End to End Software Testing Estimation Framework for Digital Products

Shailesh K S, Dr Anant Koppar,
PESU

Abstract - End to end Software testing includes requirements gathering related to testing, test planning and scripting, test execution and reporting. In this paper we propose a novel end to end software testing estimation framework (STEF) that can be used for various software testing projects. The STEF provides the estimation methods, effort calculation formula, sample guidance values for base lining that can be used for digital testing projects. The STEF also categorizes various testing activities in each of the testing phases and provides complexity scale factors for effective effort estimation. The STEF was used for 3 testing projects to estimate the overall testing effort with the pred (0.3) accuracy of 80%

Keywords - Software Engineering, Estimation, Testing estimation, test requirement estimation, Test Requirements Gathering, Test strategy and design, Test execution, Test analysis, monitoring and reporting, Test quality improvement

I. INTRODUCTION

Software testing is mainly carried out with the intent to detect defects. Out of total software development effort about 35% of effort is spent on software testing and more than half of the overall software development cost is due to testing (Myers, 1979, Harrold, 2000)

In this paper we propose a novel estimation method, "Software testing estimation framework" for digital testing projects that factors in various activities in software testing phase. We have categorized the software activities into four main categories: Test Requirements Gathering, Test strategy and design, Test execution, Test analysis, monitoring and reporting and Test quality improvement consisting of activities that belong to each of the categories.

The Software testing estimation framework provides the estimation guidelines, complexity definition scale and effort calculation formulae for all the categories such as Test Requirements Gathering, Test strategy and design, Test execution, Test analysis, monitoring and reporting and Test quality improvement consisting The project teams can compile the historical data and apply the Software testing estimation framework to predict the testing effort.

II. LITERATURE REVIEW AND RELATED WORK

Generic software estimation models include COCOMO (Boehm, 1981), Function Point/FP (Albrecht), COCOMO II (Boehm et al., 2000), SEER-SEM (Jenson 1984), SLIM (Putnam & Myers, 1992), PRICE-S (Frank Freiman), Delphi (Boehm, 1984), Rule-based/Rule of thumb, Use case point (Ochodek, Nawrocki, Kwarciak), Work breakdown structure (Jorgensen 2004), Planning poker, Story point estimation, learning (Goldberg 1989), Case based reasoning (Aamodt & Plaza 1994), Analogy based estimation, Select Estimator, top-down estimation, bottom-up estimation, price-to-win, Stepwise ANNOVA (Basha, 2010), Ordinary Least squares (Griffiths et.al.1993). Other software estimation method are machine learning-based estimation (Mair et al.), fuzzy logic based estimation (Gray, 1997), genetic programming (Burgess et al. 2001) and expert based estimating method (Jørgensen, 2004).

For software testing estimation, there are various state of the art estimation methods. Nageshwaran, 2001 categorized the main methods for test case execution as ad-hoc methods (based on budget and other subjective parameters), percentage of total development effort and function point estimation. The key data used for estimating the test effort estimation are use cases (Almedia et al., Nageshwaran, 2001, Xiaochum et al., Zhou et al.), source code (Kushwaha et al., Thomas Mccabe), test specification (Almedia et al.,), software requirement specification (Ashish et al.), UML Class diagram (Baudray et al.) and functional requirements (Veenendaal et al.).

One of the test effort estimation methodsinvolves estimating the size and complexity of test cases which uses test specification written in controlled natural language (Aranha and Borba, 2007); the authors used the execution points obtained from functional and non-functional requirements of the test cases Another popular method is to calculate test effort in v-model development lifecycle based on use cases and adjusting the weights and environment factors (Nageshwaran, 2001); Use case parameters such as actors and environment factors such as tools, test inputs, interfaces, are used to calculate test effort (Erika Almeida et. Al). Another popular estimation technique is to use the execution points by converting the code to test cases and other productivity factors to estimate the testing effort (Aranha et al. 2007, Rajan et al. 2007, Aranha et al. 2007, Silva et al. 2009). Kushwaha et al., demonstrated that cyclomatic complexity is an indicator of software complexity and is an important metric for calculating test effort. Veenendaal et al. used the test point based on functional requirements. Deckkers adopted test point analysis that uses software size, test strategy and productivity for test effort estimation. Guerreiro e Silva et al. uses data analysis, hypothesis, evaluation and efficiency for test effort estimation.

