International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2013): 4.438

Object Oriented Software Testability Survey at Designing and Implementation Phase

Dr. Pushpa R. Suri[1], Harsha Singhani[2]

[1]Department of Computer Science and Applications, Kurukshetra University, Kurukshetra -136119, Haryana, India

[2] Institute of Information Technology & Management (GGSIPU), Janak Puri, New Delhi -110058, India

Abstract: ***Software testability is coming out to be most frequent talked about subject then the underrated and unpopular quality factor it used to be in past few years. The correct and timely assessment of testability can lead to improvisation of software testing process. Though many researchers and quality controllers have proved its importance, but still the research has not gained much momentum in emphasizing the need of making testability analysis necessary during all software development phases. In this paper we investigate and review the factors, issues and methods of testability estimation of object oriented software systems during various phases of development life cycle. The paper hopes to change some common prejudices about testability. Improving software testability is key objective of our research by high lighting and relating the various factors individually affecting testability.***

Keywords: Software Testability, Testability Factors, Testability Metrics

1. Introduction

The scenario of current software industry is changing from structural to object oriented approach, which has actually eased the software development process in making a clear understanding of the requirements of the real world problems, with modular software structure. But Testing of object oriented software has presented numerous challenges due to its features. A tester often needs to spend significant time in developing a lengthy testing code to ensure that system under test is tested as per given requirement. But lack of time, efficient cost, less manpower, limited resources and all other unavoidable constraints, leads to short term test plan, leading to compromised quality software's. Software testability is introduced to measure the degree of difficulty of software test or the possibility of the software defects that can be found out. It is actually an external software attribute that evaluates the complexity and the effort required for software testing. Testable software is the one that can be tested easily, systematically and without following any adhoc measures.

The study on software testability primarily comes into view in 1975. It is accepted in McCall and Boehm software quality model, which build the foundation of ISO 9126 software quality model. Since 1990s, software engineering society began to initiate quantitative research on software testability. Formally, Software testability has been defined and described in literature from different point of views. Out of many definitions of Testability, [1] defines it as the degree to which a system or component facilitates the establishment of test criteria and performance of tests to determine whether those criteria have been met. [2] defines it as attributes of software that bear on the effort needed to validate the software product. These two standard definitions aim to different targets and both are qualitative without any operational guidelines. The testability thus has become a quality factor contributing to system maintainability as ISO standards whose measurement and can be used to predict the amount of effort required for testing and help managing the required resources effectively.

So, the study of testability actually helps in finalizing software design and coding changes for making it test friendly, thus reducing system test cost along with improved software quality. Through the test effort reduction using these new researches, not only the software design and code is improved but also the new levels of software reliability and credibility can be reached. All this leads to desire of designing and developing highly testable systems. But the desire does not end here, there is a need to measure and verify the testability, quality and reliability of the system which is where the challenges are faced. Lot of testability related issues such as software design complexity relation to testability, class contribution to testability, object oriented features of a class, object oriented metrics contribution in testability estimation are being kept in focus in these research works.

2. Software Testability Role in Object Oriented Systems

As already known object oriented system development has become leading approach within software industry. In comparison to structural program designing and coding, the testing and hence the testability of systems is quite complex in object oriented systems. The main properties of object-oriented technology such as Data abstraction and Encapsulation, Inheritance, Polymorphism and Dynamic binding are mainly responsible for the success of this approach. But some of these factors such as Inheritance lead to increased complexity and thus having a negative effect on system testing and testability. Specifically with respect to object oriented software the previous research shows that multiple inheritance decrease the level of testability of software [3]. The level of difficulty also increases with multiple units of code , inherent dependencies and interactions between the classes[4]. The failure within the code is not easily traceable.

