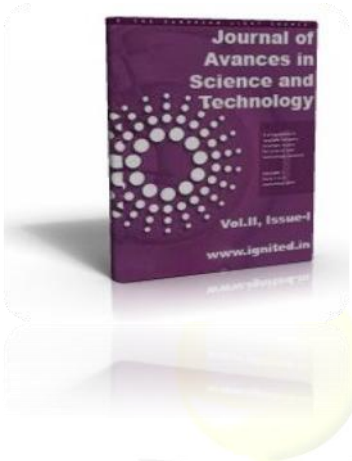
ISSN-2230-9659

Study of Different Software Testing Techniques



Harmanpreet Singh
Research Scholar, Manav Bharti University
H.P., INDIA

1 INTRODUCTION

Software testing is as old as the hills in the history of digital computers. The testing of software is an important means of assessing the software to determine its quality. Since testing typically consumes 40~50% of development efforts, and consumes more effort for systems that require higher levels of reliability, it is a significant part of the software engineering. With the development of Fourth generation languages (4GL), which speeds up the implementation process, the proportion of time devoted to testing increased. As the amount of maintenance and upgrade of existing systems grow, significant amount of testing will also be needed to verify systems after changes are made [12]. Despite advances in formal methods and verification techniques, a system still needs to be tested before it is used. Testing remains the truly effective means to assure the quality of a software system of non-trivial complexity [13], as well as one of the most intricate and least understood areas in software engineering [19]. Testing, an important research area within computer science is likely to become even more important in the future.

ISSN-2230-9659

This retrospective on a fifty-year of software testing technique research examines the maturation of the software testing technique research by tracing the major research results that have contributed to the growth of this area. It also assesses the change of research paradigms over time by tracing the types of research questions and strategies used at various stages. We employ the technology maturation model given by Redwine and Riddle [15] as the framework of our studies of how the techniques of software testing first get the idea formulated, preliminarily used, developed, and then extended into a broader solution. Shaw gives a very good framework of software engineering research paradigms in [17], which classifies the research settings, research approaches, methods, and research validations that have been done by software researchers. Shaw's model is used to evaluate the research strategies for testing techniques used in our paper.

2 THE TAXONOMY OF TESTING TECHNIQUES

Software testing is a very broad area, which involves many other technical and non-technical areas, such as specification, design and implementation, maintenance, process and management issues in software engineering. Our study focuses on the state of the art in testing techniques, as well as the latest techniques which representing the future direction of this area. Before stepping into any detail of the maturation study of these techniques, let us have a brief look at some technical concepts that are relative to our research.

2.1 The Goal of Testing

In different publications, the definition of testing varies according to the purpose, process, and level of testing described. Miller gives a good description of testing in [13]:

The general aim of testing is to affirm the quality of software systems by systematically exercising the software in carefully controlled circumstances.

Miller's description of testing views most software quality assurances activities as testing. He contends that testing should have the major intent of finding errors. A good test is one that has a

ISSN-2230-9659

high probability of finding an as yet undiscovered error, and a successful test is one that uncovers an as yet undiscovered error.

This general category of software testing activities can be further divided. For purposes of this paper, testing is the dynamic analysis of a piece of software, requiring execution of the system to produce results, which are then compared to expected outputs.

2.2 The Testing Spectrum

Testing is involved in every stage of software life cycle, but the testing done at each level of software development is different in nature and has different objectives.

Unit Testing is done at the lowest level. It tests the basic unit of software, which is the smallest testable piece of software, and is often called "unit", "module", or "component" interchangeably.

Integration Testing is performed when two or more tested units are combined into a larger structure.

The test is often done on both the interfaces between the components and the larger structure being constructed, if its quality property cannot be assessed from its components.

System Testing tends to affirm the end-to-end quality of the entire system. System test is often based on the functional/requirement specification of the system. Non-functional quality attributes, such as

reliability, security, and maintainability, are also checked.

Acceptance Testing is done when the completed system is handed over from the developers to the customers or users. The purpose of acceptance testing is rather to give confidence that the system is working than to find errors.