Gaps with state of the art techniques - The main gap with state of the art software maintenance estimations models are challenges in calculating effort for software size and

productivity (Martin, 1983). Other key gaps in the state of the art software testing methods are as follows:

- State of the art estimation methods do not estimate end to end testing lifecycle that includes Test Requirements Gathering, Test strategy and design, Test execution, Test analysis, monitoring and reporting and Test quality improvement.
- State of the art methods do not consider the adjustment factors such as requirement/domain complexity which is part of modern digital projects.
- State of the art methods do not provide sample guideline effort values that can be used as baseline in the absence of historical effort data.

III. METHOD

In this paper we have proposed an estimation framework "Software testing estimation framework" (STEF) that provides comprehensive coverage for various phases for the testing activities at various lifecycle stages of software testing:

- Test Requirements Gathering to estimate the effort in gathering comprehensive requirements for testing activity.
- Test strategy and design to estimate the activities related to test data modeling, environment setup, test scripts design and such.
- Test execution includes activities such as test script development and manual and automated testing.
- Test analysis, monitoring and reporting such as real-time monitoring of system under test and reporting the test results.
- Test quality improvement involves activities such as testing automation, continuous testing, productivity improvement activities and such that improves the performance of the system and processes.

The STEF is a framework designed to be used across wide variety of testing projects. The STEF provides estimation methodology which can be used along with historical data for estimating testing effort across all lifecycle stages of testing project.

High level steps used in the "Software testing estimation framework" are as follows:

1. Obtain the historical data for various testing activities under test requirements, test strategy and design, test execution, test monitoring and reporting and test quality improvement categories. We have elaborated the factors to categorize the activities into various complexity scales under each of the categories.
2. In the absence of historical data, use the guidance value given by the framework for various categories as a percentage of the overall testing life cycle effort.
3. Use the effort adjustment factors recommended by the STEF framework and use the effort calculation formulas for calculating effort for each of the categories.
4. The overall testing effort is the sum total of effort involved in under test requirements, test strategy and design, test execution, test monitoring and reporting and test quality improvement categories.

In the coming sections we will elaborate the calculation for each of these categories.

Pre-requisites for Software testing estimation framework

For the software testing estimation framework, we need the historical testing effort data for base lining. We need to get the historical data for various categories such as historical effort data for test requirements, test strategy, test execution, test monitoring and test quality improvements.

Test Requirements Gathering effort calculation - During the test requirements phase, we calculate the effort needed to get overall testing requirements. We use test requirements to create test plans and test scripts.

Complexity scale factors for test requirements gathering - The complexity scale factors for various requirements category is given in table 1.

Table1: Complexity scale factors for test requirements gathering

	Complexity Scale factors		
Test requirement Category	**Low**	**Medium**	**Complex**
Functional test requirements	Number of needed test cases is test than 30	Application architecture analysis, Requirement/use case analysis Number of needed test cases is test than 100	Huge number of functional requirements with more than 100 test cases. Includes complex scenarios like services testing, migration testing, batch job testing, business rules testing, workflow testing, process testing, mobile testing
Non-Functional test requirements	Minimal nonfunctional requirements	Less than 50 test cases to cover for non-functional requirements	Complex nonfunctional scenarios related to security, scalability, availability, accessibility, localization, compatibility, performance with strict SLAs. More than 50 test cases to cover for non-functional requirements

Test requirements adjustment factors - The test requirements effort is influenced by various factors which we need to consider while calculating the overall effort. Table 2 provides the rating value for test adjustment factors. We will use this for test requirements effort calculation.

