Testability Estimation Framework

Mohd Nazir
Deprtament of Computer Science
Jamia Millia Islamia
New Delhi, India

Dr. Raees A. Khan
Deprtament of IT
BBA University
Lucknow, India

Dr. K. Mustafa
Deprtament of Computer Science
Jamia Millia Islamia
New Delhi, India

ABSTRACT
Testability has always been an elusive concept and its correct measurement or evaluation a difficult exercise. Most of the studies measure testability or more precisely the attributes that have impact on testability but at the source code level. Though, testability measurement at the source code level is a good indicator of effort estimation, it leads to the late arrival of information in the development process. A decision to change the design in order to improve testability after coding has started may be very expensive and error-prone. While estimating testability early in the development process may greatly reduce the overall cost. This paper provides a roadmap to industry personnel and researchers to assess, and preferably, quantify software testability in design phase. A prescriptive framework has been proposed in order to integrate testability within the development life cycle. It may be used to benchmark software products according to their testability.

Categories and Subject Descriptors
D.3.3 [Software Testability]: Testability Estimation, Testability Factors, Software Design

General Terms
Testability Metrics, Testability Models, Testability Index.

Keywords
Software Testability, Testability Estimation Framework, Software Design, Software Quality

1. INTRODUCTION

In today's world, the importance of delivering quality software is no longer an advantage but a necessary factor. However, with the growing complexity, pervasiveness and criticality of software, major factor of assuring that it behaves according to the desired level of quality and dependability has become more crucial, increasingly difficult and expensive. Moreover, the complexity of applications and environments has substantially increased in the last couple of decades. Unfortunately, most of the software industries not only fail to deliver a quality product to their customers, but also do not understand the relevant quality attributes [21]. The development of quality software still remains a matter of guidelines, best practices and undocumented expert knowledge.

In the highly competitive IT industry, consumer pressure causes companies to accelerate the speed to market software products. Schedules are often tightly restricted; developers are forced to weigh the importance of quality against the possibility of missing deadlines. For meeting the target, '*on time delivery'*, testing time is generally reduced, which increases the potential for defects, leading to problems with the software. This includes incomplete design, poor quality, high maintenance costs, and the risk of loosing customer satisfaction. According to a statistical report, more than 80% of all software released in the United States is not reviewed for defects, at a cost to the state economy of tens of billions of dollars each year [26]. Under these circumstances, software quality tends to suffer leading to severe consequences.

It is an inevitable fact that software must be verified. It is true not only for critical systems where a failure might lead to loss of lives or great economical values, but also for non-critical systems. A system that deviates from the expected behavior is often worse than no system at all. Many things go into the release of high quality software. The software needs to be well conceived, well-designed, well-coded and well-tested. Design is the process of trade-offs between qualities. The flaws of design structure have a strong negative impact on quality attributes. Complex design often leads to poor testability, which may in turn leads to ineffective testing that may result to severe penalties and consequences. However, structuring a high-quality design continues to be an inadequately defined process [19]. Indeed, like all human activities, the process of designing software is also error prone and object-oriented design makes no exception. However, object orientation has proved its value for systems that must be maintained and modified. It has the capability to naturally lend itself to an early assessment and evaluation; it facilitates software design to be modeled at a higher level of abstraction. Consequently, any potential problem with the design can be fixed at the right time.

The process of Software Engineering evolves with a unique issue of testability. It is an external software attribute that assesses the complexity and effort required for testing software. The insight provided by testability is valuable during design, coding, testing and quality assurance [4]. Testability suggests testing intensity, and provides the degree of difficulty which will be incurred during testing of a particular location to detect a fault. Improving software testability is an important objective in order to reduce the number defects that result from poorly designed software [25]. It is an inevitable fact that testability information is useful that may be complementary to testing. Higher test coverage may

be achieved by making a system more testable for the same amount of effort. Achieving testability is mostly a matter of separation of concerns, coupling between classes and subsystems, and cohesion [24]. Dino Esposito argued that testability should be considered as a key attribute in order to guarantee the software quality [12]. Practitioners frequently advocate that testability should be planned early in the design phase. Rest of the paper is organized as follows. Section 2 briefly describes testability estimation. Section 3 summarizes the relevant work. Section 4 presents theoretical basis of the framework. Section 5 describes the framework. Section 6 concludes the paper and highlights the future work.