Software testability is an external software attribute that evaluates the complexity and the effort required for software testing. For any system which needs to be made test friendly,

the testability measures needs to be applied from the beginning itself during design phase and later should be applied further at coding and testing phase. As rightly pointed out by [5] designing for testability becomes a way of thinking, If we think of tests as a user of your system. So, it is better to keep TDD (test driven development) approach where the tests come first and largely determine the API design of the system, forcing it to be something that the tests can work with. The same is often stated as Design For Testability (DFT) also, which means building a system keeping testability measures in line at deigning as well as coding phase so that tracing errors is easier along with reduced testing effort[6]. It is basically a systematic way of development which maximises the effective testing efforts. With the course of time the lot of research has been done on testability issues during design and coding phase. At design phase generally an integrated approach is used at analysis and designing phase of software development, to improvise the object oriented software design in the beginning itself as supported by many practitioners discussed below in section 3.

But testability analysis does not end at initial stage, to develop a system more efficiently and test friendly, code time testability is adopted. It works for systems which do not give effective results with design time testability efforts. It is a method to enhance runtime testability analysis of a system prior to testing. It is the secure coding mechanism which reduces chances of failure at maximum possible extent along with generating test logs for any further improvisation. It also leads to test case reduction as per researchers [7]. Hence code time testability measures not only raise the testability standards but reduce the testing effort along with making system more reliable.

Another testability approach which has not been explored much but yet needs to be looked upon is testability improvisation during debugging and testing phase of software development. The work done so far in this field is very less and not that significant. Few researchers have only proposed some software reliability growth models for testability improvisation during this phase[8]. The proposed testability measure can result in higher fault detection and can also be used for the determination of modules that are more vulnerable to hidden faults. However, the quantification of testability measurement using reliability growth models are still needs to be explored further. Then there is IVF testability model which is used during software test process mainly to estimate software test work load and increase system reliability[9].

3. Literature Review

Testability is not an intrinsic property of a software artefact and cannot be measured directly as other software attributes. Instead testability is an extrinsic property which results from interdependency of the software to be tested and the test goals, test methods used, and test resources [6]. A lower degree of testability results in increased test effort and high development cost. In extreme cases a lack of testability may hinder testing parts of the software or software requirements at all. Measuring testability is a challenging and most crucial task towards estimating testing efforts. Several approaches like model based testability measurement, program based testability and dependability testability assessment has been proposed. Also a number of metrics on testability measurement have been proposed, some at design and analysis phase or some at source code level. The brief overview of work done so far at two major stages of software development life cycle i.e. System design and analysis stage and system code and implementation phase is listed below:

3.1. Relevant Work Done in System Design & Analysis Phase

Design time testability analysis provides a direction and guidance for testing at early stage in object oriented systems which may yield the maximum outcome by reducing testing effort later and hence improvising system testability. The focus is on capturing the test reduction patterns and issues at early stage in design diagrams specifically in UML class diagrams etc. as discussed by many researchers mentioned below.

The testability analysis thus at this stage help developers implement changes in the system design before entering implementation phase. The main purpose is to reduce the system development cost, time and errors by avoiding these design based discrepancies to be carried further, by locating the faults using testability estimation techniques at design time. Different models were proposed on the basis design time testability issues. The design issues of object oriented programming such as Inheritance, Polymorphism, Coupling, Cohesion, Encapsulation, Information hiding , Class Size and Complexity were the focus of interest for testability improvisation. Some of these factors were quantified also and thus resulted in Testability Metrics, which are easiest to implement and analyse. Many of these proposed Testability metrics were based on previously object oriented design metrics suite of [10], [11]. The quantification at design phase in object oriented systems was mainly done using various UML diagrams.

The work done so far in this field has not been in any one particular direction but rather has been exploratory in nature, which is yet to found acceptance amongst practitioners. The designing phase testability research has started taking shape in past few years only. Previously source code testability analysis was more highlighted. The important of all these research in last few years in design and analysis phase are hereby listed below in reverse chronological order in Table1.