Table 2: Adjustment factors for test requirements

Test requirement adjustment factor	Characteristic	Rating value
Requirement clarity	Highly detailed requirements	0.98
	Requirements need to be detailed	0.95
	Minimal or ambiguous requirements	0.85
Domain knowledge	Testing team has in-depth knowledge of application domain	0.98
	Testing team has moderate knowledge of application domain	0.95
	Testing team lacks knowledge of application domain	0.85
Application complexity	The application under test is highly complex	0.98
	The application under test is moderately complex	0.95
	The application under test is simple	0.90

The overall test adjustment factor score is the product of three adjustment factor values:

$$\boldsymbol{AF_{overall}} = \prod_{i=1}^{3} \boldsymbol{af_i}$$

Where $\boldsymbol{AF_{overall}}$is the overall adjust factor score and $\boldsymbol{af_i}$is therating value for each of the adjustment factors.

Overall Test requirements effort calculation - Overall test requirement effort estimate is given by equation 1:

$$\boldsymbol{Test\_requirement\_effort_s} = \boldsymbol{AF_{overall}} \times \sum_{i=1}^{n} (req\_complexity\_effort_i)$$

Equation 1

Where
$\boldsymbol{Test_requirement_effort}$is the overall effort for test requirements, n is the total number of complexity categories, the $req\_complexity\_effort_i$ is the baseline effort for each of the complexity categories (simple, medium or complex) obtained from the historical data.$\boldsymbol{AF_{overall}}$is the overall adjustment score.
Where $\boldsymbol{Test_requirement_effort}$is the overall effort for test requirements, n is the total number of complexity categories, the $req\_complexity\_effort_i$ is the baseline effort for each of the complexity categories (simple, medium or complex) obtained from the historical data.$\boldsymbol{AF_{overall}}$is the overall adjustment score.

Test strategy and design effort calculation - During the test strategy and design phase, we analyze the workload and historical data and identify the overall test strategy. During this phase we design the testing strategy and create a detailed test plan with schedule, deliverable, roles and plan.
The complexity scale factors for various categories is given in table 3.

Table 3: Complexity scale factors for test strategy and design

	Complexity Scale factors		
Category	**Low**	**Medium**	**Complex**
Workload modeling	Workload model data readily available	Workload model numbers obtained from stakeholder interviews.	Workload needs to be modeled based on the requirements and historical data analysis (user traffic, data volume, transaction rate etc.) Workload model should be obtained from log analysis to understand the traffic pattern, transaction rate etc.
Test case planning	Test tool readily available, No test plan needed.	Test environment setup, Test data setup and transaction data creation, Load test data planning and identification , Check for reusable test cases, Minimal planning needed.	Requirement for automated testing, Test tool selection, Test script design for services testing, migration testing, batch job testing, Test monitoring tool selection, Define testing milestones and schedule; identify roles and dependencies/assumptions and deliverable planning, Risk analysis and contingency planning, Defining automation strategy, Defining test objectives and success metrics.
Tool validation	Testing tools readily available for reuse	Existing tools need to be configured for testing	Conduct proof-of-concept (PoV) to evaluate the testing tools and monitoring tools for the given requirement and test scenarios.

We need to collect the historical data for each of the categories for each of the complexity categories (simple, medium, complex). Complexity scale factors are used to categorize the activities into each of the complexity categories (simple, medium, complex).
Overall test requirement effort estimate is given by equation 2:

$$\boldsymbol{Test\_strategy\_design\_effort_s} = \sum_{i=1}^{n} (category\_complexity\_effort_i)$$

Equation 2

Where
$\boldsymbol{Test_strategy_design_effort_s}$is the overall effort needed for test strategy and design, n is the total number of categories, the $category\_complexity\_effort_i$ is the baseline effort for each of the complexity categories

(simple, medium or complex) obtained from the historical data.

