5-15-2012

A SOFTWARE TESTING ASSESSMENT TO MANAGE PROJECT TESTABILITY

Robin Poston
University of Memphis

Jignya Patel
University of Memphis

Jasbir Dhaliwal
University of Memphis

A SOFTWARE TESTING ASSESSMENT TO MANAGE PROJECT TESTABILITY

Poston, Robin, University of Memphis, 300 Fogelman College Administration Building, Memphis, Tennessee, USA, rposton@memphis.edu

Patel, Jignya, University of Memphis, 300 Fogelman College Administration Building, Memphis, Tennessee, USA, jmpatel@memphis.edu

Dhaliwal, Jasbir, University of Memphis, 300 Fogelman College Administration Building, Memphis, Tennessee, USA, jdhaliwl@memphis.edu

Abstract

The demand for testing services is, to a large extend a "derived demand" influenced directly by the manner in which prior developed activities are undertaken. The early stages of a structured software development life cycle (SDLC) project can often run behind schedule, shrinking the time available for performing adequate testing especially when software release deadlines have to be met. This situation fosters the need to influence pre-testing activities and manage the testing effort efficiently. Our research examines how to measure testability of a SDLC project before testing begins. It builds on the "design for testability" perspective by introducing a "manage for testability" perspective. Software testability focuses on whether the activities of the SDLC process are progressing in ways that enable the testing team to find software product defects if they exist. To address this challenge, we develop a software testing assessment. This assessment is designed to provide testing managers with information needed to: (1) influence pre-testing activities in ways that ultimately increase testing efficiency and effectiveness, and (2) plan testing resources to optimize efficient and effective testing. We developed specific software testing assessment measures through interviews with key informants. We present data collected for the measures for large-scale structured software development projects to illustrate the assessment's usefulness and application.

Keywords: Testability, Software Project, Design for Testability, Manage for Testability, Qualitative

1. Introduction

Large-scale structured software development can suffer from inadequate quality assurance and testing in software testing prior to its release. Inadequate quality can result from insufficient testing activities which are often relegated and compressed into the last stages of the software development life cycle (SDLC) limiting the time available for finding and fixing defects (Gelperin and Hetzel, 1988). With pre-set release deadlines, the early stages of planning, analysis, design, and development within a structured SDLC can often run behind schedule, shrinking the time allowed for performing adequate testing (Whittaker, 2000). One solution would be to better plan the testing process to be more efficient, while another would be to improve how activities in the earlier stages of the SDLC affect

downstream testing acitvities, e.g., by developing less-ambiguous, easier-to-test requirements during the analysis stage. Studies show that finding and fixing software quality problems earlier in the SDLC is less costly than during later stages of the SDLC (McGregor, 2007; Pressman, 1992). Given the need to start early and to manage the testing effort efficiently, this research explores how to assess how activities in the earlier stages of a project are progressing relative to their effect on the efficiency and effectiveness of the latter stage of testing.

Many activities in the early stages of the SDLC influence the amount and type of software testing performed at the end of the SDLC (Adrion et al., 1982; Cohen et al., 2004). For example, how well requirements are understood along with how well designs delineate interface connections, will both affect how the testing team verifies that the software is working properly (Li, 1990). While progressive software development teams include members of the testing team in requirements and design walkthroughs during the early stages of the SDLC (Singh and Shivani, 2009), a software testing assessment is lacking that assesses how the activities of the early stages of the SDLC are progressing relative to their influence on tasks performed during the testing stage. Armed with such measurements, testing managers could use assessment data to attempt to facilitate positive changes at various points in the SDLC or as early warning of the testing resources needed prior to the beginning of the testing stage.