ISSN-2230-9659

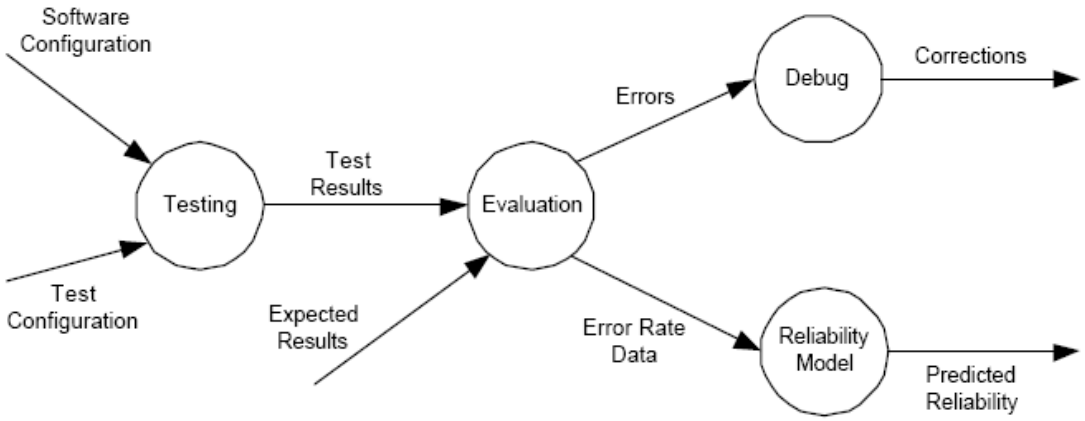


Figure 1. Testing Information Flow

3 SCOPE OF THE STUDY

3.1 Technical Scope

In this paper, we focus on the technology maturation of testing techniques, including these functional and structural techniques that have been influential in the academic world and widely used in practice. We are going to examine the growth and propagation of the most established strategy and methodology used to select test cases and analyze test results. Research in software testing techniques can be roughly divided into two branches: theoretical and methodological, and the growth in both branches push the growth of testing technology together. Inhibitors of maturation, which explains why the in-depth research hasn't brought revolutionary advantage in industry testing practice, are also within our scope of interest.

There are many other interesting areas in software testing. We limit the scope of our study within the range of testing techniques, although some of the areas maybe inseparable from our study. Specifically, we are ***not*** going to discuss:

- How testing is involved in the software development cycle

ISSN-2230-9659

- How different levels of testing are performed
- Testing process models
- Testing policy and management responsibilities, and
- Stop criteria of testing and software testability

3.2 Goal and standard of progress

The ultimate goal of software testing is to help designers, developers, and managers construct systems with high quality. Thus research and development on testing aim at efficiently performing effective testing – to find more errors in requirement, design and implementation, and to increase confidence that the software has various qualities. Testing technique research leads to the destination of practical testing methods and tools. Progress toward this destination requires fundamental research, and the creation, refinement, extension, and popularization of better methods.

The standard of progress for the research of testing techniques include:

- Degree of acceptance of the technology inside and outside the research community
- Degree of dependability on other areas of software engineering
- Change of research paradigms in response to the maturation of software development technologies
- Feasibility of techniques being used in a widespread practical scope, and
- Spread of technology – classes, trainings, management attention

4 CONCLUSION

ISSN-2230-9659

Testing has been widely used as a way to help engineers develop high-quality systems, and the techniques for testing have evolved from an ad hoc activities means of small group of programmers to an organized discipline in software engineering. However, the maturation of testing techniques has been fruitful, but not adequate. Pressure to produce higher-quality software at lower cost is increasing and existing techniques used in practice are not sufficient for this purpose. Fundamental research that addresses the challenging problems, development of methods and tools, and empirical studies should be carried out so that we can expect significant improvement in the way we test software. Researchers should demonstrate the effectiveness of many existing techniques for large industrial software, thus facilitating transfer of these techniques to practice. The successful use of these techniques in industrial software development will validate the results of the research and drive future research. The pervasive use of software and the increased cost of validating it will motivate the creation of partnerships between industry and researchers to develop new techniques and facilitate their transfer to practice. Development of efficient testing techniques and tools that will assist in the creation of high-quality software will become one of the most important research areas in the near future.

