness, which is defined as the ratio of number of faults actually found in testing to the number of faults that could have been found in testing. Defect age is another metric that can be used to measure the test effectiveness, which assigns a numerical value to the fault depending on the phase in which it is discovered. Defect age is used in another metric called defect spoilage to measure the effectiveness of defect removal activities. The effectiveness of the test cases can also be judged on the basis of the coverage provided. It is a powerful metric in the sense that it is not dependent on the quality of the software and also it is an in-process metric that is applicable when the testing is actually done.

8.2.2 Measuring fulfillment of process goals.

The software test design process should be able to perform the activities that are expected of it. The common way of assessing the conformance to a process is using a checklist. One such checklist is presented by Black in [9].

8.2.3 Measuring test completeness.

We emphasize more on specification coverage measures as these measures are more applicable at the system testing level. There are two important questions that need to be answered when determining the sufficiency of test coverage. One is that whether the test cases cover all possible output states and second is about the adequate number of test cases to achieve test coverage [37]. A common requirements coverage metric is the percentage of requirements covered by at least one test [40]. In black box testing, as we do not know the internals of the actual implementation, the test coverage metrics tries to cover the specification [39]. The specification coverage measures are recommended for safety critical domains [48]. Specification coverage measures the extent to which the test cases conform to the specification requirements. These metrics provide objective and implementation-independent measures of how the black box test suite covers requirements [48]. According to [48], three potential metrics that could be used to assess requirements coverage are requirements coverage, requirements antecedent coverage and requirements UFC (Unique First Cause Coverage). Another approach to measure testing coverage independent of source code is given by [17, 1]. This approach applied model checking and mutation analysis for measuring test coverage using mutation metric.

8.3. Metric support for cost

The category of cost included an attribute named cost effectiveness of automated tools.

8.3.1 Measuring cost effectiveness of automated tools.

The evaluation criteria for a testing tool must consider its compatibility with operating systems, databases and programming languages used organization wide. The performance requirements for significant applications under test in the organization, need for more than one tool, support for data verification, types of tests to automate and cost of training are important factors in the evaluation criteria. Specifically, the evaluation criteria for evaluating a testing tool includes ease of use, power, robustness, functionality, ease of insertion, quality of support, cost of the tool and fit with organizational policies and goals [10].

8.4. Metric support for size

The category of size included attributes named as estimation of test cases, number of regression tests and tests to automate. Estimation of test cases is discussed with combination of testing techniques attribute of the software test design process.

8.4.1 Measuring number of regression tests.

A number of regression test selection techniques are presented by Rothermel et al. in [41]. Use of program complexity measures (e.g. cyclomatic complexity and Halstead metrics), knowledge of software design, defect removal percentage and a measure of defect reported in each baseline help selecting regression tests [40]. Different metrics can be used to monitor regression testing including number of test cases reused, number of test cases added to the tool repository or test database, number of test cases re-run when changes are made to the software, number of planned regression tests executed, number of planned regression tests executed and passed [10].

8.4.2 Measuring tests to automate.

The software testing staff must be adequately trained in testing of applications because there is a strong relationship between increased training and improved worker productivity, profitability and shareholder value [13]. The testing group needs to understand the business needs, be technically proficient and have sound communication and writing skills. The different methods of training include mentoring, on-site commercial training, training in a public forum, in-house training and specialty training [13]. The testing strategy should consider making use of automated tool wherever the needs warrants its use. The benefits of automation include speed, efficiency, accuracy and precision, resource reduction and repetitiveness [34] but automation should not be considered as a substitute for human intuition and if automation runs without finding a fault, it does not mean that there is no fault remaining. Tasks that are repetitive in nature and tedious to be performed manually are prime candidates for an automated tool.

8.5. Metric support for strategy

The category of strategy included attributes named as sequence of test cases, identification of areas for further testing, combination of test techniques and adequacy of test data.

8.5.1 Sequence of test cases.

Sequence of test cases is dependent on the prioritization of test cases with the purpose of increasing the rate of fault detection, increasing the detection of high-risk faults, increasing the coverage of code under test and increasing the reliability of the system under test. Test case prioritization techniques can either be general or version specific [42]. General test case prioritization techniques are useful over a sequence of subsequent modified versions of a given program while version specific test case prioritization techniques are less effective on average over a succession of subsequent releases. While selecting a test case prioritization technique, there are different considerations, such as granularity levels, incorporation of feedback to adjust for test cases already executed and making use of the modified program version [15]. In [15], Elbaum et al. classify eighteen test case prioritization techniques into three categories. Experiments and case studies have shown that the use of test case prioritization techniques improves the rate of fault detection [15, 43].

