Software Testing using Optimization Techniques

Objective of the Proposed Research (Abstract)

1. Identification, characterization and automatic prioritization of test cases in software testing using techniques like control flow analysis, Resources usage, etc.
2. Proposing a new approach for software testing process, optimizing testing effort, path coverage, testing complexity, quality & reliability issues.

Introduction Background and Problem Definition

Software testing is the process of validation and verification of the software product. Effective software testing will contribute to the delivery of reliable and quality oriented software product, more satisfied users, lower maintenance cost, and more accurate and reliable result. However, ineffective testing will lead to the opposite results; low quality products, unhappy users, increased maintenance costs, unreliable and inaccurate results. Hence, software testing is a necessary and important activity of software development process. According to Myers [1] "Software Testing is the process of executing a program with the intent of finding errors". The importance of testing can be understood by the fact that "around 35% of the elapsed time and over 50% of the total cost are expending in testing programs" [2-3]. Software is expected to work, meeting customer's changing demands, first time and every time, consistently and predictably. Earlier software systems were used for back-office and non-critical operations of organizations. Now more and more critical applications are implemented globally. This increased expectation for error-free functioning of software has increased the demand for quality output from software vendors [4]. Software Testing is the process used to help and identify the correctness, completeness, security, reliability & quality of developed software. Testing is a process of technical investigation, performed on behalf of stakeholders, that is intended to reveal quality-related information about the product with respect to the context in which it is intended to operate.A concise history of software testing is given by Gelperin & Hetzel [5].Software development environment consists of series of phases, in all the phases of Software Development Life Cycle (SDLC), software testing is one of the major phases. Bach's test strategy model [6] is responsible for quality factors during the testing process with the guidelines of testing maturity model [7]. Key Process Areas (KPAs) consists of the main quality & reliability parameters, which are to be achieved through software testing process. Quantification of those parameters in software testing will be established during the proposed work of research.

The use of metaheuristic search techniques for the automatic generation of test data has been of great interest among researchers in recent years. Previous attempts to automate the test generation process have been limited and also test data generation (test cases) is a difficult process [8]. Metaheuristic search techniques are of much promise in this regard. Metaheuristic search techniques are high-level frameworks, which utilize heuristics to seek solutions for combinatorial problems at a reasonable computational cost. These techniques have been applied to automate test data generation for structural and functional testing as well as others testing techniques. McMin [8] did the surveys on some of the work undertaken in software testing field, discussing possible new future directions of research in each of its different aspects.

The potential cost savings from handling software errors within a development cycle, rather than the subsequent cycles, has been estimated at nearly 40 billion dollars by the National Institute

of Standards and Technology [9]. This figure emphasizes that current testing methods are often inadequate, and hence reduction of software bugs and errors is an important area of research with a substantial payoff. This is particularly true for the increasingly complex, distributed systems used in many applications from embedded control systems to military command and control systems. These systems may exhibit intermittent or transient errors after prolonged executions that are very difficult to diagnose. Using Genetic Algorithmic Approach in software testing explores strategies that combine automated test suite generation techniques with high volume or long sequence testing. Long sequence testing repeats test cases many times, simulating extended execution intervals. These testing techniques have been found useful for uncovering errors resulting from component coordination problems, as well as system resource consumption (e.g. memory leaks) or corruption. Coupling automated test suite generation with long sequence testing could make them more scalable and effective [9].

A classic question in software development is “How much testing is enough?”. Aside from dynamic coverage-based metrics, there are few measures that can be used to provide guidance on the quality of an automatic test suite as development proceeds. Software Testing and Reliability Early Warning (STREW) project [10] contributed static metric suite which provides a developer with indications of changes and additions to their automated unit test suite and code for added confidence that leads to high product quality.