For instance, if the $category\_complexity\_effort_i$ effort for "medium" complex category is 20 person days for "Workload modeling" category from historical data and if the current project is of "medium" complexity for "Workload modeling", category, then we need to consider 20 person days of effort for "Workload modeling" category and similarly calculate the effort for "Test case planning" category and "Tool validation" category. Overall $\boldsymbol{Test\_strategy\_design\_effort}$ is the sum total of effort from "Workload modeling" category, "Test case planning" and "Tool validation" categories

Test execution effort calculation - During test execution stage, we develop the test cases (for manual testing) and test scripts (for automated testing). The developed test cases and test scripts are then executed at various stages of test execution phase.

The complexity scale factors for various categories is given in table 4.

Table 4: Complexity scale factors for test execution

	Complexity Scale factors		
Category	**Low**	**Medium**	**Complex**
Test cases and test script development	Number of test cases < 50 Test cases are readily available for reuse	Number of test cases < 100, Test cases need to be developed for functional testing and basic security testing and basic performance testing, Tool based test script development	Number of test cases > 100, Test cases need to be developed for functional testing, security testing, load testing, stress testing, availability testing, endurance testing, Test scripts need to be developed to automate testing, Manual test script development.
Test cases and test script execution	Execute readily available test cases, Execute/schedule automated test scripts	Execute < 100 functional test cases, Configure automated test cases.	Manually execute > 100 test cases for white box and black box testing, Execute test cases belonging to various testing types such as functional testing, smoke testing, stress testing, failover testing, integration testing, system testing, load testing, multi-device testing, endurance testing, performance testing, security testing, accessibility testing, localization testing, Execute load testing at various loads for various transactions. Multiple rounds and iterations of testing for each test type.

We need to collect the historical data for each of the categories for each of the complexity categories (simple, medium, complex). Complexity scale factors are used to categorize the activities into each of the complexity categories (simple, medium, complex).

Overall test requirement effort estimate is given by equation 3:

$$\boldsymbol{Test\_execution\_effort_s} = \sum_{i=1}^{n} (category\_complexity\_effort_i)$$

Equation 3

Where

$\boldsymbol{Test\_execution\_effort_s}$ is the overall effort needed for test execution, n is the total number of complexity categories, the $category\_complexity\_effort_i$ is the baseline effort for each of the complexity categories (simple, medium or complex) obtained from the historical data.

For instance, if the $category\_complexity\_effort_i$ effort for "medium" complex category is 30 person days for "Test cases and test script development" category from historical data and if the current project is of "medium" complexity for "Test cases and test script development" category, then we need to consider 30 person days of effort for "Test cases and test script development" category and similarly calculate the effort for "Test cases and test script execution" category. Overall $\boldsymbol{Test\_execution\_effort_s}$ is the sum total of effort from "Test cases and test script development category" and "Test cases and test script execution" categories.

Test analysis, monitoring and reporting effort calculation - While executing various test scenarios, the test systems and the applications need to be constantly monitored. Post testing, the test results need to be analyzed and reported to all the concerned teams.

The complexity scale factors for test analysis, monitoring and reporting category is given in table 5.

Table 5: Complexity scale factors for test analysis, monitoring and reporting

	Complexity Scale factors		
Category	**Low**	**Medium**	**Complex**
Test result analysis	Minimal test results analysis needed.	Basic test metrics such as response time, throughput are analyzed.	The test metrics such as response time, throughput, user load specific performance, 90 percentile values need to be analyzed, Performance bottleneck analysis and profiling need to be analyzed to identify the root cause.
Test monitoring	Minimal monitoring needed	Basic system parameters such as CPU	Monitoring tools are setup to monitor various system

		utilization, memory utilization are monitored during testing.	health parameters such as CPU utilization, memory utilization, system throughput, response times at various loads.
Test reporting	Basic test reporting	Reuse of existing reports through configuration	Development of test results dashboard and visualizations to depict all test metrics such as coverage metrics, performance SLAs, defect rate and such, Periodic and iterative reporting, Automated notifications setup upon test completion.