Software testing assessment frameworks currently exist that inform software development teams on ways to both design software code to be more testable and provide the means of estimating testing effort (Binder, 1994; Voas and Miller, 1992). From a "design for testability" (DFT) perspective, software testability reflects whether code has been designed in such a way that the testing team will be able to find software product problems if they exist (Binder, 1994). A product problem is an existing defect which is an error, failure, flaw, or weakness in a program or system that produces an incorrect or unexpected result, or causes unintended behaviors (ISO, 1991). Software testability is a cumulative measure of the design attributes of a developing software product that reflect how easy it will be to assess if the product is working, i.e., the level of effort needed to perform adequate testing. The less testable a software product, the more testing effort will be needed to ensure its quality prior to its release. Proposed DFT assessments have focused on improving test cases (Bache and Mullerberg, 1990), class diagram interactions (Baudry et al., 2002), input and output states of the code, and state transitions of the program (Freedman, 1991). These assessments illustrate the importance of utilizing design and code methodologies to ensure more testable software products enter the testing phase. The DFT research addresses ways to manage testing efficiency and effectiveness at the software-product design level, with little attention given to ways to manage testing efficiency and effectiveness at the SDLC process level.

Our research extends the DFT perspective by introducing notions about how to influence the testing effort following a "manage for testability" (MFT) perspective. From a MFT perspective, software testability reflects whether the activities of the SDLC process are progressing in ways that are informing and supporting the testing team with the appropriate software project information to enable finding software product problems if they exist, both during the earlier SDLC stages as well as during the later testing stage. Following MFT, our proposed assessment focuses on the process and product characteristics of how the activities of the SDLC are progressing relative to their influence on tasks that will be performed during the testing stage, thus ultimately influencing testing efficiency and effectiveness. Projects with low (high) software testability assessment scores indicate that greater (less) testing effort will be needed. Along with important product characteristics, e.g., the ability to control business rule parameters, our proposed software testability assessment also focuses on process characteristics, e.g., the test team's understanding of the business requirements, system requirements, and interface designs, as well as measures of documentation completeness and test team involvement in walkthroughs and inspections. In support of the MFT perspective, prior research has acknowledged the need for assessing testability at the SDLC process level (Binder, 1994); however details of assessment criteria have not been offered. While the recognition of the need for MFT persists, little guidance exists as to how a software testing assessment can be developed to help testing managers

evaluate how activities performed throughout the SDLC influence a software project's testability and the testing effort that will ultimately be required.

The goal of this research is to develop a software testing assessment to manage project testability. The software testing assessment is designed to provide testing managers information they need: (1) to influence pre-testing activities in ways that ultimately increase testing efficiency and effectiveness, and (2) to plan testing resources that facilitate an efficient and effective testing phase. Thus, in our research, we move beyond the DFT research (e.g., Baudry et al., 2002; Freedman, 1991; Mouchawrab et al., 2005) to address how activities across the SDLC in large-scale structured projects influences testing activities. First, we reviewed the prior testability literature from a DFT perspective to understand the factors that affect testability and testing efforts in order to define an MFT perspective. We then developed specific software testing assessment measures through several rounds of interviews with key informants (i.e., testing managers at a global transportation company). We solicited the expertise of key informants specifically to identify the relevant activities of the SDLC impacting the amount and type of testing performed for adequate quality assurance. Our aim was to discover and define measures of testability for testing managers to use to influence how activities of the SDLC progressed and to better plan testing resources before the start of the testing stage. We next validated the testability measures with testing managers at a global aviation company and updated our assessment accordingly. Finally we collected data for the measures for large-scale structured software development projects at the original global transportation company, as well as, at a global business-to-business supply chain company. We conclude by discussing implications for practice and research.

2. Testability

To increase the chances of finding software problems, development and testing teams strive to improve the testability of the software (Mouchawrab et al., 2005). In general, software testability is a measure of the probability of finding a problem in the software if one exists (ISO, 1991), and as such it indicates the amount of testing effort needed to find errors. Attributes of the software product, e.g., observability of the code's operations, and attributes of the development process, e.g., how well testers understand business requirements, contribute to the probability of finding software problems. The tougher it is to find defects, the more effort is needed to provide adequate quality assurance through software testing (Binder, 1994). As a result, researchers seek better ways to design software programs for better testability (DFT), as well as manage the SDLC process to improve software testability (MFT) (Voas and Miller, 1992). Next we summarize the DFT literature, and then we build on the DFT perspective to examine the MFT perspective.