2. SOFTWARE TESTABILITY ESTIMATION

Testability is one of the most important quality indicators. Its measurement leads to the prospects of facilitating and improving a test process. However, testability has always been an elusive concept and its correct measurement or evaluation a difficult exercise [1]. There is no clear definition to '*what aspects of software are actually related to testability'* [11]. Several approaches, including prominently the Program-based Testability Measurement, Model-based Testability Measurement, and Dependability-based Testability Assessment have been proposed [13]. Further, several internal metrics on testability measurement have been published so far [18]. But, most of the metrics are applicable only at the later stage in the system development life cycle i.e. during implementation. Researchers and Practitioners suggested different ways of measuring and improving testability of software design. Wang argued that testability at class and system levels can be quantitatively modeled and analyzed [20]. John Hunt considers testability as a key design criterion, while Soumar et al advocated that measuring testability based on design artifacts can yield highest payoffs [1, 15]. Hence, it seems highly desirable and significant to implement testability at the design stage. Practitioners emphasize on the need of having a systematic approach for testability estimation. Therefore, there is a potential to develop a more systematic solution for testability estimation. Hence, the techniques for measuring testability that can be applied at the design stage are most likely preferable.

3. RELEVANT WORK

Software testability analysis has been an important research direction since 1990s and became more pervasive in 21st century [11]. A number of researchers addressed software testability, but in the context of conventional structured design. The question of testability [6] has been revived with the object-orientation [7, 8]. Despite the fact that object oriented technology has now been widely accepted by the software industry, only a few research studies have been devoted to explore the concepts of testability in object oriented systems. Several developments on the measurement of testability, design for testability have been reported in the literature [9, 10, 16, 17, 18]. Unfortunately, these achievements have not been widely accepted and hence, not been adopted in practice by industry [11]. Following sections briefly summarize some of the relevant efforts made by researchers in the area.

Voas and Miller [5] proposed testability metric based on the inputs and outputs domains of a software component. To measure testability, they proposed PIE (propagation, infection and execution) analysis technique [6]; but estimating testability via the PIE technique was a difficult and computationally expensive process. Hence, to obtain an indication of the testability of a program early in the software development process and without actually performing the PIE analyses, they suggested use of a semantic metric, the domain-to-range ratio (DRR): the ratio of the cardinality of the possible inputs to the cardinality of the possible outputs. Binder did a novel work highlighting the need and significance of software testability in system development [8]. He argued that a more testable system may provide increased reliability for a fixed testing budget. He proposed a fishbone model representing the key factors of testability. These factors are only described at a high level of abstraction, which lead to no clear relationship with the metrics that are based on design artifacts and the implementation. Bruce and Haifeng Shi [3] explored the factors of object oriented software that affect testability. Based on the fault-failure model of software testing, they proposed a framework that estimates total testability from the individual testability factors of its components.

Bruntink and van Deursen [2] defines a set of metrics for assessing the testability of classes of a Java system, and testability was characterized using source code metrics. Jungmayr [18] takes an integration testing point of view, and focuses on dependencies between components. He presented a novel approach that allows identifying local dependencies which are critical for global testability. Further, the notion of test-critical dependencies was proposed to identify these dependencies. The reduction *metric was used* to evaluate the impact of particular dependency on testability with respect to a given testability metric. Baudry et al. discussed the impact of specific types of class interactions on testability and suggested the use of various coupling and class interaction metrics that characterizes testability [14]. For measuring testability, Jerry and Ming proposed the quantifiable approach that is based on a pentagon model [13]. Samar et al proposed a framework to assess testability of design modeled with the UML [1]. They also proposed a set of operational hypotheses for each attribute that can explain its expected relationship with testability; but the hypotheses are not empirically validated. Mulo presented a report strengthening the integration of testability throughout development process [9].

Several approaches have proposed in the literature for measuring software testability. A survey of the relevant literature reveals that maximum efforts have been devoted at the later stage of development life cycle. In fact, testability measures give an indicator so as to the effectiveness and efficiency of testing. A decision to change the design in order to improve testability after coding has started may be very expensive and error-prone. Therefore, it is an obvious fact that assessing testability early in the development process may greatly reduce testing time, efforts and costs.