8.5.2 Measuring identification of areas for further testing.

During code development, if it can be predicted which files or modules are likely to have the largest concentrations of faults in the next release of a system [6], the effectiveness and efficiency of testing activities can be improved. One way to predict the number of faults in files is using a fault-prediction model like a negative regression model. This model predicts the faults for each file of a release based on characteristics like file size, whether the file was new to the release or changed or unchanged from the previous release, the age of the file, the number of faults in the previous release and the programming language [6]. Khoshgoftaar et al. [31] used software design metrics and reuse information (i.e. whether a module is changed from previous release) to predict the actual number of faults in the module. Graves et al. reported a study in which the fault predictions in a module were based on modules age, the changes made to the module and the ages of the module [20]. Ohlsson and Alberg predict the fault proneness of software modules based entirely on the pre-coding characteristics of the system design [33]. Another approach to predict fault prone modules is using random forests [22].

8.5.3 Measuring combination of testing techniques.

The black box testing techniques is the dominant type of testing at the system level. A study by Torkar and Mankefors [46] compares three black box techniques namely equivalence partitioning, random testing and boundary value analysis to find the efficiency with respect to find-

ing severe faults. Equivalence partitioning was found to be the single most effective test methodology to detect faults that leads to severe failures. Another study carried out by Lauterbach and Randall (found in [49]) compared six testing techniques and the results indicated that a combination of different techniques discovered as many as 20 faults in the software that were not previously found. A study carried out by Basili and Selby [4] compared three testing strategies and one of their results showed that incase of professional programmers, code reading resulted in higher fault detection rates than functional and structural testing. An approach by Schroeder and Korel claim to significantly reduce the number of black box tests by identifying relationships between program inputs and outputs [43]. Kaner and Bach lists down a list of paradigms of black box software testing in [30]. A testing paradigm defines types of tests that are relevant and interesting. The list of paradigms for black box testing involves the testing techniques types of domain driven, stress driven, specification driven, risk driven, random/statistical, functional, regression, scenario/use case/transaction flow, user testing and exploratory. Moreover, there is a considerable consensus in using a combination of testing techniques to take advantage of the strengths of each [13, 14, 32].

8.5.4 Measuring adequacy of test data.

The prerequisites for an effective test data are a good data dictionary and detailed design documentation [14]. While acquiring the test data, there are several concerns to address including depth, breadth and scope of test data, data integrity during test execution and conditions specific data [14]. Gerhart and Goodenough laid the foundations of test adequacy criterion by defining it as a predicate that defines what properties of a program must be exercised to constitute a thorough test [18]. Weyuker defined 11 axioms for program based test adequacy criteria, a critique of this effort is that the paper is theoretical with few hints for practitioners (see e.g. [49]).

9. Conclusions

The majority of metrics in literature focuses on the test execution phase and are based on the number of faults found during test execution. This study contributes by consolidating the rather dispersed attributes of software test planning and test design processes. Similarly, partitioning the attributes in different categories is a step to a hierarchical view of the identified attributes, which clarifies their structure, eases understanding and can lead to further research within each category. The study presented the different ways to measure each of the attributes with the intention to provide a variety of methods for the manager to consider based on project and process specifics. Now the project team has to decide among the most effective of the metrics to use from a reference set as presented in this study.

An interesting aspect, resulting from this work, is that although measurements help informed decision making, a degree of subjectivity, expert judgment and analogy still plays important roles in the final decision relating to software test planning and test design. An organization can build an effective testing strategy that includes measurements in these two processes on the foundations of attributes identified in the study. Such an effort should lead to informed decision making about different software testing activities, and provide a baseline for measuring improvements.

10. Future work

During the course of this work, some related areas were found to be relevant to further investigation. One of these areas addresses the possible investigation into the measurable attributes of software test execution and software test review phases and the metric support available for those attributes. Another interesting area is to explore metric support at each level of testing including unit, integration and user acceptance levels. Further areas of work also include answers to questions like how to integrate the identified metrics into an organization-wide software metrics program, how to collect the metrics data in an optimal way and how valid and reliable the identified metrics are.

References

[1] P. Ammann and P. E. Black. A Specification-Based Coverage Metric to Evaluate Test Sets. In *HASE '99: The 4th IEEE International Symposium on High-Assurance Systems Engineering*, pages 239–248, Washington, DC, USA, 1999. IEEE Computer Society.