2.1 Measurements in Design for Testability

DFT is a strategy focused on aligning the design artifacts of the software development process to the product's testability, with the goal of maximizing testing effectiveness (Binder, 1994). Table 1 illustrates a summary of the selected literature on DFT. Researchers generally agree on several testability heuristics for software designers and programmers to consider:

- Controllability—the degree to which it will be possible to control the state of the product under test,
- Observability—the degree to which it will be possible to observe the workings of the product,
- Isolateability—the degree to which the component can be tested in isolation,
- Simplicity—the degree to which the product has a single, well-defined responsibility,
- Understandability—the degree to which the product is documented or self-explaining,
- Automatability—the degree to which it will be possible to automate testing of the product, and

- Heterogeneity—the degree to which the product involves diverse technologies necessitating diverse test methods and tools in parallel (Bach, 2003).

Author	Testability Definition	Study Context	Design for Testability
Lammermann et al., 2008	"...evolutionary testing can automate test case generation for a given test object… We term this quality evolutionary testability of a test object. …" (p. 1019)	Test case design	Propose testability measures: • Executable Lines of Code, • Halstead's Vocabulary, • Halstead's Length, • Cyclomatic Complexity, • Myers Interval, • Nesting Level Complexity, and • Number of Test Aims Tie measures to testing efforts
Baudry et al., 2002	"…design with an unreachable testing goal can be either improved or rejected as not testable" (p. 2)	Object-oriented software UML class diagrams integration design	Class interactions highlight: • Designs needing improvement, • Structural modifications, and • Constraints specifications To improve testability and testing efforts
Jungmayr, 2002	"...degree to which a software artifact facilitates test tasks in a given test context…" (p. 1)	Object-oriented software metrics for system dependencies and coupling	Define and use design and coding metrics: • Small number of dependencies has a large effect on testability • Coupling is not a good predictor of these dependencies
Bertolino and Strigini, 1996	"…probability that a test of the program on an input drawn from a specified probability distribution of the inputs is rejected, given a specified oracle and the program is faulty " (p. 9)	Measurement of testing confidence after software execution and testing is complete	Program correctness is based on: • Coverage of the testing oracle, • Ability of software to tolerate internal errors, • Relationship between execution profile and distribution failure inputs
McGregor and Srinivas, 1996	"Testability is the prediction of a method's ability to reveal faults in its implementation given a particular input distribution." (p.4)	Testability of a method in a class and indirect estimates on effort needed to test a class	Visibility into a class method • Accessibility of information that must be inspected to evaluate the correctness of method's execution How to define and use accessibility metrics
Voas and Miller, 1995	"…probability that a piece of software will fail on its next execution during testing…if the software includes a fault" (p. 19)	Design improvements in ability to verify software quality	Design, code, and test phase metrics used throughout the SDLC
Voas and Miller, 1992	"..is the tendency of code to reveal existing faults during random testing" (p. 1)	Testability design measurements	Testability measures using: • Formal specifications, • Design documents, and • Code itself
Freedman, 1991	"Domain testability refers to the ease of modifying a program so that it is observable and controllable" (p. 553)	Testability design measurements of observability and Controllability	Testability of programming structures: • Define new metric for program and functional specifications • Tie metrics to testing effort

Table 1. Selected literature on design for testability (DFT)

Using the testability heuristics, DFT researchers offer a variety of testability strategies. One strategy for achieving greater testability involves employing a code measurement system based on the evolution of code development and its relationship to automating test cases (Lammermann et al., 2008). Other strategies propose measurements for object-oriented software to improve object-oriented class interactions (Baudry et al., 2002) and system dependencies and coupling (Jungmayr, 2002) with the goal of increasing the chances of finding design and programming errors. Other strategies suggest measuring accessibility attributes (McGregor and Srinivas, 1996) and design measurements of observability and controllability (Freedman, 1991). Regardless of the strategy used, measuring design and code testability has been beneficial in offering insights that foster improvements in software programs during the design, code, and testing phases of the SDLC (Voas and Miller, 1992, 1995).