ANNOTATED BIBLIOGRAPHY

[1] G. Bernet, L. Bouaziz, and P. LeGall, "A Theory of Probabilistic Functional Testing," Proceedings of the 1997 International Conference on Software Engineering, 1997, pp. 216 – 226 [BBL97] A framework for probabilistic functional testing is proposed in this paper. The authors introduce the formulation of the testing activity, which guarantees a certain level of confidence into the correctness of the system under test. They also explain how one can generate appropriate distributions for data domains including most common domains such as intervals of integers, unions, Cartesian products, and inductively defined sets. A tool assisting test case generation according to this theory is proposed. The method is illustrated on a small formal specification.

Question: Method/Means

ISSN-2230-9659

Result: Technique

Validation: Analysis

[2] B. Beizer, "Software Testing Techniques," Second Edition, Van Nostrand Reinhold Company Limited, 1990, ISBN 0-442-20672-0 [Beizer90] This book gives a fairly comprehensive overview of software testing that emphasizes formal models for testing. The author gives a general overview of the testing process and the reasons and goals for testing. In the second chapter of this book, the author classifies the different types of bugs that could arise in program development. The notion of path testing, transaction flowgraphs, data-flow testing, domain testing, and logic-based testing are introduced in detail in the chapters followed. The author also introduces several attempts to quantify program complexity, and more abstract discussion involving paths, regular expression, and syntax testing. How to implement software testing based on the strategies is also discussed in the book.

[3] S. Beydeda and V. Gruhn, "An integrated testing technique for component-based software," ACS/IEEE International Conference on Computer Systems and Applications, June 2001, pp 328 – 334 [BG01] Testing is made complicated with features, such as the absence of component source code, that are specific to component-based software. The paper proposes a technique combining both black-box and white-box strategies. A graphical representation of component software, called component-based software flow graph (CBSFG), which visualizes information gathered from both specification and implementation, is described. It can then be used for test case identification based on well-known structural techniques.

Question: Method/Means

Result: Technique

Validation: Analysis

ISSN-2230-9659

[4] A. Bertolino, P. Inverardi, H. Muccini, and A. Rosetti, "An approach to integration testing based on architectural descriptions," Proceedings of the IEEE ICECCS- 97, pp. 77-84 [BIMR97] In this paper the authors propose to use formal architectural descriptions (CHAM) to model the behavior of interest of the systems. Graph of all the possible behaviors of the system in terms of the interactions between its components is derived and further reduced. A suitable set of reduced graphs highlights specific architectural properties of the system, and can be used for the generation of integration tests according to a coverage strategy, analogous to the control and data flow graphs in structural testing.

Question: Method/Means

Result: Technique

Validation: Persuasion

[5] J.B. Good Enough and S. L. Gerhart, "Toward a Theory of Test Data Selection," IEEE Transactions on Software Engineering, June 1975, pp. 156-173 [GG75] This paper is the first published paper, which attempted to provide a theoretical foundation for testing. The "fundamental theorem of testing" brought up by the authors characterizes the properties of a completely effective test selection strategy. The authors think a test selection strategy is completely effective if it is guaranteed to discover any error in a program. As an example, the effectiveness of branch and path testing in discovering errors is compared. The use of decision table (a mixture of requirements and design-based functional testing) as an alternative method is also proposed.

Question: Evaluation

Result: Analytic Model

Validation: Analysis

ISSN-2230-9659

[6] D. Gelperin and B. Hetzel, "The Growth of Software Testing", Communications of the ACM, Volume 31 Issue 6, June 1988, pp. 687-695 [GH88] In this article, the evolution of software test engineering is traced by examining changes in the testing process model and the level of professionalism over the years. Two phase models, the demonstration and destruction models, and two life cycle models, the evolution and prevention models are given to characterize the growth of software testing with time. Based on the models a prevention oriented testing technology is introduced and analyzed in detail.