[2] J. Bach. Test Plan Evaluation Model. Professional Report, Satisfice, Inc., VA, USA, 1999.

[3] N. Bajaj, A. Tyagi, and R. Agarwal. Software Estimation: A Fuzzy Approach. *SIGSOFT Software Engineering Notes*, 31(3):1–5, 2006.

[4] V. R. Basili and R. W. Selby. Comparing the Effectiveness of Software Testing Strategies. *IEEE Transactions on Software Engineering*, 13(12):1278–1296, 1987.

[5] M. J. Bassman, F. McGarry, and R. Pajerski. Software Measurement Guidebook NASA-GB-001-94. Technical report, National Aeronautics and Space Administration, Goddard Space Flight Center, Maryland, WA, USA, 1994.

[6] R. M. Bell, E. J. Weyuker, and T. J. Ostrand. Predicting the Location and Number of Faults in Large Software Systems. *IEEE Transactions on Software Engineering*, 31(4):340–355, 2005.

[7] B. Berger. Evaluating Test Plans Using Rubrics. In *STAR West*, Anaheim, CA, USA, November 2004.

[8] R. Black. *Managing the Testing Process*. Microsoft Press, Redmond, WA, USA, 1999.

[9] R. Black. *Critical Testing Processes: Plan, Prepare, Perform, Perfect*. Addison-Wesley Longman Publishing Co., Inc., Reading, MA, USA, 2004.

[10] I. Burnstein, T. Suwannasart, and R. Carlson. Developing a Testing Maturity Model for Software Test Process Evaluation and Improvement. In *Proceedings of the IEEE International Test Conference on Test and Design Validity*, pages 581–589, Washington, DC, USA, 1996. IEEE Computer Society.

[11] Y. Chernak. Validating and Improving Test-Case Effectiveness. *IEEE Software*, 18(1):81–86, 2001.

[12] J.-F. Collard and I. Burnstein. *Practical Software Testing*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[13] R. D. Craig and S. P. Jaskiel. *Systematic Software Testing*. Artech House, Inc., Norwood, MA, USA, 2002.

[14] E. Dustin. *Effective Software Testing : 50 Specific Ways to Improve Your Testing*. Addison-Wesley Longman Publishing Co., Inc., 2003.

[15] S. Elbaum, A. G. Malishevsky, and G. Rothermel. Test Case Prioritization: A Family of Empirical Studies. *IEEE Transactions on Software Engineering*, 28(2):159–182, 2002.

[16] N. E. Fenton. *Software Metrics: A Rigorous Approach*. Chapman & Hall, Ltd., London, UK, UK, 1991.

[17] A. Gargantini and C. Heitmeyer. Using Model Checking to Generate Tests from Requirements Specifications. In *ESEC/FSE-7: Proceedings of the 7th European Software Engineering Conference Held Jointly with the 7th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 146–162, London, UK, 1999. Springer-Verlag.

[18] J. B. Goodenough and S. L. Gerhart. Toward a Theory of Test Data Selection. In *Proceedings of the International Conference on Reliable Software*, pages 493–510, New York, NY, USA, 1975. ACM Press.

[19] R. B. Grady. *Practical Software Metrics for project Management and Process Improvement*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1992.

[20] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy. Predicting Fault Incidence Using Software Change History. *IEEE Transactions on Software Engineering*, 26(7):653–661, 2000.

[21] J. Gray. Why do Computers Stop and What can be Done About It? *Tandem Technical Reports*, July 1985.

[22] L. Guo, Y. Ma, B. Cukic, and H. Singh. Robust Prediction of Fault-Proneness by Random Forests. In *ISSRE '04: Proceedings of the 15th International Symposium on Software Reliability Engineering*, pages 417–428, Washington, DC, USA, 2004. IEEE Computer Society.

[23] M. L. Hutcheson. *Software Testing Fundamentals: Methods and Metrics*. John Wiley & Sons, Inc., New York, NY, USA, 2003.

[24] K. Iberle and S. Bartlett. Estimating Tester to Developer Ratios (or Not). In *Proceedings of the 25th Annual Pacific Northwest Software Quality Conference*, Portland, OR, USA, October 2006.

[25] Institute of Electrical and Electronics Engineers. *IEEE Std 1059-1993 IEEE Guide for Software Verification and Validation Plans*, 1993.

[26] Institute of Electrical and Electronics Engineers. *IEEE Std 1061-1998 IEEE Guide for Software Quality Metrics Methodology*, 1998.