DFT researchers point out that testability strategies have limitations. Bertolino and Stringini (1996) illustrate that an over-reliance on increasing code-related testability may "produce a program which will be less trustworthy, even after successful testing" (p. 1). This suggests measures beyond testable code (e.g., that of human abilities) should also be considered. In our research, we build on the concepts of DFT to consider an MFT perspective. We recognize that testability must consider attributes of software products, and given the need to measure more than code testability, we also consider attributes of the SDLC process. We extend the notion of testability from a prior focus on primarily the code level to the project level in the SDLC.

2.2 Measurements in Manage for Testability

Testability studies define DFT at the source code or design level of software projects. We build on the suggestions of several DFT researchers to define and measure MFT (Binder, 1994; Voas and Miller, 1992). Binder (1994) uses fishbone diagrams to illustrate the myriad facets of the testing process which influence testability, and emphasizes that "testability cannot be considered apart from the [SDLC] process" (p. 88). However, the paper fails to define measures of the activities of the SDLC process prior to the start of testing that influence testability. Voas and Miller (1992) focus on random black-box testing DFT and suggest that repeated measures of testability are needed throughout the SDLC. However, they fail to define measures. Producing high-quality software is not only a function of creating high-quality software product designs, but also managing high-quality software development processes. Using the proposed testability assessment, we propose managers could assess how activities in the earlier stages of a project are progressing relative to their effect on the latter SDLC stage of testing.

Many activities of the SDLC have facets that affect the testability of software development products. The software testing assessment comprises a list of testability measures of project documentation, testing employees, the product being developed, etc., which are measured to develop a comprehensive score of a project's testability. Table 2 shows a list of the measures which assess a variety of SDLC-related activities that influence the project's testability. For example, in the planning stage, a testability measure is the quality of (i.e., number of problems found in) the original software in a modification project. Lower quality (i.e., more problems) in prior versions of the software would suggest greater challenges in finding problems if they exist as there could be more problems to find, which involves more testing work. As another example, in the analysis stage, a testability measure is the level of involvement that testing representatives have in document walkthroughs. Less involvement means the testing team has less input as well as potentially less understanding of the project and would suggest greater challenges in finding problems and more testing resources needed. In these examples, using a software testing assessment earlier in the SDLC would highlight which testability attributes are deficient and provide information to test managers to work with their SDLC counterparts on ways to improve the product or process before testing begins. The assessment would also offer testing managers early warning about the testing challenges to be expected and testing resources needed prior to the beginning of the testing stage.

3. Research Approach

To accomplish the research goals, we created a software testing assessment for testing managers to use in evaluating testability. Based on our review of the literature, we use the DFT perspective as the foundation for developing an MFT approach. Using the DFT literature as the base, we followed three main steps: interviewing key informants to define the appropriate testability measures to include in the assessment; gathering feedback from additional key informants to determine the clarity, comprehensiveness, and accuracy of the attributes; and collecting data from testing managers across multiple Fortune 500 level companies.

In the interviews, six managers, four testing leads and two testing audit managers, involved in software testing at a global transportation company were asked questions about the attributes of work performed in software development that affected the ability of the testing team to find problems in project artifacts and that influenced work activities performed in the testing stage. Each person had an in-depth understanding of software testing activities and challenges across the SDLC. Multiple interviews were held with each manager and continued until saturation was reached with no new measures surfacing. While the key informant pool represents a convenience sample, they were selected based on the recommendation of senior software testing executives and on the basis of their knowledge and expertise.

To further establish the validity of our assessment, we used triangulation as part of the feedback step. Triangulation is accomplished through the use of multiple data sources and multiple researchers (Mason, 2002). Iterative comparison, contrasting, and cross-examination of our work across multiple key informant interviewees allow us to ensure that the outcomes of this assessment are well developed. Two researchers conducted the interviews, with one researcher asking the questions and the other listening, taking notes, and asking follow-up questions. The presence of multiple researchers allows us to systematically recognize, discuss, and debate different interpretations and improve our understanding of the testability measures. To further improve the validity of the assessment we employed member-checking and peer-debriefing (Corbin, 2008). We presented drafts of our measures to the members of the testing community including the top testing management team (senior managing director and vice president at a global transportation company) (i.e., member-checking) as well as with other researchers and practitioners at a research workshop and a separate research colloquium to gather additional input (i.e., peer-debriefing). In addition, key informants from a global aviation company reviewed and commented on each testability measure highlighting wording issues, ambiguity problems, and missing content, which provided input for updating the assessment. All these steps serve to ensure the assessment and its results have greater credibility, and validity.