Software testing by both developers and dedicated quality assurance staff is one way to uncover flaws. Automated test generation techniques can be used to augment the process, free of the cognitive biases that have been found in human testers. Breeding software test cases using genetic algorithms as part of a software testing cycle has been attempted [11]. An evolving fitness function that relies on a fossil record of organisms results in interesting search behaviors, based on the concepts of novelty, proximity, and severity. A case study that uses a simple, but widely studied program is used to illustrate the approach. Several visualization techniques are also introduced to analyze particular fossil records, as well as the overall search process. A framework that distinguishes between absolute and relative fitness functions is used to organize past research and characterize this work’s reliance on a relative or changing fitness function. In particular, the genetic algorithm includes a fossil record that records past organisms, allowing any current fitness calculations to be influenced by past generations. Three factors are developed for the fitness function: novelty, proximity, and severity. The interplay of these factors produces fairly complex search behaviors in the context of an example triangle program used in past software testing research. Lastly, several techniques for fossil record visualization are developed and used to analyze different fitness function weights and resulting search behaviors [11].

… …

Gaps in Existing Research (problems)

There is a gap between generation of test suite, reuse test suite, and test case prioritization. Companies generally save the test suites for reuse, because it accounts for almost half of the maintenance cost. Hence, there is a need to maximize the effectiveness for different test cases, which will provide the test team with sound test case prioritization at a minimum cost. There is no exhaustive method till date to find out suitable test cases.
For search-based (metaheuristic approach) structural testing, there are still gaps involving flag and enumeration variables; unstructured control flow; and state behavior. Furthermore, there may be a variety of other reasons as to why test data can not be found with ease for program structures using search-based techniques.

There is a research gap in the area of search-based functional testing compared to structural testing. Functional tests can be derived from different forms of specification. For tests derived in this way, a present barrier to complete automation is the fact that a mapping needs to be provided from the abstract model of the specification to the concrete form of the implementation. For system tests, a potential problem is the size of the search space.

Effort on Software Testing process is around 40% (SEI) of the total development cost [8]. Justification of this effort in terms of money, staffing level, and resources required, etc. is not clear to the customer. It is proposed to provide proper justification for the efforts and cost involved. Statistical techniques with related statistical software will be helpful in determining these parameters.

A reliable and accurate estimate of software development effort had always been a challenge for both the software industry and academia. We propose to improve software-testing effort estimation with suitable optimization techniques. Metaheuristic approach may also be helpful to improve software-testing effort estimation. Impact of software test process in software quality and reliability requires proper quantification and measurement. We propose to devise a framework for quality and reliability parameters to check and ensure bug free, efficient and reliable software.

Due to the growing importance of testing management for business success, it is increasingly employed to enable organizations to achieve sustainable competitive advantages. However, no approach has been developed yet which allows organizations to determine their current state of testing management on a process and product level, which can derive necessary steps for further development. To fill these gaps, we propose to develop new mathematical models.
There is no universal approach to measure the complexity of software testing. Effort will we made to find out a final size formula to measure the complexity of software testing.

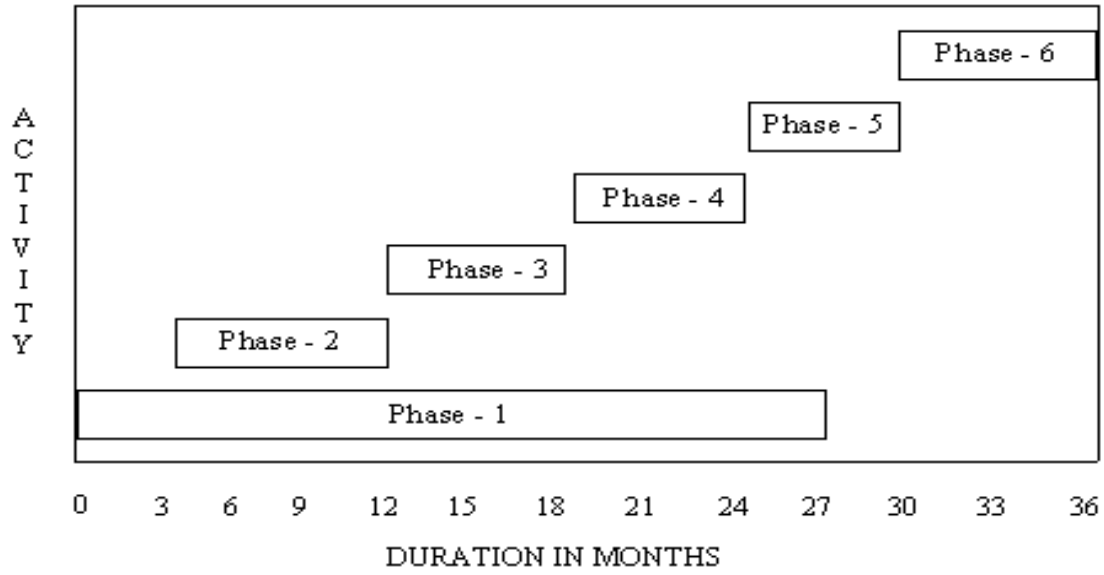
Proposed Solution with Technical Details