Table 1: Testability Research at Analysis & Design Phase

S. NO.	*Pub. Year*	*Author & Citation*	*Research Summary*	*Limitations and Gaps*
1.	2010 - 2013	Nazir et.al. [12]–[14]	Contribution is in the form quantification of Testability Metrics, which has been calculated using two major factors Understandability and Complexity using CK Metrics suite.	Though suggestive testability metrics is verified but its empirical study with very large scale software systems is yet to be done.
2.	2010	Khalid et.al. [15]	Measured design phase complexity and testability with quantifiable results using five design metrics.	Not applied on self descriptive and large scale systems designs.
3.	2009	Khan and Mustafa [16]	The Research basically proposed a model named MTMOOD for the assessment of testability in object-oriented design. This model had then been validated using structural and functional information from object oriented software. This quantitative measurement of testability is useful and practical to determine on what module to focus during testing.	The model takes very restrictive view of the OO programming concept. So, less sufficient for Self-Descriptive Systems.
4.	2007	E. Mulo [17]	The main focus though was on designing software's for testability but the study stressed on testability estimation though out SDLC cycle.	The suggestive measurement estimation cost can go up very high along with required added effort may be.
5.	2005	Mouchwrab [3]	Research revolves around UML diagrams at design level and measurement of their testability as per proposed framework. Claims to reduce testing cost.	Research work only gives the starting point of UML designs but not yet empirically proved.
6.	2004	Shih [18]	Basically the research presented a model that is based on Gao's pentagram model and analytical approach for testability measurement of components. The testability review process at design and analysis phase was performed keeping five major characteristics of testability, which were used in testability metrics calculation and obtaining five testing points to draw the final pentagram.	The study was validated on large scale industrial software's. Though it was validated well on few case studies but still model did not gain much popularity due to complex notations.
7.	2002	Jungmayr [19], [20]	Suggestive Model relates testability to dependencies between components (e.g. classes) and investigates testability measurement based on static dependencies within OO systems keeping integration testing point of view.	More tests are required to exercise their interfaces. Other important factors such as observability and controllability except from dependencies are overlooked in his research.
8.	2001	Baudry et. al. [21]	Identified Class diagrams and state charts of UML for testability analysis. Focused mainly on complex interactions within design which cause problems in testing the software also called testability anti patterns, identified by using a class dependency graph (CDG).	Assumes that multiple paths between classes are redundant, from a semantic viewpoint that is expensive to test
9.	1998	Lo & Shi [22]	The researchers proposed OO design Testability Factors at structural, communication and Inheritance levels. These factors values contributed in testability estimation.	Yet to find the usefulness of individual testability factor metrics and correctness of proposed model. Moreover some metrics still needs refinement w.r.t. industrial standards.
10.	1994	Binder [6]	Identifies Testability Fish bone Model and six high-level factors affecting testability: built-in test, test suite, test support environment, implementation characteristics, and representation characteristics at initial phase. Also suggests set of metrics to be used for design level testability measurement.	All factors are related to higher level of abstraction having not in depth relation with object oriented design constructs. Also, all suggested metrics are not empirically proved.
11.	1991	Freedman [23]	System is made observable and controllable with domain testability at design level. Along with that it also defines that a software artefacts that is easily testable has the desirable quality attributes: test sets are non-redundant, test sets are small, test outputs are easily interpreted and software faults are easily findable.	Test Input-Output Inconsistency demonstration missing.

3.2 Relevant Work Done in Coding & Implementation Phase

The analysis at code level is more complex than at the design level, but it gives more detailed measures about testability. More- over, code testability analysis can help to identify the low testable parts in software re-engineering. So , it is important for those dynamic object oriented systems where conventional design time testability analysis will not be effective. So code time testability analysis often called runtime testability is done at before final test cases are executed. This is required, as to ensure that a good design which may not have improved system testability due to poor coding practices should not lead system towards failure along with increasing effort of testing. So, some of the testability improvisation techniques listed below in Table 2 are adapted during system coding and testing phase so as to raise the testability performance of the system by reducing the system testing effort along with development cost reduction and making system more reliable.