4. THEORITICAL BASIS

A number of researchers had addressed the notion of software testability and proposed various approaches for testability measurement. The mechanisms available for testability measurement may be used in later phases of the development life cycle. Though, testability assessment at the source code level is a

good indicator of effort estimation, it leads to the late arrival of information in the development process. A decision to change the design in order to improve testability after coding has started may be very expensive and error-prone.

A critical review of relevant literature reveals that existing work on the topic either takes a very specific viewpoint or remains at a very general level [1, 22]. Furthermore, software testability as a field has not matured enough. Even processes, guidelines and tools related to testability are missing, but advocated generally to be inevitable. Researchers and practitioners frequently advocated that testability should be measured based on the design artifacts. The early estimation of testability, exclusively at design phase can yield the highest payoffs. On the other hand, the lack of testability at design stage may not be compensated during subsequent development activities.

Practitioners strongly felt and recommended that a systematic approach, which can incorporate testability at the design stage, is highly desirable and significant [1, 9]. It may guide to avoid wastage of resources and to enable continuous improvement. It is evident from the literature survey that there is no known comprehensive and complete model or framework for evaluating the testability of designs developed using a object oriented approach based on its internal design property[1][22][23]. Aforementioned discussion and facts forms a strong theoretical basis to formulate a testability estimation roadmap to be integrated during design phase. Further, such a framework or road map, which can quantify testability of object oriented software at design stage, seem to be worthwhile and fruitful.

5. THE FRAMEWORK

As a matter of fact, researchers and practitioners highly recommend an efficient and accurate measure of software testability early in design phase. There is a common consensus among industry professionals and academicians in integrating testability within the development life cycle in order to deliver quality software. Unfortunately, there is no standard methodology or guideline available to quantify software testability. Therefore, such a roadmap or framework, which can be followed by industry personnel and researchers to quantify testability early in design phase, appears highly desirable and significant. A prescriptive framework as depicted in figure 5.0 (a) has been proposed to estimate testability of object oriented software at design level. Moreover, a fishbone model shown in figure 5.0 (b) has been presented in order to emphasize the importance of estimating testability at design stage, and to more clearly elaborate 'the idea illustrated in the framework'. The framework comprised of seven phases including a common phase of *'Design Review'***.** A brief description of the framework components is given as follows.

5.1 Testability Factorization

Testability is a high level factor to software quality. In order to quantify testability, its direct measures are to be identified. In this phase, the commonly accepted set of factors to testability is to be identified. Design level factors will also be investigated keeping in view their impact on the overall testability.

5.2 Software Characterization

Different software characteristics have their impacts on testability and quality as well. Object oriented software characteristics will be identified in this phase. The contribution of each characteristic to improve the design will also be analyzed.

5.3 Metric Selection

The metrics are the calculation of the skill of the development team in making their classes testable. Metric selection is an important step in estimating testability. In the absence of any testability metric in design phase, a suite of testability metrics is to be proposed that may serve the purpose.

5.4 Correlation Establishment

This is also the key step of the proposed framework, where the identified testability factors are to be correlated with the OO design characteristics. A regression line will be established to quantify testability factors in terms of design characteristics with the help of design metrics.

5.5 Testability Quantification

Established regression will be used to quantify testability factors using design metric values. A design hierarchy will be used as an input to the set formulation. Metric values are to be computed using the given hierarchy and these values are to be used to quantify testability factors.

5.6 Qualitative Assessment

On the basis of the quantitative values obtained, a qualitative assessment of testability factors is performed. A contextual finding will be discussed and used for review and revision of the given design. This phase will help in benchmarking software products according to their testability.

5.7 Design Review

On the basis of the results obtained from the qualitative assessment phase, the given design is to be reviewed and revised to achieve better level of testability. Design constructs are to be critically examined and may be adjusted accordingly in order to achieve the index value.

6. CONCLUSION

The framework proposed in the paper will address testability during software development life cycle. It may help putting testability benchmarking of software projects. The framework is generic in nature, and may be used by industry practitioners to quantify testability in order to make design decisions early in the development life cycle. Strong theoretical basis presented in the paper supports the claim of the framework's usability to estimate testability of object oriented software at design phase. Framework's implementation is in progress, and will come out as our future work.

