Journal of Theoretical and Applied Information Technology
10th February 2015. Vol.72 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

A STUDY ON HUMANIZING SOFTWARE TEST EFFORT AND QUALITY

Dr N.SRINIVASAN
Professor, Department of Computer Applications
Sathyabama University,Chennai

E-Mail: professorsrini@gmail.com

ABSTRACT

For improving software development processes with the goal of developing high-quality software within budget and planned cycle time, Capability Maturity Model (CMM) has become a popular methodology. Prior investigation focusing on CMM level 5 projects, has identified many factors as determinants of software development effort, quality, and cycle time. Using a linear regression model based on data collected from different CMM level 5 projects of reputed organizations, that high levels of process maturity, as indicated by CMM level 5 rating, reduce the effects of most factors that were previously believed to impact software development effort, quality, and cycle time were found. The only factor found to be significant in determining effort, cycle time, and quality was software size. Testing is more than just debugging. The purpose of testing can be quality assurance, verification and validation, or reliability estimation. Particularly regression testing is an expensive, but important, process. Unfortunately, there may be insufficient resources to allow for the re execution of all test cases during regression testing. In this situation, test cases are needed to be prioritized. Regression testing improves the effectiveness of regression by ordering the test cases so that the most beneficial are executed first. There are many studies on regression test case prioritization which mainly has focuses on Greedy Algorithms(GA). However, it is known that these algorithms may produce suboptimal results because they may construct results that denote only local minima within the search space. By contrast, meta heuristic and evolutionary search algorithms aim to avoid such problems. This paper addresses the problems of choice of fitness metric, characterization of landscape modality and determination of the most suitable search technique to apply. The empirical results replicate previous results concerning GA. The results show that GA perform well, although Greedy approaches are surprisingly effective given the multimodal nature of the landscape.

Keywords: Capability Maturity Model (CMM).Greed Algorithms (GA), Kilo Source Lines Of Code (KSLOC), Capability Maturity Model Integration (CMMI), Function Points (FP), Total Quality Management (TQM).

1. INTRODUCTION

The main goal of every software development organization is to develop software to meet clients functional needs with acceptable levels of quality, within schedule and cost estimates,. For this, two major contributions are focused. First, it is necessary to identify key project factors such as software size that determine software project development outcomes for projects. Second, it is to provide benchmarks for effort and quality based on project data. The results suggest that estimation models based on project data are portable across multiple organizations. The object-oriented software development process is increasingly used for the construction of complex distributed systems. In [3], behavior models have long been recognized as the basis for systematic approaches to requirements capture, specification, design, simulation, code generation, testing, and verification. In general, conformance testing of concurrent applications, testing of all possible invocation orderings is unrealistic due to a combinatorial explosion in the number of orderings permitted by the specification. User-defined test objectives constitute a way of limiting the number of test cases to be produced by test synthesis from a specification. Test objectives can be described in the form of high-level test scenarios, which are then easily understood as behavioral test patterns by developers. The advantages of using test case synthesis according to test objectives for both centralized and distributed applications are the following:

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

1.1 Productivity Gain

A test objective specifies the essential aspects of a test, independent of low-level design and implementation choices. While defining a high-level test scenario is not difficult when the main classes are identified, refining and adapting it to the final software product is a difficult process. Automating the completion of the test objective with the low-level design details obtained from the UML model holds out the promise of significant productivity gain. The main expected behaviors can easily be represented as test objectives. Such test objectives can be derived from use case scenarios, contributing to the overall consistency of the development process.

1.2 Version/Product Independence

Test objectives can be chosen to be independent of software versions and variants. This is particularly important in a product-line context, since it enables generic test objectives to be defined for an entire product line. Regression Testing is frequently applied but expensive maintenance process that aims to (re)verify modified software. The principle that the element with the maximum weight is taken first, followed by the element with the second-highest weight, and so on, until a complete, but possibly suboptimal, solution has been constructed. Greedy search seeks to minimize the estimated cost to reach some goal. It is simple, but in situations where its results are of high quality, it is attractive because it is typically inexpensive both in implementation and execution time. The 2-Optimal GA is an instantiation of the K-Optimal Greedy Approach [12] when K ¼ 2. The K-Optimal approach selects the next K elements that, taken together, consume the largest part of the problem. In the case of K-Optimal Additional Greedy, it is the largest remaining part of the problem that is selected. In this study, a 2-Optimal Additional GA was used.