Question: Characterization

Result: Descriptive Model

Validation: Persuasion

[7] J. Hartmann, C. Imoberdorf, and M.Meisinger, "UML-Based Integration Testing," Proceedings of the International Symposium on Software Testing and Analysis, ACM SIGSOFT Software Engineering Notes, August 2000 [HIM00] Unified Modeling Language (UML) is widely used for the design and implementation of distributed, component-based applications. In this paper, the issue of testing components by integrating test generation and test execution technology with commercial UML modeling tools such as Rational Rose is addressed. The authors present their approach to modeling components and interactions, describe how test cases are derived from these component models and then executed to verify their conformant behavior. The TnT environment of Siemens is used to evaluate the approach by examples

Question: Method/Means

Result: Technique

Validation: Experience

ISSN-2230-9659

[8] W. E. Howden, "Reliability of the Path Analysis Testing Strategy", IEEE Transactions on Software Testing, September 1976, pp. 208-215 [Howden76] The reliability of path testing provides an upper bound for the testing of a subset of a program's paths, which is always the case in reality. This paper begins by showing the impossibility of constructing a test strategy that is guaranteed to discover all errors in a program. Three commonly occurring classes of errors, computations, domain, and subcase, are characterized. The reliability properties associated with these errors affect how path testing is defined.

Question: Characterization

Result: Technique

Validation: Analysis

[9] W. E. Howden, "Functional Testing and Design Abstractions," The Journal of System and Software, Volum 1, 1980, pp. 307-313 [Howden80] The usual practice of functional testing is to identify functions that are implemented by a system or program from requirements specifications. In this paper, the necessity of testing design as well as requirement functions is discussed. The paper indicates how systematic design methods, such as Structured design and the Jackson design can be used to construct functional tests. Structured design can be used to identify the design functions that must be tested in the code, while the Jackson method can be used to identify the types of data which should be used to construct tests for those functions.

Question: Method/Means

Result: Technique

Validation: Persuasion

[10] J. C. Huang, "An Approach to Program Testing," ACM Computing Surveys, September 1975, pp.113- 128 [Huang75] This paper introduces the basic notions of dynamic testing

ISSN-2230-9659

based on detailed path analysis in which full knowledge of the contents of the source program being tested is used during the testing process. Instead of the common test criteria by which to have every statement in the program executed at least once, the author suggested and demonstrated by an example, that a better criterion is to require that every edge in the program diagraph be exercised at least once. The process of manipulating a program by inserting probes along each segment in the program is suggested in this paper.

Question: Method/Means

Result: Technique

Validation: Analysis

[11] P. Jalote and Y. R. Muralidhara, "A coverage based model for software reliability estimation," Proceedings of First International Conference on Software Testing, Reliability and Quality Assurance, 1994, pp. 6 –10 (IEEE) [JM94] There exist many models for estimating and predicting the reliability of software systems, most of which consider a software system as a black box and predict the reliability based on the failure data observed during testing. In this paper a reliability model based on the software structure is proposed. The model uses the number of times a particular module is executed as the main input. A software system is modeled as a graph, and the reliability of a node is assumed to be a function of the number of times it gets executed during testing – the larger the number of times a node gets executed, the higher its reliability. The reliability of the software system is then computed through simulation by using the reliabilities of the individual nodes.

Question: Method/Means

Result: Technique

Validation: Analysis

ISSN-2230-9659

[12] J. J. Marciniak, "Encyclopedia of software engineering", Volume 2, New York, NY: Wiley, 1994, pp. 1327-1358 [Marciniak94] A book intended for software engineers, this books gives introductions, overviews, and technical outlines of the major areas in software engineering. A review in to test generators is given where the major types of test case generators are given and their intended purpose and principles are discussed. A review on the testing process is given where the entire process of testing is discussed from planning to execution to achieving to maintenance retesting. All the common terms and ideas are discussed. A review of testing tools is given where the testing tools for each purpose is discussed and a couple for state of the art systems are given.

[13] E. F. Miller, "Introduction to Software Testing Technology," Tutorial: Software Testing & Validation Techniques, Second Edition, IEEE Catalog No. EHO 180-0, pp. 4-16 [Miller81] This article serves as the one of the introductory sections of the book Tutorial: Software Testing & Validation Techniques. A cross section of program testing technology before and around the year 1980 is provided in this book, including the theoretical foundations of testing, tools and techniques for static analysis and dynamic analysis, effectiveness assessment, management and planning, and research and development of soft ware testing and validation. The article briefly summarizes each of the major sections. The article also gives good view of the motivation forces, the philosophy and principles of testing, and the relation of testing to software engineering.