[27] Institute of Electrical and Electronics Engineers. *IEEE Std 829-1998 IEEE Guide for Software Test Documentation*, 1998.

[28] S. H. Kan. *Metrics and Models in Software Quality Engineering*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[29] C. Kaner. Measuring the Effectiveness of Software Testers. In *International Conference for Software Testing Analysis and Review (STAR East)*, Orlando, FL, USA, May 2006.

[30] C. Kaner and J. Bach. Paradigms of Black Box Testing (keynote address at Software Testing, Analysis & Review Conference (STAR) West). http://www.kaner.com/pdfs/swparadigm.pdf, November 1999.

[31] T. M. Khoshgoftaar, E. B. Allen, K. S. Kalaichelvan, and N. Goel. Early Quality Prediction: A Case Study in Telecommunications. *IEEE Software*, 13(1):65–71, 1996.

[32] L. M. Laird and M. C. Brennan. *Software Measurement and Estimation: A Practical Approach (Quantitative Software Engineering Series)*. Wiley-IEEE Computer Society Pr, 2006.

[33] N. Ohlsson and H. Alberg. Predicting Fault-Prone Software Modules in Telephone Switches. *IEEE Transactions on Software Engineering*, 22(12):886–894, 1996.

[34] R. Patton. *Software Testing (2nd Edition)*. Sams, Indianapolis, IN, USA, 2005.

[35] D. J. Paulish and A. D. Carleton. Case Studies of Software-Process-Improvement Measurement. *Computer*, 27(9):50–57, 1994.

[36] T. Pearse, T. Freeman, and P. Oman. Using Metrics to Manage the End-Game of a Software Project. In *METRICS '99: Proceedings of the 6th International Symposium on Software Metrics*, page 207, Washington, DC, USA, 1999. IEEE Computer Society.

[37] P. Piwowarski, M. Ohba, and J. Caruso. Coverage measurement Experience During Function Test. In *ICSE '93: Proceedings of the 15th International Conference on Software Engineering*, pages 287–301, Los Alamitos, CA, USA, 1993. IEEE Computer Society Press.

[38] R. S. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill Higher Education, 2001.

[39] T. Pyhälä and K. Heljanko. Specification Coverage Aided Test Selection. In *ACSD '03: Proceedings of the Third International Conference on Application of Concurrency to System Design*, page 187, Washington, DC, USA, 2003. IEEE Computer Society.

[40] S. R. Rakitin. *Software Verification and Validation for Practitioners and Managers*. Artech House, Inc., Norwood, MA, USA, 2nd edition, 2001.

[41] G. Rothermel and M. J. Harrold. Analyzing Regression Test Selection Techniques. *IEEE Transactions on Software Engineering*, 22(8):529–551, 1996.

[42] G. Rothermel, R. J. Untch, and C. Chu. Prioritizing Test Cases For Regression Testing. *IEEE Transactions on Software Engineering*, 27(10):929–948, 2001.

[43] P. J. Schroeder and B. Korel. Black-Box Test Reduction Using Input-Output Analysis. In *ISSTA '00: Proceedings of the 2000 ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 173–177, New York, NY, USA, 2000. ACM Press.

[44] J. Seo and B. Choi. Tailoring Test Process by Using the Component-Based Development Paradigm and the XML Technology. In *APSEC '00: Proceedings of the Seventh Asia-Pacific Software Engineering Conference*, page 356, Washington, DC, USA, 2000. IEEE Computer Society.

[45] J. Tian. *Software Quality Engineering: Testing, Quality Assurance, and Quantifiable Improvement*. Wiley-IEEE Computer Society Press, 2005.

[46] R. Torkar and S. Mankefors. A Comparative Study on Fault Finding Effectiveness in Common Black-Box Testing Techniques. In *Proceedings of the 3rd Conference on Software Engineering Research and Practice in Sweden*, Lund, Sweden, 2003.

[47] K. Vaidyanathan and K. S. Trivedi. Extended Classification of Software Faults Based on Aging, November 2001. Fast abstract at International Symposium on Software Reliability Engineering (ISSRE 2001).

[48] M. W. Whalen, A. Rajan, M. P. Heimdahl, and S. P. Miller. Coverage Metrics for Requirements-Based Testing. In *ISSTA '06: Proceedings of the 2006 International Symposium on Software Testing and Analysis*, pages 25–36, New York, NY, USA, 2006. ACM Press.

[49] H. Zhu, P. A. V. Hall, and J. H. R. May. Software Unit Test Coverage and Adequacy. *ACM Computing Surveys*, 29(4):366–427, 1997.