In the final data collection step, face-to-face meetings were held with managers in order to gather their assessments of current software development project using our testability measures. Each manager assessed one large-scale development project. A total of fifteen projects were assessed across five Fortune 500 level companies: five projects from the global transportation company, three from a global business-to-business supply chain company, three from a global retail company, two from a major utility company, and one each from a large non-profit healthcare company and worldwide manufacturer of engineering solutions. All data is from projects which were following a large-scale structured waterfall development methodology. See Appendix A for an overview of project data. The purpose of the data collection was to illustrate how the testability measures would be evaluated. The following sections describe the software testability assessment and its application in more detail.

4. Software Testing Assessment

The software testing assessment with 53 items is provided in Table 2. Measures were developed for the following information technology components: software, hardware, documentation, security, data, and facilities. Within these components each area was further broken down into

testability facets. When using the assessment for development projects, testing and/or project managers were asked to rate each testability attribute for their project on a scale of 1 to 7, with 7 being highest in testability. For example, if error messages provide clear descriptions of the problem, the associated attribute would be rated 7, meaning this activity provides insightful information about errors which facilitates the ability of the testing team to find and fix software problems if they exist.

Testability Facets	**Testability Measures**
Software	
Critical applications	Quality of original software before testing starts - specifically, unit test results along with build and known issues are available
	Quality of original software before testing starts - specifically, first cycle of integration (end-to-end) testing results are good
Where and how applications are executed	Visibility to data mapping to input and output of interfacing systems
	Ability to control business rule parameters (e.g., modify data retention periods)
Are patches are up-to-date?	All patches been applied within the test environment before the start of testing
Input and output controls	Data dependencies are documented
	Changes that affect other systems are documented
Error messages	Error messages provide clear description of the problem
	Error handling processes are efficient
	Ability to perform fail-over and recovery testing
Hardware	
System fileservers: fileserver integrity	All fileservers are operational
Documentation	
System components	
Business Requirements (BRS)	Level of involvement of testing representative(s) in the document walkthrough
	Understanding of BRS by testing team members
	Comprehensive assumptions and constraints have been included
	Detail business scenarios and examples have been included
	High level specifications for de-coupling have been included
	Stakeholder review and approvals exist
	Version control in place and followed
	Open issues are tracked and addressed
System Requirements (SRS)	Level of involvement of testing representative(s) in the document walkthrough
	Understanding of SRS by testing team members
	Comprehensive assumptions and constraints have been included
	Detail scenarios and examples have been included
	Traceability to BRS has been documented
	Stakeholder review and approvals exist
	Version control in place and followed
	Open issues are tracked and addressed
System Architecture Specification (SAS)	Completed and provided with entire system flow
	Visibility of all interface changes
	Defined data mapping between systems
De-coupling/ Back-out Plan	Document is complete and provided
	Ability to decouple specific functions within a project
	Degree of ability to decouple the code between interfacing systems /domains (more data/switch driven less code driven)
Detail Test Plan Specification (DTPS)	Stakeholder review and approvals exist
	Understanding of DTPS by testing team members
	Version control in place and followed
	Known location of organized repository of project files
	Mitigation and contingency plan known risks
	Well defined test strategy
	Well defined test cases
	Well defined test data plan
Log files: Defect log files	All defects and their remedies are logged in an easily accessible manner by the testing group
Overall	Defined process for tracking and resolving testing issues/concerns/queries
Security	
Access controls	Access rights to all impacted systems have been set up before the start of testing
	Access rights have been completely defined before the start of testing

Internal controls on key applications	Ability to test software compliance (e.g., HIPPA, SOX, PCI)
Data	
Data security policies: Is there any formal written data security policy?	Test data is locked down and secure
	Production data is efficiently cleansed of sensitive information
Data files / database access	Updates and database files are accurate and available
Data encryption	Ability to simulate sensitive data
	Ability to simulate encrypted data
	Level of complexity in decrypting encrypted data
Facilities	
Test environment	Separate testing environment from the remaining software development team