Phase – 1: *Literature survey on software testing.*
Detailed literature study will be done to update information regarding quality & reliability issue aspects in Software testing.
Phase–2: *Survey for software testing process. (Regarding quality and reliability of software)*
A survey will be conducted for collecting secondary data for software testing process through questionnaire, feedback etc. Also data will be collected online from International Software Benchmarking Standards Group (ISBSG)
Phase–3: *Identify and analyze software testing parameters on the basis of collected data.*
Analytical and empirical study of some specific cases would be conducted. In this phase, we conduct test suite, test plan, test case priority, defect analysis and develop mathematical models.
Phase–4: *Minimize the testing effort &analyze software testing process for software quality &reliability quantification.*
Software-testing process will be quantified and impact of quantified steps on the overall quality of the software will be decided. Software testing efforts will be optimized in this phase using suitable optimization techniques.
Phase–5: Software test automation, new strategy (testing), complexity, and quality parameters will be analyzed based on data and techniques in this phase. Based on mathematical models developed, software testing cost and process, complexity, quality and reliability will be measured. Justification will be carried out in this phase.
Phase–6: *Discussion, comparable study, verification and validation of results, Thesis writing and submission.*

The final phase will involve description of our research and justification of our work in the proposed field of software testing. Future directions and limitations of the research work would be clearly specified in this phase. Also thesis will be written and submitted.

Project Scope, Deliverables, Timelines and Success Criteria/Metrics

This project is useful for both academics and software industry, because this project gives the theme of saving money, man power, and time and client satisfactions criteria.



Existing infrastructure, required funding, source of resources/funding etc

For this project I need minimum 1lakh up to 2 lakh (depend upon my need).
Till now I don't have any source of resource(except of institute lab) but I will try my best because I need some testing tool, some on line data and some software's ,also I will send some papers to different conferences and journals ,for this I need a money.
Since I am hardworking researcher so I think at least I will publish 8 papers in this project in different journals

Project Team and their background/skills

1. G.Raghurama Ph.D. (Indian Institute of Science (IISc), Bangalore), Msc (physics) IIT MADRAS. More than 20 years of experience
2. Praveen Ranjan Srivastava M.Tech (software engineering) (M.N.N.I.T ALLAHABAD), M.C.A .6 years of experience.

References

1. Myers G. J., The Art of Software Testing, 1st Edition, John Wiley and Sons, NY, USA, 1979.
2. Beizer, B., Software Testing Techniques, 2nd. Edition, Van Nostrand Reinhold, USA 1990.