2. LITERATURE REVIEW

The impacts of the factors from prior research, which have been used to estimate development effort and quality is summarized.

2.1 Software Development Effort

Software development effort typically includes human effort expended for high-level design, detailed design, coding, unit testing, integration testing and customer acceptance testing. Effort is often regarded as a surrogate for software development cost since personnel cost is the dominant cost in software development. Many models is such as COCOMO [7], PRICE-S [8], ESTIMACS [9], SEER-SEM [10] have been developed to estimate software development costs. Effort-estimation models such as COCOMO primarily use the number of source lines of code (SLOC) as the basis for effort estimation [10]. Thus, effort in man-months is expressed as a function of Kilo Source Lines cf Code (KSLOC). The COCOMO II model, which is the current version of COCOMO, uses 17 effort multipliers and five scale factors to estimate development effort based on project size. Some of these effort multipliers such as application experience (APEX) and language and tool experience (LTEX) have been found to be insignificant [7]. An alternative metric for SLOC is Function Points (FPs), where the FP is the product of the number of function counts and the processing complexity adjustment [11]. An excellent summary of early models to estimate software development effort has been provided by Mohanty [12]. For a software system with 36,000 lines of machine language executable instructions and well-defined specifications for all independent variables, the various models described in [34] have predicted costs ranging from $300,000 to $2,500,000 and development times ranging from 13 to 25 months. Kemerer [15] compared software estimation models such as COCOMO[7],SLIM[13], FPs[14] and ESTIMACS [16] using data from 15 projects with an average size of a little under 200 KSLOC and found that various estimation models resulted in average effort estimation error rates ranging from 85 to 772 percent. This wide range has been attributed to the differences in productivity between the test environment and the environments in which the models were calibrated, suggesting wide variations in software development outcomes across organizations. Also, differences in application domain influenced the accuracy of these estimates. For example, the projects in the data set used in [16] were primarily business applications with 12 out of 15 projects implemented in COBOL. In contrast, the COCOMO database consisting of 63 projects had only seven business applications [32]. A study by Maxwell et al. [24] found that a relatively small set of factors explained the required effort to complete a software project (size in SLOC), and productivity factors such as application category, language, required reliability and programming practices. This study also found that organization specific models predicted

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

required effort more accurately than general models. It was therefore important to identify organization-level factors that affected software development costs. Banker and Slaughter found that data complexity, defined as the number of data elements per unit of application functionality, significantly increased the enhancement cost of software. Specifically focusing on the impact of capability maturity, improvements in process maturity were found to be associated with reductions in effort [4], [25]. According to an SEI report[38], by adopting Capability Maturity Model Integration (CMMI) based process improvements, Boeing Australia had a 60 percent reduction in work, whereas Lockheed Martin achieved a 30 percent increase in software productivity. In [26] the authors found that process improvements were not significantly related to development costs. Perhaps reflecting on the lack of theory on software estimation, a number of studies found success at effort estimation by simply using analogies to compare the features of a new project with earlier projects [8], [22].