Table 2. Software testing assessment

To illustrate how the testability attributes were evaluated and their usefulness in designing and managing for testability, we collected data for fifteen large-scale software development projects. We asked respondents to consider assessing a project in the testing or release stage of the SDLC in order to encourage participants to consider how each measure influenced the ability to find defects if they existed. Limiting our data collection to projects in the final stages of development helped us continue to validate the newly created testability measures. When specific testability measures are irrelevant, we ask respondents to enter 'n/a' for that attribute. For comparison across projects, we removed the effects of the irrelevant attributes by calculating the percentage of the total possible score for each project. Table 3 summarizes the percentage of the total possible score for all projects, illustrating eight of the fifteen projects were at or below a 70% score suggesting just that over half of the projects included in this effort would be considered 'significantly challenged' based on the testability measured.

Project Name	Total Score	Total Possible	Testability Score (Total Score/ Possible)
Ink and Toner Saver	183	224	82%
Lab data management	228	287	79%
ePrint	192	245	78%
Management GUI	250.5	336	75%
JRB Conversion	238	323	74%
New service introduction	232	315	74%
I Roads	266	371	72%
International Returns	219	315	70%
Pricing enhancements	200	315	63%
Vendor Conversion	169	294	57%
DSO Process Improvement	206	364	57%
Global Tax Engine	197	357	55%
ILS	155	315	49%
Event Report	161	371	43%
Plant Metric Dashboard	131	350	37%

Table 3. Summary of project software testing assessments

5. Discussion

In this research, we started with an understanding of the DFT perspective, and then developed testability measures and integrated them into a software testing assessment grounded in an MFT perspective. The attributes were created based on input from expert informants and cross-validated with additional testing professionals and academic peers. Data was gathered on fifteen software projects to assess the project testability and illustrate the assessment's usefulness. Testability data scores ranged from 37% to 82%, averaging 64%, which illustrates all projects contained some testability issues and some projects are heavily challenged in MFT. Based on the findings, facets of

DFT and MFT should combine to create a comprehensive assessment of testability. This study illustrates that not only are design issues important, but SDLC process issues also have the potential to influence how the test team finds defects in the software project if problems exist. Future research is needed to determine the means and mechanisms by which different measures of the software testing assessment influence different types of testing outcomes, e.g., quality of test cases, and within different phases of SDLC.

The findings must be assessed in light of the study's limitations. For this study, the increased application afforded by interviewing key informants must be traded off against the inherent limitations of the approach, primarily that of measurement validity. The use of key informants and the amount and type of data collected all limit the validity of our results. Key informants from one organization were identified based on their knowledge and expertise in running software testing projects. To mitigate the potential bias of having input from only one organizational perspective, we used an approach based on triangulation involving multiple researchers, presented the testability measures in member-checking and peer-debriefing sessions, obtained feedback from key informants at a different organization, and obtained input from managers completing the assessment for real projects. In one company, we shared the testability scores of the projects with executives of the testing management team (the software quality managing directors and vice president) to gain their feedback. We asked how well the testability scores reflected their knowledge of the testing challenges encountered with each project. The testing executives confirmed that the order from highest to lowest testability scores did reflect the relative amount of challenges and testing effort incurred within each of the projects. This feedback supports the validity of the testability assessment. Using input from key informants and testing executives to create a software testing assessment based on industry best practices enhances face validity and content validity, however, we cannot adequately assess the predictive, convergent, and discriminant validity of the measures. Future research should validate the testability measures using rigorous statistical analysis across multiple organizational contexts and development methods.

Also, this research illustrated how the software testing assessment could be used based on self-reported measures with one respondent assessing one project in the final stages of development. This improves homogeneity of responses for comparability and the ability to gather confirmation through feedback that the measures are valid. However, future research should consider collecting additional data with projects assessed at different points across the SDLC and with different and multiple SDLC stakeholder viewpoints. Collecting additional data would allow researchers to use factor analytical methodologies to determine if common constructs emerge to form a nomological network of factors that determine which testability measures are most relevant to which stages of the SDLC.