3. M. Harrold, Testing: A Roadmap, International Conference on Software Engineering, (ICSE 00), Limerick, Ireland, 2000.
4. Srinivasan D., Gopalaswamy R., Software Testing: Principles and Practices, 1st Edition, Pearson Education, New Delhi India, 2006.
5. Gelperin D., Hetzel B., the Growth of Software Testing, Communications of the ACM, 1988, 31: 687-695.
6. Cem Karner, Software Testing as a Social Science, IFIP Working Group10.4, Florida Institute of Technology, Siena Italy, July 2004.
7. Guidelines of Testing Maturity, Professional Tester, Vol. 3, Issue 1, March 2002.
8. Phil McMin, Search-based Software Test Data Generation: A Survey, Proceedings in Software Testing, Verification and Reliability, 2004, 14:105-156.
9. D. J. Berndt and A. Watkins, High Volume Software Testing Using Genetic Algorithms, Proceedings of the 38th (IEEE) Hawaii International Conference on System Sciences, Waikoloa, Hawaii, Jan 2005.
10. Nachiappan Nagappan, Laurie Williams, Jason Osborne, Mladen Vouk, Pekka Abrahamsson: Providing Test Quality Feedback Using Static Source Code and Automatic Test Suite Metrics, Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering, Chicago, 2005, pp85 - 94.
11. D. Berndt, J. Fisher, L. Johnson, J. Pinglikar, and A. Watkins: Breeding Software Test Cases with Genetic Algorithms, Proceedings of the 36th (IEEE) Hawaii International Conference on System Sciences, Waikoloa, Hawaii, Jan 2003.
12. Mark Last, Menahen Friedman, Abraham Kandel, the Data Mining Approach to Software Testing, Proceedings of KDD, 2003, ACM Press, Washington.
13. Roger S.Pressman, Software Engineering: A Practitioner's Approach, 6th Edition, McGraw-Hill International Edition, 2005.
14. Sommerville, Software Engineering, 7th Edition, Pearson Education, 2005.
15. National Institute of Standards & Technology, The Economic Impacts of Inadequate Infrastructure for a Software Testing Planning Report, USA, May 2002.
16. Singh S., Kotze P., An Overview of Systems Design and Development Methodologies with Regard to the Involvement of Users and Other Stakeholders, Proceedings of SAICSIT 2003, SA, September 2003, pp. 37-47.
17. Burns T., Klashner R., Software Development and Analysis: A Cross-Collegiate Analysis of Software Development Course Content, Proceedings of the 6th Conference on Information Technology Education, USA, ACM Press, October 2005, pp. 333-337.
18. John D.Musa, Software Reliability Engineering, 2nd Edition, Tata McGraw-Hill Edition, 2005.
19. Bluvband, Rishon-Lezion, Reliability Centered Software Testing, Proceedings Annual Reliability and Maintainability Symposium, Seattle, USA, 2002.
20. Mark Last, Menahen Friedman, Abraham Kandel, The Classification Approach to SoftwareTesting, Proceedings of KDD-2003, ACM Press, Washington.
21. Hema Srikanth, Laurie Williams, On the Economics of Requirements-Based Test Case Prioritization, EDSER05, ACM Press, St Louis, Missouri, USA, May 2005.
22. H. Srikanth, Requirements Based Test Case Prioritization, Student Research Forum in 12th ACM SIGSOFT Symposium, 2004.
23. J. Karlsson, K. Ryan, A Cost-Value Approach for Prioritizing Requirements, IEEE Software Transaction, 1997, 14: 67-74.
24. B. Boehm, Value-Based Software Engineering, ACM Software Engineering Notes, 2003, 28: 3-6.