2.2 Software Quality

Initially software quality was defined as conformance to a standard or a specification. Later, the definition was changed to adapt to highly dynamic business environments. In 1991, the International Organization for Standardization adopted ISO 9126 as the standard for evaluating software quality. This standard defines quality as "the totality of features and characteristics of a product or service that bears on its ability to satisfy given needs" ISO 9126 compliments ISO 9001, which deals with the quality assurance of the process used for developing products. A commonly used definition of software quality is the density of post release defects in a software program, which is measured as the number of defects per thousand lines of code [6], Gaffney [18] reported that the best estimator for the number of errors in a software module was the number of lines of code. Krishnan and Kellner [23] also confirmed this finding. Harter and Slaughter[22] found that product complexity significantly lowered software quality, which is somewhat contrary to [18], which did not find software complexity affecting error rates significantly. Banker and Slaughter[37] found that software volatility, defined as the frequency of enhancements per unit of functionality in a given time frame to be a significant predictor of software errors. Data complexity is defined as the number of data elements per unit of application functionality also increased the number of defects. They also found that structured programming techniques moderated the effects of volatility and data complexity on software errors. Using a game-theoretic model Austin suggested that under schedule pressures, developers were likely to compromise on quality. Krishnan et al. [26] found personnel quality, which is measured using peer and supervisor assessments, to be a significant estimator of software quality. They also found that front-end investments, which improved customer requirements analysis, enhanced quality. A number of approaches have been proposed to improve software quality .These include (TQM), Six Sigma [3], and CMM [1]. The basic idea behind in all these approaches is to identify ways to improve quality in a given situation. The relationship between process improvements and quality has also been investigated. The most significant development in this area has been the development of CMM [2], For example, as a software unit at Motorola improved from CMM level 2 to level 5, the average defect density reduced from 890 defects per million assembly-equivalent lines of code to about 126 defects per million assembly-equivalent lines of code [6]. In an empirical study using 33 software products developed over 12 years by an IT company, Harter et al. [4], found that a one percent improvement in process maturity was associated with a 1.589 percent increase in product quality. In another study, Krishnan and Kellner [23] found that process maturity and personnel capability to be significant predictors of the number of defects.

3. RESEARCH METHODOLOGY

3.1 Software Development Effort:

Software development effort typically includes human effort expended for high-level design, detailed design, coding, unit testing and integration. This is attributed to the reduction in rework due to improved processes thereby leading to reduced, testing and customer acceptance testing. Effort is often regarded as a surrogate for software development cost since personnel cost is the dominant cost in software development. Many models such as COCOMO, PRICES, ESTIMACS, SEER-SEM have been developed to estimate software development costs. Effort-estimation models such as COCOMO primarily use the number of (SLOC) as the basis for effort estimation.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

3.2 Software Quality

Software quality is defined as "the totality of features and characteristics of a product or service that bears on its ability to satisfy given needs". It deals with the quality assurance of the process used for developing products. A commonly used definition of software quality is the density of post release defects in a software program which is measured as the number of defects per thousand lines of code. A number of approaches have been proposed to improve software quality. These include TQM, Six Sigma, and CMM. The basic idea behind all these approaches is to identify ways to improve quality in a given situation. The relationship between process improvements and quality has also been investigated [2]. The most significant development in this area has been the development of CMM.

3.3 Design

It can be seen that prior research on software process improvement has focused on finding evidence of reduced effort, improved quality and faster cycle times from software process improvements. The most important factors identified from prior research on software development effort and quality, while focusing specifically on CMM level 5 projects

3.4 Development

It includes effort during high level design, detailed design, coding, unit testing, integration testing and customer acceptance testing.

3.5 Product Quality

The metric used for product quality (QUAL) is defects, which were measured as the total number of defects that escaped to the customer and were detected during the first three months of production use of the software. A period of three months is used, as it is typically the warranty period of newly developed software and the defect data for the first three months is generally tracked by software development organizations [2].

3.7 Product Size

The actual lines of codes developed, excluding comments and blank lines are measured in KSLOC to represent product size. Although the use of KSLOC is in line with prior research a limitation of this measure is that it is usually not consistent across programming languages.

3.8 Product Complexity

Product complexity (COMPLX) is measured using two items on a seven point Likert scale, ranging from low to high data complexity and decision complexity.

3.9 Schedule Pressure

Schedule pressure (SP) is defined as the relative compression of the development schedule mandated by management compared to the initial estimate provided by the development team based on project parameters [5]. The size of a team at its peak is considered a good proxy for the relative size of the team compared to other projects. Also, the peak team size is easier to measure than the average team size over the life of the team. Therefore, TEAM is measured as the peak team size. SP=(Team estimated cycle-time Management mandated cycle-time)=Team estimated cycle-time.