Table 2: Testability Research at Coding & Implementation Phase

S. NO.	*Pub. Year*	*Author & Citation*	*Research Summary*	*Observations, Limitations & Gaps*
1.	2011	Badri et. al. [7], [24]	The basic purpose their study was to establish a model around source code metrics for testability assessment. The proposed testability MTMOOP model was based on inheritance, coupling and encapsulation metrics at source code level. The positive correlation was established between the proposed metrics and five test class metrics suing commercial Java software's. They also established strong correlation between various popular object oriented and test class metrics, in terms of effort reduction, hence testability improvisation.	The proposed metrics was based on limited set of assertions and needs further extension on other object oriented characteristics too. The systems used for study were limited to one language only, so may need extension in other commercial programming languages.
2.	2011	Boyle & Moghadan [25]	The research was interesting & surrounded around refactoring of source code for testability improvisation using LSCC cohesion metrics.	The approach was suggestively new but could not be carried further as it was inconclusive, time consuming & needed extra effort.
3.	2011	Harman et. al. [26], [27]	The main research was focussed on program transformation for Testability Improvisation using Refactoring as one of the method, hence adding input to new field of testability research called Testability Transformation.	The study of course highlights a critical test improvisation method based on test case equivalence lays the field for future study needs further exploration as the proposed roadmap is yet to be argued upon in context with advance testing techniques.
4.	2011	Khatri [8]	The research was based on approach for improving testability using software reliability growth models for fault detection and determination of modules that are more vulnerable to hidden faults. The concept that prior knowledge of proportion of fault of different complexity lying dormant in the software can ease the process of revealing faults has been demonstrated in their work.	The proposed model is purely theoretical and does not give any quantitative measure of improvement of testability.
5.	2008, 2010	Singh & Saha [28], [29]	The basic study was focused on software contracts and testability. The quality of software contracts to decrease the testing effort help in testability improvisation. The flow graph of class under study, with and without contract was used to demonstrate the test case reduction.	The proposed method is studied at class level which needs to be extended further at higher level system testing.
6.	2010	Ding [9]	They proposed an IVF (Iteration of Vector Factorization) software testability model based on statistics and analysis of test data of the Software Test and Evolution Centre and validates it with practice. The work was based on Gao pentagon model. The proposed IVF model was validated with 20 software system, by calculating software's testability value and unit average testing effort.	The proposed model is compact and practical and can e directly applied. In spite of that the model has not gained much popularity, as vector factorization method is not easily understandable and applicable in all object oriented systems.
7.	2009	González [30]	They proposed a Run time testability measurement (RTM) technique, which was based on the idea of the amount of runtime testing limitation by the characteristics of the system, its components, and the test cases themselves. This approach is well suited for usage in an interactive tool, enabling system engineers to receive real-time feedback about the system they are integrating and testing at runtime.	The evaluation of accuracy of the predicted values and of the effect of runtime testability on the system's reliability is left for later study[1]. More validation using industrial cases and synthetic systems has not been done using proposed model. The approach may not be useful for basic object oriented systems.
8.	2008	Zheng & Bundell [31]	Their research was based on contract-based testing having Contract for Testability as the principal goal in line with the Design by Contract (DbC) principle. The DbC concept was extended to the software component testing (SCT) domain, in developing a new TbC technique and applied it to UML-based component integration testing (CIT) with a case study.	The main idea behind TbC technique was to improve component testability through improved traceability, observability and controllability. The technique needs further evaluation and validation with industrial data
9.	2005	Nguyen et. al. [32]	In this research code testability analysis was done using source code data flow diagram. These graphs are converted to ITG (Information transfer graphs) and further to ITN (Information transfer nets) to be used with SATAN tool. The algorithm to automatically translate the SSA form into a testability model is verified with a case study.	The analysis method used at code level is very complex. The study uses complex graphs, whereas there are simplest measures available in accordance with the axioms, except that they can't take into account information on the data by the flows.
10.	2003, 2004 & 2006	Bruntink et. al. [33]–[35]	The study was basically to demonstrate correlation between Class level metrics (FOUT, LOCC, RFC etc.) and test level metrics (dLOCC, dNOTC) keeping the main focus on issue of testability using open source commercial java systems cases.	The study needs to be further extended with more empirical data. Further the study was only from unit testing perspective using CK metrics, which may further be analysed from using other