7. REFERENCES

[1] S. Mouchawrab, L. C. Briand, and Y. Labiche, "A measurement framework for object-oriented software testability", Info. and Software Technology, Volume 47, Issue 15, December 2005, Pages 979-997.

[2] M. Bruntink and A. V. Deursen, Predicting class estability using object-oriented metrics, in Proc. IEEE international

Workshop on Source Code Analysis and Manipulation, 2004, pp. 136-145.

[3] Bruce W.N.Lo and Haifeng Shi, A preliminary testability model for object-oriented software, in Proc. International Conf. on SoftwareEngineering, Education, Practice, Pages 330{337. IEEE. 1998.

[4] Voas and Miller, Improving the software development process using testability research, *IEEE Software*, pp. 114-121, 1992.

[5] Voas and Miller, Semantic metrics for software testability, *Journal of Systems and Software*, Vol. 20 (3), pp. 207-216, 1993.

[6] Voas and Miller, "Software Testability: The New Verification". IEEE Software. Vol. 12(3), p. 17-28, 1995.

[7] J.M. Voas. "Object-Oriented Software Testability". In proceedings of *International Conference on Achieving Quality in Software*, January 1996

[8] R.V. Binder, "Design for testability in object-oriented systems". Communications of the ACM. Vol. 37(9), p. 87-101, 1994.

[9] E. Mulo, "Design for Testability in Software Systems", Master's Thesis, 2007.
URL:swerl.tudelft.nl/twiki/pub/Main/ResearchAssignment/RA-Emmanuel-Mulo.pdf

[10] S. Jungmayr, "Design for Testability", *CONQUEST* 2002, pp. 57-64.

[11] L. Zhao, "A new approach for software testability analysis", International Conference on Software Engineering, Proceeding of the 28th international conference on Software Engineering, Shanghai, 2006, pp. 985–988.

[12] Dino Esposito, "Design Your Classes for Testability", 2008.
URL:http://dotnetslackers.com/articles/n net/Design-Your-Classes-for- Testability.aspx

[13] J. Gao and Ming-Chih Shih, A component testability model for verification and measurement, In Proc. of the 29th Annual International Computer Software and Applications Conference, pages 211–218. IEEE Comp Society 2005.

[14] Baudry and Traon, Measuring Design Testability of a UML Class Diagram. Information and Software Technology, 47(13):859–879,2005.

[15] J. Hunt, "Designing Software for Testability", Oct 2007.
URL:http://www.theregister.co.uk/2007/10/29/design_for_testability/

[16] Pettichord, B. Design for Testability. In Proc. of Pacific Northwest Software Quality Conference, 2002.

[17] Jimenez, G., Taj, S., and Weaver, J. Design for Testability. in Proceedings of the 9th Annual NCIIA Conference, 2005.

[18] Jungmayr, S. Testability Measurement and Software Dependencies. In Proceedings of the 12th International Workshop on Software Measurement, pp. 179–202, October 2002.

[19] C. Valdaliso, O. Eljabiri, F.P. Deek, "Factors Influencing Design Quality and Assurance in Software Development: An Empirical Study", Electronic Proceedings of the First International Workshop on Model-based Requirements Engineering (MBRE 01), San Diego, California, 2001.

[20] Y. Wang, "Design for Test and Software Testability", University of Calgary, 2003.
URL:http://www.ucalgary.ca/~ageras/wshop/abstracts/2003/design-for-estability.htm

[21] R. A. Khan, K. Mustafa, I Ahson, "An Empirical Validation of Object Oriented Design Quality Metrics, Journal King Saud University, Computer & Information Science, Vol. 19, pp. 1-16, Riyadh (1427H/2007).

[22] M. Nazir, R. A. Khan, "Testability Estimation of Object Oriented Software: A Critical Review", in the proceeding of International conference on Information and Communication Technologies", Dehradun, 2007.

[23] R A Khan & K Mustafa, "A Model for Object Oriented Design Quality Assessment", *Proceedings, Integrated Design and Process Technology Symposium*, Kusadasi, Izmir, Turkey, June 28-July 2, 2004.