3.10 Personal Capability

The technical skill of each project team is computed as the mean of the five items used to measure the technical capabilities of the team members [6]. Team skill is calculated as the mean of the three items used to measure individual team skills. Finally, it is also included that an overall item is to obtain the supervisor's average rating of the team member.

3.11 Project Supervisor Experience

To account for the management skills of project supervisors, their experience can be used in the software industry (INDEXP), as well as their experience in managerial roles (MGROL) within the industry. Both measures are used to examine the impact of managerial quality on software outcomes [7].

3.12 Greedy Algorithm

A GA is an implementation of the "next best" search philosophy. It works on the principle that the element with the maximum weight is taken first, followed by the element with the second-highest weight, and so on, until a complete, but possibly suboptimal, solution has been constructed. Greedy search seeks to minimize the estimated cost to reach some goal. It is simple, but in situations where its results are of high quality, it is attractive because it is typically inexpensive both in implementation and execution time. Consider the example of statement coverage for a

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

program containing m statements and a test suite containing n test cases. For the GA, the statements covered by each test case should be counted first, which can be accomplished in O(mn) time, then the test cases should be sorted according to the coverage. In the second step, quick sort can be used, thereby increasing the time complexity by O(n log n). Typically, m is greater than n, in which case, the cost of this prioritization is O(mn). The Additional GA requires coverage information to be updated for each unselected test case following the choice of a test case. Given a program containing m statements and a test suite containing n test cases, selecting a test case and readjusting coverage information has cost O(mn) and this selection and readjustment must be performed O(n) times. Therefore, the cost of the Additional GA is $O(mn^2)$ Note that, after 100 percent coverage has been achieved, there are possible remaining unprioritized test cases that cannot add additional coverage. These remaining test cases could be ordered using any algorithm. In this study, the remaining test cases were ordered by reapplying the same Additional GA. However, after 100 percent coverage has been achieved, no further fitness improvement will be possible. In the case of K-Optimal Additional Greedy, it is the largest remaining part of the problem that is selected. In this research, a 2-Optimal Additional GA was used, hereinafter referred to as 2-Optimal" Algorithm: The "2-Optimal" GA for shortness. The K-Optimal approach has been studied in the area of heuristic search to solve the Traveling Salesman Problem (TSP) that is defined as "find the cycle of minimum cost that visits each of the vertices of a weighted graph G at least once". Extensive experiments suggest that 3-optimal tours are usually within a few percent of the cost of optimal tours for TSP. As shown by Skiena, for K >3, the computation time increases considerably faster than solution quality. The 2-Optimal approach has been found to be fast and effective .Again, consider statement coverage: The 2-Optimal Algorithm updates coverage information for each unselected test case following the choice of each pair of test cases. Given a program containing m statements and a test suite containing n test cases, selecting a pair of test cases and readjusting coverage information has cost $O(mn^2)$ and this selection and readjustment must be performed O(n) times. Therefore, the time complexity of the 2-Optimal Algorithms is $O(mn^3)$.

3.13 Generic Algorithm

In Generic Algorithm procedure, which includes initializing a population P, evaluating individuals, selecting pairs of individuals that are combined and mutated to generate new individuals, and forming the next generation. The search proceeds through a number of generations until the termination condition has been met. The initial population is a set of randomly generated individuals. Each individual is represented by a sequence of variables / parameters (called genes),known as the chromosome. The chromosome encodes a possible solution to a given problem. The encoding can take many forms, for example, binary, real-valued, or character-based. A biased selection depending on the fitness value decides which individuals are to be used as the "parents" for producing the next generation. Crossover is a genetic operator that combines two individuals (the parents) to produce a new individual (the offspring). A probability of crossover determines whether crossover should be performed. The mutation operator alters one or more gene values in the individual, depending on the probability of mutation. A GA is not guaranteed to converge upon a single solution. The termination condition is often specified as a maximal number of generations, or as a given value of the fitness function that is deemed to be sufficient. There were 1000 small test suites and 1000 large test suites in many programs. All test suites were obtained from the same infrastructure as the programs. In order to reduce the computation time for the experiments, without significant loss of generality, half of these test suites were used in the experiments. Box plots of the fitness metrics APBC, APDC, and APSC for all programs with small test suites. Each row of subfigures indicates the results for one program. They increase in program size, the differences between the algorithms become more evident. For small programs, the algorithms show almost identical performance. The mean fitness value for each program revealed that the GA is the worst and the Genetic Algorithm is slightly better than the others. First, consider the small programs with small test suites. The ANOVA results for these experiments are summarized.