The findings of this research offer several important implications for research. Prior research has focused primarily on methods for designing better software for testability (DFT) and has maintained a more granular design and code level view. This research builds on the DFT perspective and suggestions of several DFT researchers to define and measure testability across the SDLC (Binder, 1994; Voas and Miller, 1992). Producing high-quality software is a function of creating high-quality software product designs and code and also managing high-quality software development processes. Future research should utilize this software testing assessment to assess how activities in the earlier stages of a structured development project are progressing relative to their effect on the latter SDLC stage of testing to empirically delineate the factors that influence testing outcomes. While the focus of this study was on waterfall development approaches, this assessment should also be used to assess how testability occurs in projects following more agile approaches.

Future research can also make use of case study methodologies, e.g., action research, to examine the cultural implications of adopting the assessment in companies to measure diffusion and individuals' reactions to the assessment's usefulness. This would give a deeper understanding of how the assessment both affects and is affected by project stakeholders thus educating practitioners on the optimum ways to use the assessment. Another implication for research is to measure the costs and benefits of using the assessment to examine whether the cost of its use justifies the improvements to testing stage activities.

The findings also have several important implications for practice. Software testing teams can use the software testability measures as a benchmark-type tool to determine whether projects are more or less testable. A database of projects can be gathered and used to determine patterns of the factors that drive testability. Factors could include project size, project manager style, the use of offsourcing, criticality of the software to the user base, etc. As benchmark data builds, best practices in software testability can be derived and shared with future projects assessed to determine if improvements have been made. Comparing measures across and within SDLC stages may provide useful insights as well. Through statistical analysis of the data, assessments can be made as to which measures drive testability and which testability criteria are most critical to the testability of software projects.

6. Summary and Conclusion

Testing managers lack the means to systematically assess how the activities of the SDLC are progressing in their relationship to a software product's testability, which ultimately impacts the ability to find software defects if they exist and the amount of testing effort required in the testing stage. To address this, we propose managers utilize the software testing assessment. We provide testability measures which could be used as a useful audit tool or a checklist for project managers to determine the level of testability in their projects. Assessing development projects before testing begins can help development teams build testability into their projects and testing managers can gain forewarning of issues prior to the beginning of the testing stage. Knowing when problems are coming ahead of time and where testability weaknesses are allows testing managers to better allocate limited resources in ways that improve testing processes. This also gives the testing management team ways to open discussions with SDLC stakeholders about areas of improvement.

As shown by the findings of Table 3, issues that affect project testability are pervasive as all fifteen projects scored below 85%, with eight projects scoring at or below 70%. Armed with such assessment data, testing managers can use the attributes and their scores for initiating discussions among SDLC stakeholders to find ways to improve the development process and testing performance. The software testing assessment proposed by this research offers researchers and practitioners a means for uncovering and gaining an understanding of socio-technical challenges in SDLC projects that inhibit the ability to meet the goals of delivering high-quality software solutions faster and less expensively.

References

Adrion W., Cherniavsky J., Branstad M. (1982). Validation, Verification, and Testing of Computer Software. Computer Surveys, 14(2), 159-192.

Bach J., (2003). Heuristics of Testability. Downloaded from: www.satifice.com/tools/testable.pdf.

Bache R., and Mullerburg M. (1990). Measures of testability as a basis for quality assurance. Software Engineering Journal, 5(2), 86-92.

Baudry B., Sunye G., and Traon Y. (2002). Testability Analysis of a UML Class Diagram. IEEE symposium on Software Metrics.

Bertolono A., and Strigini L. (1996). On the Use of Testability Measures for Dependability Measures. IEEE Transactions on Software Engineering, 22(2), 97-109.

Binder R. (1994). Design for Testability in Object Orientated Systems. Communication of the ACM, 37(9), 87-101.

Cohen C., Birkin S., Garfield M., and Webb H. (2004). Managing Conflict in Software Testing. Communication of the ACM, 47(1), 76-81.

Corbin J., and Strauss A. (2008). Basics of Qualitative Research. Sage Publications. California

Freedman R. (1991). Testability of Software Components. IEEE Transactions on Software Engineering, 17(6), 553-564.