				metrics and at functional or Integration testing level.
11.	1997	Wang [36]	Research was done to provide Testable OO software (TOOS), with built-in testability. TOOP approach for software development was proposed and the testability of OOS at BCS level (CBCS), object level (OTA) and system level (STA) are quantitatively modelled. This built-in testable mechanism in objects improves the testability of OO software in terms of test controlability and observability, which can be inherited and reused further.	The approach has not been empirically tested and verified w.r.t. industrial data.
12.	1997	Lin [37]	The purpose of their research was to analyse testability of software by tracing the source code instead of testing it. The proposed model was refined over Voas PIE model with respect to a particular input distribution. The test data conducted using a program showed new PIE values closer but accurate than that of Voas model.	The technique needs to be further explored as it is found to be time consuming though useful in testability analysis without any formal testing. The direct correspondence with object oriented software development approach was not empirically established.
13.	1996	McGregor & Srinivas [38]	The main focus of the research was to analyse testability in terms of visibility component of method for, which actually measures the accessibility of the information that must be inspected to evaluate the correctness of the execution of a method. Testability metrics in terms of visibility component was conceptualised here.	The study was quite useful for structural programming software though not sufficient for object oriented programming systems.
14.	1992-94	Voas & Miller [39], [40] & [41], [42]	Basic works surrounded around system code and hidden fault location through testability implementation. A new technique called PIE (Propagation, Infection and Execution) based on the software failure model was proposed. This technique measures testability of each statement in software by a dynamic analysis, i.e. while running the software.	The technique was formulated and tested on structural programs. But the technique added to complexity of the overall test procedure and hence not adopted much in industry. Though lot of variation in the basic PIE model has been proposed in later years.

4. Important Observations

4.1 Testability Constructs & Factors

During the study of testability at different stages of software development life cycle in object oriented stems, lot of important factors affecting the testability were observed. It is important to throw some light on these factors, antipatterns and improvisation techniques as well. The major constructs contributing to testability are:

- **Key Object Oriented Features:** Object oriented software characteristics are mandatory to be recognized and after that the set of testability factors suitable at the design phase should be finalized. All these major object oriented features such as Class Size, Coupling, Cohesion, Encapsulation, Inheritance and Polymorphism contribute to testability as shown by researchers especially at design time. These features contribute to the key quality factors affecting testability hence need to be looked in detail from all perspectives.
- **Object Oriented Metrics:** Many of these features are incorporated in the form of popular object oriented metrics such as LOC, NOC, LCOM, CBO, RFC, DIT,WMC etc. are found suitable [10], [11] in testability estimation both at deigning & coding level [16], [24], [43].The metrics have been empirically validated using commercial java software's and junit test classes for establishing strong correlation with testability by many of them[3], [14], [24], [44]. Thus directly or indirectly the role of these metrics had been trivial in testability quantification.
- **Six Major Contributing Factors:** It has been found after rigorous study that overall testability is affected by these six quality factors ***Controllability, Observability, Complexity, Traceability, Understandability, and Built In Test***[6]**.** Many factors have been the focus of attention by different researchers at different stages of software development cycle[3], [12], [15], [17], [19], [22], [45], [46], which has lead to various theories, models and metrics for quantification of these factors, which later help improving testability. But the study still does not show an elaborative impact of all of them together for testability improvisation or test effort reduction.

4.2 Testability Anti-patterns

The software characteristics or design patterns, which leads to testability weakness and raise test effort have been termed as testability anti-patterns [47].The few of these diagnosed anti-patterns which affect the design and code of the software in terms of testability reduction are Cyclic dependencies, Dynamic Binding, Exception Handling, Recursive Implementation, Unstructured Code, Self usage relationships, Class interactions. Though the avoidance of these factors is not the focus of discussion but how and at what level these anti-patterns need to be handled needs to be analysed. Some of these have already been analysed in previous research but overall relation of all in context of object oriented systems needs more elaboration.

4.3 Testability Improvisation Techniques

There are many suggestive ways to improvise the overall testability of object oriented systems at design and code time. Commercial systems should take these into consideration at various phases during development so as to bring design changes and code transformations at right stage avoiding high testing effort and development cost. Lot of improvisations is suggested in UML diagrams [15], [48] along with various testability estimation techniques using several metrics on object oriented design[14], [16], [22],

[49] applicable at design time to resolve testability issues. The further continuity of anti-patterns and testability issues exist in the systems due to unstructured coding and complex testing techniques which can be avoided or improved using coding testability analysis such as PIE [50] or by introducing Dependency Injections [51], software contracts [29], [52]z, checkpoints & wrappers [53], simulation function, built in test code [54]–[57]. Also lot of efforts have been put on investigating testability from unit test perspective [35], [43], [44] using various metrics, which may also help in building batter testable systems.