4. ANALYSIS AND DISCUSSIONS

Table 4.1

Table 4.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

4.1 ANOVA Analysis and LSD Multiple Comparisons

The corresponding results from Table 4.1 and Table 4.2 for the programs, with large test suites are given. This data shows that the same results as those produced from programs with small test suites. That is, there is no significant difference between Additional Greedy and the GA for small programs and no significant difference between Additional Greedy and the 2-Optimal Algorithm.

Table 4.3

Table 4.4

5. CONCLUSION

In this paper, by considering the previous research a method and a tool is proposed for the automated production of test cases from state-based design models of distributed software guided by test objectives. From the practical point of view, the specific contribution is in motivating the test generation process with behavioral test patterns, i.e., reusable (and incomplete) testing scenarios. Overall, the results indicates that the adoption of highly mature software development processes during software development reduced the significance of many factors such as personnel capability ,requirements pecifications requirements volatility and so forth. From this it can be concluded that increased adoption of best practices by client organizations that were to a great degree influenced by the software development organizations, thereby leading to well-defined requirements, and software development organizations leveraging their expertise from prior engagements in assisting clients in requirements gathering and specification. The data and analysis indicate that the Greedy Algorithm performs much worse than Additional Greedy, 2-Optimal, and Genetic Algorithms overall. Also, the 2-Optimal Algorithm overcomes the weakness of the Greedy Algorithm and Additional Greedy Algorithm referred to by previous authors. However, the experiments indicate that, in terms of effectiveness, there is no significant difference between the performance of the2-Optimal and Additional Greedy Algorithms. This suggests that, where applicable, the cheaper-to-implement and execute Additional Greedy Algorithm should be used. The choice of coverage criterion does not affect the efficiency of algorithms for the test case prioritization problem. The size of the test suite determines the size of the search space, thereby affecting the complexity of the test case prioritization problem. The size of the program does not have a direct effect, but increases the difficulty of computing fitness values.

REFERENCES

[1] L.C. Briand, Y. Labiche, and J. Cui, "Automated Support for Deriving Test Requirements from UML Statecharts," J. Software and Systems Modeling, vol. 4, no. 4, Nov. 2005.

[2] S. Burton, J. Clark, and J. McDermid, "Automatic Generation of Tests from Statecharts Specifications," Proc. First Workshop Formal Approaches to Testing of Software (FATES '01), E. Brinksma and J. Tretmans,eds.,Aug.2001,http://www.brics.dk/NS/01/4/BRICSNS-01-4.pdf.

[3] L.D. Bousquet, H. Martin, and J.-M.Je´ze´quel, "Conformance Testing from UML Specifications, Experience Report," Proc. UML 2001 Workshop: Practical UML-Based Rigorous Development Methods, A. Evans, R. France, A. Moreira and B. Rumpe, eds.,2001.

[4] D. Clarke, T. Je´ron, V. Rusu, and E. Zinovieva, "STG: A Symbolic Test Generation Tool," Proc. Eighth Int'l Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS '02), J.-P. Katoen and P.Stevens, eds., Apr. 2002.

[5] W. Damm, B. Josko, A. Pnueli, and A. Votintseva, "A Discrete- Time UML Semantics for Concurrency and Communication in Safety-Critical Applications," Science of Computer Programming, vol. 55, nos. 1-3, 2005.