Gelperin D., and Hetzel B. (1988). The growth of Software Testing. Communication of the ACM, 31(6), 687-695.
ISO. (1991). International Standard ISO/IEC 9126. Information technology: Software product evaluation: Quality characteristics and attributes for their use.
Jungmayr S. (2002). Testability Measurement and Software Dependencies. International Workshop on Software Measurement.
Lammermann F., Baresel A., and Wegener J. (2008). Evaluating evolutionary testability for structure-oriented testing with Software Measurements. Applied soft Computing, 8, 1018-1027.
Li E. (1990). Software Testing in a System Development Process: A Life Cycle Perspective. Journal of Systems Management, 41(8), 23-31.
Mason J. (2002). Qualitative Researching. Sage Publications. London.
McGregor J. (2007). Test early, test often. Journal of Object Technology, 6(4), 7-14.
McGregor J., and Srinivas S. (1996). A Measure of the Testing Effort. USENIX. Toronto: Conference on Object-Oriented Technologies.
Mouchawrab S., Briand L., and Labiche Y. (2005). A Measurement Framework for Object-Oriented Software Testability. Ottawa, Canada: Software Quality Engineering Laboratory, Carleton University.
Pressman R. (1992). Software Engineering: A Practitioner's Approach, 3rd Edition. McGraw Hill. New York.
Singh Y., and Shivani G. (2009). Role of Testing in Phases of SDLC and Quality. International Journal of Information Technology and Knowledge Management, 2(2), 343-346.
Voas J., and Miller K. (1992). Improving the Software Development Process using Testability Research. Software Reliability Engineering.
Voas J., and Miller K. (1995). Software Testability: The new Verification. IEEE Software, 12(3), 17-28.
Whittaker J. (2000). What is Software Testing? And why is it so hard? IEEE Software, 17(1), 70-79.

Appendix A: Project Data

Project name	Pricing enhance-ment	Mgmt GUI	New service intro.	Lab data mgmt.	I Roads	Ink & Toner Saver	International Returns	ePrint	Event Rpt	DSO Process Improvement	Global Tax Engine	Vendor Convert.	ILS	JRB Convert.	Plant Metric Dash-board
Respon-dent Title	Testing Manager	Testing Manager	Testing Manager	Project Manager	Testing Manager	IT Manager	Testing Manager	Project Manager	Testing Manager	Business Analyst	Busi-ness Analyst	Testing Manager	Testing Manager	Devel-oper	Data Analyst
SDLC Stage	Release	Release	Release	Release	Testing	Release	Release	Release	Testing	Testing	Testing	Testing	Testing	Testing	Testing
Number of interfaces with other systems	15+	25+	100+	3	5	0	25+	0	0	6	3	4	5	4	2
Number of test cases	1,000+	2,109	20,000	250	1000+	2400	3000+	0	0	750	25	200	110	n/a	0
Project manager's years of experience with testing	8	18	20+	20	11	20	8	25	0	6	5	0	11	10	0
Project manager's years of experience with the company	4	13	20+	15	20	25	4	25	1	14	2	4	20	7	9
Number of staff hours to code project	12,000+	2,500	100,000	100,000 +	100,000+	100,000 +	100,000+	100,000 +	30	1800	300	500	100,000+	320	900
Software new or modifi-cation of existing code	Modificati on	Brand new	Both	Brand new	Modificatio n	Brand new	Brand new	Brand new	Modificatio n	Modification	Brand new	Modificatio n	Modificatio n	Brand new	Brand new
Org. Industry	Trans-portation	Trans-portatio n	Trans-portatio n	Trans-portation	Trans-portation	B2B Supply Chain	B2B Supply Chain	B2B Supply Chain	Nonprofit healthcare	Retail	Retail	Retail	Utility	Utility	Manu-facturin g
Total Score	200	250.5	232	228	266	183	219	192	161	206	197	169	155	238	131
Total Possible	315	336	315	287	371	224	315	245	371	364	357	294	315	323	350
Score / Possible	0.63	0.75	0.74	0.79	0.72	0.82	0.7	0.78	0.43	0.57	0.55	0.57	0.49	0.74	0.37