25. Eugenia Diaz, Javier Tuya, Raquel Blanco, Automated Software Testing Using a Metaheuristic Technique Based on Tabu Search, Proceedings of the 18th IEEE International Conference on Automated Software Engineering, Canada, October, 2003.
26. Tracey N., Clark J., Mander K., Automated Program Flaw Finding Using Simulated Annealing, International Symposium on Software Testing and Analysis, ACM/SIGSOFT, 1998.
27. Francisca Emanuelle Vieira, Ronaldo Menezes, M arcio Braga, Using Genetic Algorithms to Generate Test Plans for Functionality Testing, ACM SE '06, Florida, USA, March, 2006.
28. Kai-Yuan Cai, T.Y. Chen, Yong-Chao Li, Wei-Yi Ning, Y. T. Yu, Adaptive Testing of Software Components, SAC'05, ACM Press, New Mexico, USA, March, 2005.
29. Kai-Yuan Cai, Yong-Chao Li, W. Eric Wong, Hai Hu, Wei-Yi Ning, Optimal and Adaptive Testing with Cost Constraints**,** *AST'06*, ACM Press, Shanghai, China, May, 2006.,
30. Susan Khor, Peter Grogono**,** Using a Genetic Algorithm and Formal Concept Analysis to Generate Branch Coverage Test Data Automatically, Proceedings of the 19th IEEE International Conference on Automated Software Engineering (ASE'04), Linz, Austria, September, 2004.
31. Rees, Coolen, Goldstien, Woof, Managing the Uncertainties of Software Testing: Bayesian Approach, Proceedings 14th Advanced in Reliability Technology Symposium, Manchester, November 2000.
32. I. F. Barcelos Tronto, J. D. Simoes da Silva, N. SantAnna, The Artificial Neural Networks Model for Software Effort Estimation, INPE , 2006, Vol. 1, pp. 2-21.
33. Lakshmi S.Iyer, Babita Gupta, Nakul Johri, Performance Scalability and Reliability Issues in Web Application, Industrial Management and Data System, 2005, 105:561-576.
34. Sun-jen Huang, Nan-Hsing Chiu, Optimization of Analogy Weights by Genetic Algorithm for Software Effort Estimation, Information and Software Technology, 2006, 48:1034-1045.
35. Martina Marre, Antonia Bertolino, Using Spanning Sets for Coverage Testing, IEEE Transactions on software engineering, 2003,29: 974-984.
36. Ahilton Barreto, Marcio Barros, Claudia Werner; Staffing A Software Project: A Constraint Satisfaction Approach, EDSER'05, ACM Press, St. Louis, Missouri, USA. May 2005.
37. Christian W. Dawson, An Artificial Neural Network Approach to Software Testing Effort Estimation, Transaction on Information and Communication Technologies (WIT Press), 1998, U.K.
38. Wagner S., Seifert T., Software Quality Economics for Defect-Detection Techniques Using Failure Prediction, Proceedings of the Third Workshop on Software Quality, 2005, ACM Press , St. Louis, Missouri, 2005 , pp. 1-6.
39. Green D., DiCaterino A., A Survey of System Development Process Models, Center for Technology in Government, NY, February 1998.
40. Liu C., Platform-Independent and Tool-Neutral Test Descriptions for Automated Software Testing, Proceedings of the 22nd International Conference on Software Engineering, Limerick, Ireland, ACM Press, New York, June 2000, pp.713-715.
41. Mark L., Friedman M., Kandel A., Industrial/Government Track: The Data Mining Approach to Automated Software Testing, Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, D.C., August 2003, ACM Press, pp. 388-396.

42. Nigel Tracey, John Clark, Keith Mander, Automated Program Flaw Finding Using Simulated Annealing**,** ISSTA 98, ACM Press, Clearwater Beach Florida, USA.,1998.
43. Cyrille Artho, Howard Barringer, Allen Goldberg, Klaus Havelund, Sarfraz Khurshid, Mike Lowry, Corina Pasareanu, Grigore Rosu, Koushik Sen, Willem Viscera, Rich Washington, Automated Testing Using Symbolic Model Checking and Temporal Monitoring, Journal of Theoretical Computer Science, March 2004.
44. Stephen H. Edwards, Black-Box Testing Using Flowgraphs: An Experimental Assessment of Effectiveness and Automation Potential: Software Testing, Verification and Reliability, 2000, 10: 249-262.
45. Fevzi Belli, Christof J. Budnik, Minimal Spanning Set for Coverage Testing of Interactive Systems, Proceedings of International Colloquium on Theatrical Aspect and Computing (ICTAC), Springer Verlag, Berlin ,New York, June 2004.
46. Fevzi Belli, Christof J. Budnik, Towards Minimization of Test Sets for Coverage Testing of Interactive Systems, Software Engineering (SE), Essen, Germany, March 2005.
47. Hui Zeng, David Rine, Estimation of Software Defects Fix Effort Using Neural Networks, Proceedings of the 28th(IEEE) Annual International Computer Software and Applications Conference (COMPSAC'04),USA, 2004.
48. Hassan Pournaghshband, Shahriar Movafaghi, Asaleh Sharifi, Software Quality and Testing, Proceedings in World Congress in Computer Science, Computer Engineering, and Applied Computing (WORLDCOMP'06), USA, 2006
49. Weyuker E., Evaluating Software Complexity Measure, IEEE Transaction on Software Engineering, 1988, 14:1357-1365.
50. M**.** Rauterberg, S. Schluep and M. Fjeld, Modeling of Cognitive Complexity with Petri Nets: An Action Theoretical Approach, Cybernetics and Systems 98, Wien: Austrian Society for Cybernetic, 1998, 2: 842-847.
51. Manish Agrawal, Kaushal Chari, Software Effort, Quality, and Cycle Time: A Study of CMM Level 5 Projects, IEEE Transaction of Software Engineering, 2007, 33: 145-156.