[6] The Testing and Test Control Notation, Version 3. Part 3: TTCN-3 Graphical Presentation Format (GFT), ETSI ES 201 873, parts 1 to 7 V3.0.0 (2005-03), European Telecomm. Standards Inst., 2005.

[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.B.W. Boehm et al., Software Cost Estimation with COCOMO II. Prentice-Hall,

[8] "True S and Price S: Software Development and Lifecycle Estimating Models," PRICE Systems, 2006.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[9] "CA-Estimacs," Computer Assoc.,2006.

[10] B.W. Boehm, Software Engineering Economics. Prentice-Hall, 1981. Function Point Counting Practices Manual. Int'l Function Point Users Group, 2006.

[12] S.N. Mohanty, "Software Cost Estimation: Present and Future," Software-Practice and Experience, vol.11, pp. 103-121, 1981

[13] C.A. Behrens, "Measuring the Productivity of Computer Systems Development Activities with Function Points," IEEE Trans. Software Eng., vol. 9, no. 6, pp. 648-652, Nov.1983.

[14] H.A. Rubin, "Macroestimation of Software Development Parameters: The Estimacs System," Proc. SOFTFAIR Conf. Software Development Tools, Techniques, and Alternatives, 1983.

[15] R.D. Banker and S.A. Slaughter, "The Moderating Effects of Structure on Volatility and Complexity in Software Enhancement," Information Systems Research, vol. 11, pp. 219-240, 2000.

[16] D.R. Goldenson and D.L. Gibson, "Demonstrating the Impact and Benefits of CMMI: An Update and Preliminary Results," Technical Report CMU/SEI-2003-SR-009, Software Eng. Inst., 2003.

[17] S.S. Vicinanza, T. Mukhopadhyay, and M.J. Prietula, "Software-Effort Estimation: An Exploratory Study of Expert Performance," Information Systems Research, vol. 2,pp. 243-262, 1991.

[18] T. Mukhopadhyay and S. Kekre,"Software Effort Models for Early Estimation of Process Control Applications," IEEE Trans.Software Eng., vol. 18, no. 10, pp. 915-924, Oct. 1992.

[19] C.K. Prahalad and M.S. Krishnan,"The New Meaning of Quality in the Information Age," Harvard Business Rev., vol. 1999, pp. 109-118, 1999.

[20] ISO/IEC 9126-1, 2001, Int'lStandards Organization, 1991.

[21] C. Fox and W. Frakes, "The Quality Approach: Is It Delivering?" Comm. ACM, vol. 40, pp. 25-29, 1997.

[22] D.E. Harter and S.A. Slaughter, "The Cascading Effect of Process Maturity on Software Quality," Proc.Int'l Conf. Information Systems, 2000.

[23] R.D. Austin, "The Effects of Time Pressure on Quality in Software Development: An Agency Model," Information Systems Research, vol. 12, pp. 195-207, 2001.

[24] W.S. Humphrey, "Characterizing the Software Process: A Maturity Framework," IEEE Software, vol. 5, no.3, pp. 73-79, Mar. 1988.

[25] M.C. Paulk et al., "Capability Maturity Model, Version 1.1," IEEE Software, vol. 10, no. 4, pp. 18-27, July 1993.

[26] F.P.J. Brooks, The Mythical Man-Month, second ed. Addison-Wesley, 1995.

[27] M. van Genuchten, "Why is Software Late? An Empirical Study of Reasons for Delay in Software Development," IEEE Trans. Software, Eng., vol. 17, no. 6, pp. 582-590, June 1991.

[28] D.E. Harter, S.A. Slaughter, and M.S. Krishnan, "Benefits of CMM Based Process Improvements for Support Activities-An Empirical Study," Proc. Am. Conf. Information Systems, 1998.

[29] A.J. Albrecht and J.E.J. Gaffney, "Software Function, Source Lines of Code and Development Effort Prediction: A Software Science Validation," IEEE Trans. Software Eng., vol. 9, pp. 639-648, 1983.

[30] J. Baik, "Disaggregating and Calibrating the Case Tool Variable in COCOMO II," IEEE Trans. Software Eng., vol. 28, no. 6, pp. 1009-1022, Nov. 2002.