[24] D. Jeremy Miller, "Designing for Testability", the Shade Tree Developer, Jun 29 2007.
URL:http://codebetter.com/blogs/jeremy.miller/archive/2007/06/29/designing-fortestability.aspx

[25] "Design for Testability", An e-newsletter published by Software Quality Consulting,Inc. 3[3], March 2006.
URL:http://www.swqual.com/newsletter/vol3/no3/vol3no3.html

[26] Light, Matt., "Use Good Practices in Software Quality and Testing or Pay the Price", Gertner Toolkit Tutorial, August, 2007.

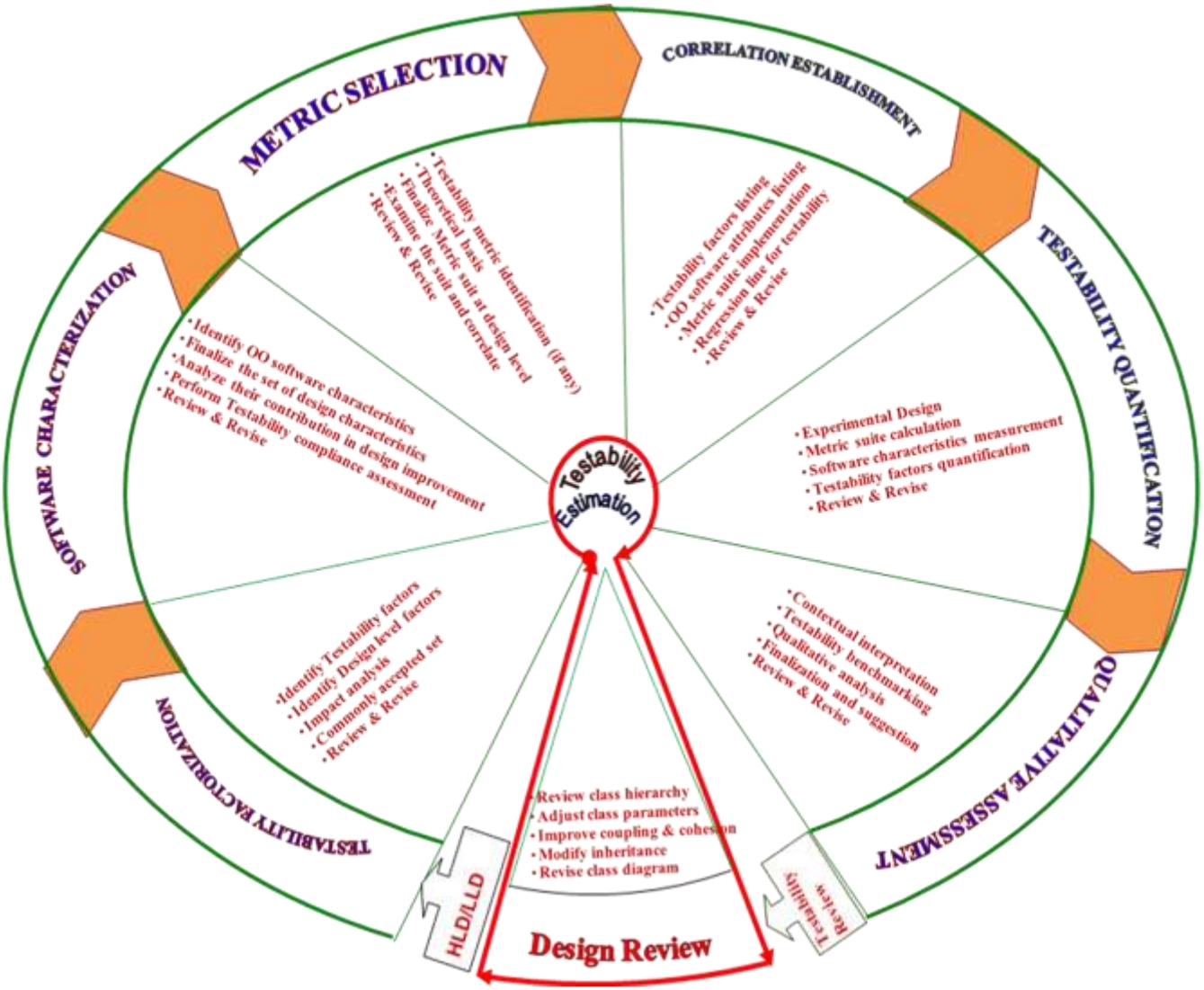

Figure 5.0 (a) Testability Estimation Framework at Design Level

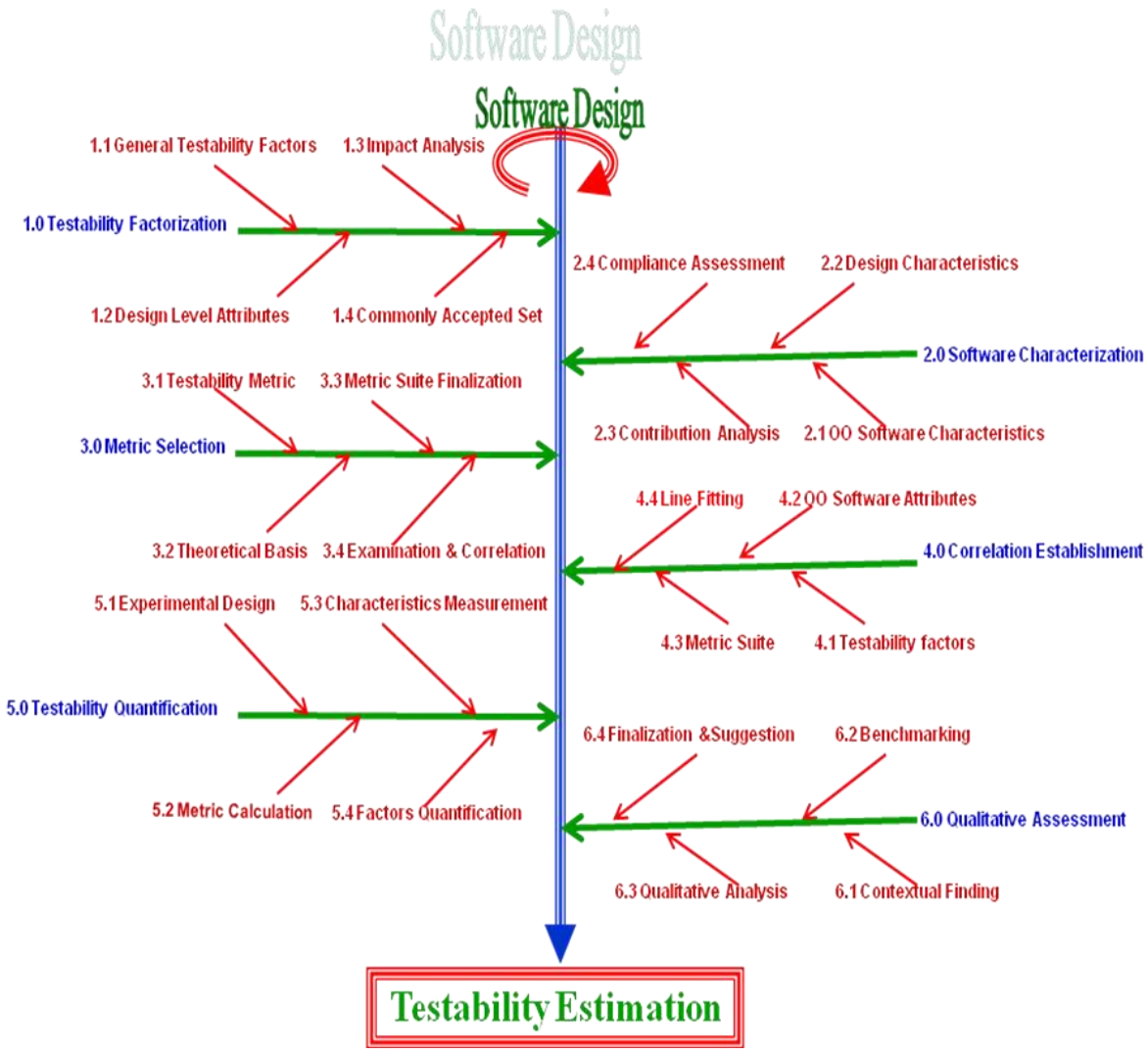

Figure 5.0 (b) Testability Fishbone Model