5. Conclusion

Software testability is becoming an important factor to consider during the software development and assessment. The overall review of testability at various stages in software development life cycle and in various forms reveals lot of gaps in the study of testability. The study needs constructive and quantitative analysis of major quality factors at design as well as coding phase. The individual impact few of these factors may have been taken care of already but the combined effect of all of them keeping key object oriented features in focus has not yet been analysed. Also the shift in testability study focus from coding to designing phase is gaining popularity. Though the attention is provided at designing phase but one cannot ignore the various hidden programming faults, which gets added to more test effort, hence one should not restrict the focus only on designing but also attention should be paid at source code and testing phase testability improvisation. This in turn will help the software engineers to not only reduce testing effort and development cost but also improving the quality of software significantly along with producing highly reliable, maintainable and easily testable software.

References

[1] IEEE, "IEEE Standard Glossary of Software Engineering Terminology (IEEE Std 610.12-1990)," 1990.
[2] ISO, "ISO/IEC 9126: Software Engineering Product Quality," 2002.
[3] S. Mouchawrab, L. C. Briand, and Y. Labiche, "A measurement framework for object-oriented software testability," *Inf. Softw. Technol.*, vol. 47, no. April, pp. 979–997, 2005.
[4] D. Romano, P. Raila, M. Pinzger, and F. Khomh, "Analyzing the impact of antipatterns on change-proneness using fine-grained source code changes," *Proc. - Work. Conf. Reverse Eng. WCRE*, pp. 437–446, 2012.
[5] R. Osherove, *The Art of Unit Testing: with .NET Examples*, vol. 0. 2010.
[6] R. V Binder, "Design For Testabity in Object-Oriented Systems," *Commun. ACM*, vol. 37, pp. 87–100, 1994.
[7] M. Badri, A. Kout, and F. Toure, "An empirical analysis of a testability model for object-oriented programs," *ACM SIGSOFT Softw. Eng. Notes*, vol. 36, no. 4, p. 1, 2011.
[8] S. Khatri, "Improving the Testability of Object-oriented Software during Testing and Debugging Processes," *Int. J. Comput. Appl.*, vol. 35, no. 11, pp. 24–35, 2011.
[9] Z. G. Ding, "Research and practice of the IVF software testability model," *Proc. - 2010 2nd WRI World Congr. Softw. Eng. WCSE 2010*, vol. 2, pp. 249–252, 2010.
[10] S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object Oriented Design," *IEEE Trans. Softw. Eng.*, vol. 20, no. 6, pp. 476–493, 1994.
[11] B. Henderson-Sellers, *Object-Oriented Metric*. New Jersey: Prentice Hall, 1996.
[12] M. Nazir, "An Empirical Validation of Complexity Quatification Model," *Int. J. Adv. Res. Comput. Sci. Softw. Eng.*, vol. 3, no. 1, pp. 444–446, 2013.
[13] M. Nazir, R. A. Khan, and K. Mustafa, "A Metrics Based Model for Understandability Quantification," vol. 2, no. 4, pp. 90–94, 2010.
[14] M. Nazir and K. Mustafa, "An Empirical Validation of Testability Estimation Model," *Int. J. Adv. Res. Comput. Sci. Softw. Eng.*, vol. 3, no. 9, pp. 1298–1301, 2013.
[15] S. Khalid, S. Zehra, and F. Arif, "Analysis of object oriented complexity and testability using object oriented design metrics," in *Proceedings of the 2010 National Software Engineering Conference on - NSEC '10*, 2010, pp. 1–8.
[16] R. A. Khan and K. Mustafa, "Metric based testability model for object oriented design (MTMOOD)," *ACM SIGSOFT Softw. Eng. Notes*, vol. 34, no. 2, p. 1, 2009.
[17] E. Mulo, "Design for testability in software systems," 2007.
[18] M.-C. Shih, "Verification & Measurement of Software Component Testabiliy," 2004.
[19] S. Jungmayr, "Design for testability," in *Pacific Northwest Software Quality Conference*, 2002, pp. 57–64.
[20] S. Jungmayr, "Reviewing Software Artifacts for Testability," 2002.
[21] B. Baudry, Y. Le Traon, G. Sunye, and J. M. Jézéquel, "Towards a ' Safe ' Use of Design Patterns to Improve OO Software Testability," *Softw. Reliab. Eng. 2001. ISSRE 2001. Proceedings. 12th Int. Symp.*, pp. 324–329, 2001.
[22] B. W. N. Lo and H. Shi, "A preliminary testability model for object-oriented software," *Proceedings. 1998 Int. Conf. Softw. Eng. Educ. Pract. (Cat. No.98EX220)*, pp. 1–8, 1998.
[23] R. S. Freedman, "Testability of software components - Rewritten," *IEEE Trans. Softw. Eng.*, vol. 17, no. 6, pp. 553–564, 1991.
[24] L. Badri, M. Badri, and F. Toure, "An empirical analysis of lack of cohesion metrics for predicting testability of classes," *Int. J. Softw. Eng. its Appl.*, vol. 5, no. 2, pp. 69–86, 2011.
[25] M. Ó. Cinnéide, D. Boyle, and I. H. Moghadam, "Automated refactoring for testability," *Proc. - 4th IEEE Int. Conf. Softw. Testing, Verif. Valid. Work. ICSTW 2011*, pp. 437–443, 2011.
[26] M. Harman, "Refactoring as testability transformation," in *Proceedings - 4th IEEE International Conference on Software Testing, Verification, and Validation Workshops, ICSTW 2011*, 2011, pp. 414–421.
[27] M. Harman, A. Baresel, D. Binkley, and R. Hierons, "Testability Transformation: Program Transformation to Improve Testability," 2011.
[28] Y. Singh and A. Saha, "Enhancing data flow testing of classes through design by contract," *Proc. - 7th*