[14] D. Richardson, O. O'Malley and C. Tittle, "Approaches to specification-based testing", ACM SIGSOFT Software Engineering Notes, Volume 14 , Issue 9, 1989, pp. 86 – 96 [ROT89] This paper proposes one of the earliest approaches focusing on utilizing specifications in selecting test cases. In traditional specification-based functional testing, test cases are selected by hand based on a requirement specification, thus makes functional testing consist merely heuristic criteria. Structural testing has the advantage of that the applications can be automated and the satisfaction determined. The authors propose approaches to specification-based testing by extending a wide variety of implementation-

ISSN-2230-9659

based testing techniques to be applicable to formal specification languages, and demonstrate these approaches for the Anna and Larch specification languages.

Question: Method/Means

Result: Technique

Validation: Analysis

[15] S. Redwine & W. Riddle, "Software technology maturation," Proceedings of the Eighth International Conference on Software Engineering, May 1985, pp. 189-200 [RR85] In this paper, a variety of software technologies are reviewed. The technology maturation process by which a piece of technology first gets the idea formulated and preliminarily used, then is developed and extended into a broader solution, and finally is enhanced to product-quality applications and marketed to the public. The time required for a piece of technology to mature is studied, and the actions that can accelerate the maturation process are addressed. This paper serves as a very good framework for technology maturation study.

Question: Characterization

Result: Empirical Model

Validation: Analysis

[16] S. Rapps and E. J. Weyuker, "Selecting Software Test Data Using Data Flow Information," IEEE Transactions on Software Engineering, April 1985, pp. 367-375 [RW85] A family of test data selection criteria based on data flow analysis is defined in this paper. The authors contend that data flow criteria are superior to currently path selection criteria being used in that using the latter strategy program errors can go undetected. Definition/use graph is introduced and compared with a program graph based on the same program. The interrelationships between these data flow criteria are also discussed.

ISSN-2230-9659

Question: Method/Means

Result: Technique

Validation: Analysis

[17] M. Shaw, "Prospects for an engineering discipline of software," IEEE Software, November 1990, pp. 15-24 [Shaw90] Software engineering is still on its way of being a true engineering discipline. This article studies the model for the evolution of an engineering discipline and applies it to software technology. Five basic steps are suggested to the software profession to take towards a true engineering discipline: to understand the nature of expertise, to recognize different ways to get information, to encourage routine practice, to expect professional specializations, and to improve the coupling between science and commercial practice. The significant shifts in research attention of software engineering since the 1960s are also given in this article.

Question: Characterization

Result: Descriptive Model

Validation: Persuasion

[18] L. J. White and E. I. Cohen, "A Domain Strategy for Computer Program Testing," IEEE Transactions on Software Engineering, May 1980, pp. 247-257 [WC80] Domain errors are in the subset of the program input domain, and can be caused by incorrect predicates in branching statements or incorrect computations that affect variables in branching statements. In this paper a set of constraints under which it's possible to reliably detect domain errors is introduced. The paper develops the idea of linearly bounded domains. The practical limitations of the approach are also discussed, of which the most severe is that of generating and then developing test points for all boundary segments of all domains of all program paths.

ISSN-2230-9659

Question: Method/Means

Result: Technique

Validation: Analysis

[19] J. A. Whittaker, "What is Software Testing? And Why Is It So Hard?" IEEE Software, January 2000, pp. 70-79 [Whit00] Being a practical tutorial article, the paper answers questions from developers how bugs escape from testing. Undetected bugs come from executing untested code, difference of the order of executing, combination of untested input values, and untested operating environment. A four-phase approach is described in answering to the questions. By carefully modeling the software's environment, selecting test scenarios, running and evaluating test scenarios, and measuring testing progress, the author offers testers a structure of the problems they want to solve during each phase.

Question: Characterization

Result: Qualitative & Descriptive Model

Validation: Persuasion