[31] J.E.J. Gaffney, "Estimating the Number of Faults in Code," IEEE Trans. Software Eng., vol. 10, no. 4, pp. 459-464, July 1984

[32] H. Wohlwend and S. Rosenbaum, "Schlumberger's Software Improvement Program," IEEE Trans. Software Eng., vol. 20,no. 11, pp. 833-839, Nov. 1994.

[33] M. Diaz and J. Sligo, "How Software Process Improvement Helped Motorola," IEEE Software, vol.14, no. 5, pp. 75-81, Sept.-Oct., 1997.

[34] M.S. Krishnan et al., "An Empirical Analysis of Productivity and Quality in Software Products," Management Science, vol. 46,pp. 745-759, 2000.

[35] M.S. Krishnan and M.I. Kellner, "Measuring Process Consistency: Implications Reducing Software Defects," Management Science, vol. 25, pp. 800-815, 1999.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[36] M.C. Paulk, "How ISO 9001 Compares with the CMM," IEEE Software, vol. 12, no. 1, pp. 74-83, Jan.1995.

[37] T. Pyzdek, The Six Sigma Handbook: The Complete Guide for Greenbelts, Blackbelts, and Managers at All Levels. McGraw-Hill,2003.

[38] D.E. Harter, M.S. Krishnan, and S.A. Slaughter, "Effects of Process Maturity on Quality, Cycle Time and Effort in Software Product Development," Management Science, vol. 46, pp. 451-466, 2000.

[39] J.E. Matson, B.E. Barrett, and J.M.Mellichamp, "Software Development Cost Estimation Using Function Points," IEEETrans. Software Eng., vol. 20, pp. 275-287, 1994.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 4.1 ANOVA Analysis For Small Programs With Small Test Suites

	Sum of Squares	Df	Mean Square	F	Significance
Between Groups	15430.153	4	3857.538	776.233	0.000
Within Groups	149062.028	29995	4.970	-	-
Total	164492.181	29999	-	-	-

Table 4.2 ANOVA Analysis For The Large Program (Space) With Small Test Suites

	Sum of Squares	Df	Mean Square	F	Significance
Between Groups	75903.588	4	18975.857	2904.341	0.000
Within Groups	48969.569	7495	6.534	-	-
Total	124873.157	7499	-	-	-

Table 4.3 Multiple Comparisons (Least Significant Difference) For Small Programs With Small Test Suites

Algorithm X	Algorithm Y	Mean-diff (X-Y)	Significance
Greedy	A-Greedy	-1.88928(*)	000
	2-Optimal	-1.57361(*)	000
	HC	-1.01015(*)	000
	GA	-1.91488(*)	000
Greedy	A-Greedy	-1.88928(*)	000
	2-Optimal	-.31567(*)	000
	HC	-.87913(*)	000
	GA	-.02560 (*)	.529
2-Optimal	A-Greedy	1.5736(*)	000
	2-Optimal	-.3157(*)	000
	HC	-.3635(*)	000
	GA	-.3413(*)	000

Table 4. 4 Multiple Comparisons (Least Significant Difference) For The Large Program (Space) With Small Test Suites

Algorithm X	Algorithm Y	Mean-DIFF (X-Y)	Significance
Greedy	A-Greedy	7.34957(*)	000
	2-Optimal	7.41580(*)	000
	HC	.76127(*)	000
	GA	5.14745(*)	000
Greedy	A-Greedy	-7.34957(*)	000
	2-Optimal	-0.6623 (*)	-478
	HC	-6.65452(*)	000
	GA	-2.26835 (*)	000
2-Optimal	Greedy	-5.1474(*)	000
	A-Greedy	-2.2021 (*)	-478
	HC	-2.26201(*)	000
	GA	-2.2635(*)	000
2-Optimal	Greedy	1.91488(*)	000
	A-Greedy	02.560(*)	000
	2-Optimal	.34127(*)	000
	GA	-.9047(*)	000