IEEE/ACIS Int. Conf. Comput. Inf. Sci. IEEE/ACIS ICIS 2008, conjunction with 2nd IEEE/ACIS Int. Work. e-Activity, IEEE/ACIS IWEA 2008, pp. 567–574, 2008.
[29] Y. Singh and A. Saha, "Improving the testability of object oriented software through software contracts," *ACM SIGSOFT Softw. Eng. Notes*, vol. 35, no. 1, p. 1, 2010.
[30] A. González, É. Piel, and H.-G. Gross, "A model for the measurement of the runtime testability of component-based systems," in *IEEE International Conference on Software Testing, Verification, and Validation Workshops, ICSTW 2009*, 2009, pp. 19–28.
[31] W. Zheng and G. Bundell, "Test by contract for UML-based software component testing," in *Proceedings - International Symposium on Computer Science and Its Applications, CSA 2008*, 2008, no. 4, pp. 377–382.
[32] T. B. Nguyen, M. Delaunay, and C. Robach, "Testability Analysis of Data-Flow Software," *Electron. Notes Theor. Comput. Sci.*, vol. 116, pp. 213–225, 2005.
[33] M. Bruntink and A. Vandeursen, "Predicting class testability using object-oriented metrics," in *Proceedings - Fourth IEEE International Workshop on Source Code Analysis and Manipulation*, 2004, pp. 136–145.
[34] M. Bruntink, "Testability of Object-Oriented Systems : a Metrics-based Approach," 2003.
[35] M. Bruntink and A. Van Deursen, "An empirical study into class testability," *J. Syst. Softw.*, vol. 79, no. 9, pp. 1219–1232, 2006.
[36] Y. Wang, G. King, I. Court, M. Ross, and G. Staples, "On testable object-oriented programming," *ACM SIGSOFT Softw. Eng. Notes*, vol. 22, no. 4, pp. 84–90, 1997.
[37] J.-C. Lin, S. Lin, and L. Huang, "An approach to software testability measurement," in *Proceedings of Joint 4th International Computer Science Conference and 4th Asia Pacific Software Engineering Conference*, 1997, pp. 515–516.
[38] J. McGregor and S. Srinivas, "A measure of testing effort," in *Proceedings of the Conference on Object-Oriented Technologies, USENIX Association*, 1996, vol. 9, pp. 129–142.
[39] J. M. Voas and K. W. Miller, "Improving the software development process using testability research," *Softw. Reliab. Eng. 1992. ...*, 1992.
[40] J. M. Voas, K. W. Miller, and J. E. Payne, "PISCES: a tool for predicting software testability," *[1992] Proc. Second Symp. Assess. Qual. Softw. Dev. Tools*, 1992.
[41] J. M. Voas and K. W. Miller, "Software Testability : The New Verification," pp. 187–196, 1993.
[42] J. M. Voas, "Testability, testing, and critical software assessment," *Proc. COMPASS'94 - 1994 IEEE 9th Annu. Conf. Comput. Assur.*, no. 2, pp. 165–167, 1994.
[43] M. Badri, "Empirical Analysis of Object-Oriented Design Metrics for Predicting Unit Testing Effort of Classes," *J. Softw. Eng. Appl.*, vol. 05, no. July, pp. 513–526, 2012.
[44] Y. Singh and A. Saha, "Predicting Testability of Eclipse: Case Study," *J. Softw. Eng.*, vol. 4, no. 2, pp. 122–136, 2010.
[45] G. Jimenez, S. Taj, and J. Weaver, "Design For Testability," in *The NCIIA 9th Annual Meeting*, 2005, pp. 75–84.
[46] J. Gao, "Componenet Testability and Component Testing Challenges."
[47] B. Baudry, Y. Le Traon, G. Sunye, and J. M. Jezequel, "Measuring and improving design patterns testability," *9th Int. Symp. Softw. Metrics, METRICS 2003*, pp. 50–59, 2003.
[48] B. Baudry and Y. Le Traon, "Measuring design testability of a UML class diagram," *Inf. Softw. Technol.*, vol. 47, no. 13, pp. 859–879, 2005.
[49] T. J. McCabe and C. W. Butler, "Design complexity measurement and testing," *Commun. ACM*, vol. 32, no. 12, pp. 1415–1425, 1989.
[50] J. M. Voas, "PIE : A Dynamic Failure-Based Technique," *IEEE Trans. Softw. Eng.*, vol. 18, no. 8, pp. 717–727, 1992.
[51] R. Lindooren, "Testability of Dependency Injection," 2007.
[52] L. L. Liu, B. Meyer, and B. Schoeller, "Using Contracts and Boolean Queries to Improve the Quality of Automatic Test Generation," *Lect. Notes Comput. Sci.*, pp. 114–130, 2007.
[53] J.-C. Lin, S. Lin, and Ian-Ho, "An estimated method for software testability measurement," in *Proceedings Eighth IEEE International Workshop on Software Technology and Engineering Practice incorporating Computer Aided Software Engineering*, 1997, pp. 116–123.
[54] J. Vincent and G. King, "Principles of Built-In-Test for Run-Time-Testability in Component-Based Software Systems," pp. 115–133, 2002.
[55] T. Jeon, "Increasing the Testability of Object-Oriented Frameworks with Built-in Tests," *Building*, pp. 169–182, 2002.
[56] B. Pettichord, "Design for Testability," *Pettichord.com*, pp. 1–28, 2002.
[57] J. Gao, "Short Course Topic : Advances in Component-Based Software Testing," 2006.

Authors Profile

Dr. Pushpa R. Suri received her Ph.D. Degree from Kurukshetra University, Kurukshetra. She is working as Associate Professor in the Department of Computer Science and Applications at Kurukshetra University, Kurukshetra, Haryana, India. She has many publications in International and National Journals and Conferences. Her teaching and research activities include Discrete Mathematical Structure, Data Structure, Information Computing and Database Systems.

Harsha Singhani received her Master of Computer Application degree from Maharishi Dayanand University, Rohtak, Haryana, India. She has got experience of over 12 years of teaching in field of I.T. At present, she is pursuing Ph.D. (Computer Science) from Kurukshetra University, Kurukshetra, Haryana. Her teaching and research areas include database systems, automata theory, object oriented programming and software testing.