We need to collect the historical data for each of the categories for each of the complexity categories (simple, medium, complex). Complexity scale factors are used to categorize the activities into each of the complexity categories (simple, medium, complex).
Overall test monitoring effort estimate is given by:

$$\boldsymbol{Test\_monitoring\_effort_s} = \sum_{i=1}^{n} (category\_complexity\_effort_i)$$

Equation 4

Where
$\boldsymbol{Test\_monitoring\_effort_s}$ is the overall effort needed for test analysis, monitoring and reporting, n is the total number of complexity categories, the $category\_complexity\_effort_i$ is the baseline effort for each of the complexity categories (simple, medium or complex) obtained from the historical data.
For instance, if the $category\_complexity\_effort_i$ effort for "medium" complex category is 20 person days for "test reporting" category from historical data and if the current project is of "medium" complexity for "test reporting" category, then we need to consider 20 person days of effort for "test reporting" category and similarly calculate the effort for "Test analysis" and "test monitoring" categories. Overall $\boldsymbol{Test\_strategy\_design\_effort}$ is the sum total of effort from "test reporting" category, "Test analysis" and "test monitoring"

Note: *The "test result analysis" category includes activities based on the nature of the testing project. For instance, in a performance testing project the "test result analysis" includes bottleneck analysis, profiling and such performance related activities. For a data migration testing scenario, the "test result analysis" category includes data integrity analysis, data duplication analysis, data mismatch analysis and such.*

Test quality improvement effort calculation - In some of the digital projects, we include pro-active quality improvements for testing activities. These pro-active quality improvements include productivity improvements, testing automation, real-time application monitoring and reporting, knowledge management and such. As test quality improvement is not needed in all digital projects, this is an optional category that STEF provides for large and complex digital testing projects. Various test quality improvement initiatives are as follows:

- **Automation -** The testing team has to automate various test activities such as test execution (regression testing, smoke testing, creation of test suites), test monitoring (system monitoring, performance monitoring) to minimize manual efforts and to improve overall productivity.
- **Test case reusability -** Test team can enhance the reusability through test script parameterization wherein a single test script can be reused across multiple test scenarios (by varying the parameters such as load value, application end point etc.)
- **Productivity -** Improvement/Continuous improvement: In this category testing team minimizes the overall test execution time through automation, reusability and usage of various tools. Adopt parallel testing wherever possible to optimize the test execution time. Test iteratively using continuous integration tools.
- **Knowledge management -** This includes creation of defect and test script knowledge base for easier management of testing activities. The test knowledge base can also be used for test result trend analysis, reporting and such activities.

The effort for pro-active test quality enhancements is normally estimated using the overall complexity of the involved activities and the regular effort estimation methods (such as function point estimation or use case point based estimation). The overall effort for quality improvements is the sum total of effort needed for automation, reusability, Productivity Improvement/Continuous improvement and knowledge management activities:

$$Test\_quality_{effort} = Automation_{effort} + test\_case\_reusability_{effort} + productivity\_improvement_{effort} + knowledge\_management_{effort}$$

Equation 5

Sample testing effort guidance values - For green field digital projects with niche technologies, we won't be having historical data. Even when there is historical data, the application domain, technologies may be different from the current project leading to quality issues in the historical data. In such scenarios the STEF provides a sample guidance value for effort estimation.

The effort guidance value is obtained from 15 digital testing projects. This can be used as a sample baseline in the absence of the correct historical data.

Table 6 Guidance values for testing effort

Testing categories	% of overall test lifecycle effort
Test requirements	15%
Test strategy and design (including test case/test script preparation)	35%
Test execution	35%
Test monitoring and reporting	5%
Test quality improvement	10%

In order to use the sample guidance values, we need to calculate the overall testing effort and then we can use the guidance values given in table 6 to get the approximate effort for each of the test categories. Overall testing effort can be obtained from historical data. For instance, based on the analysis of historical project data we find that overall testing effort is approximately same as overall development effort; then take the overall development effort as the overall testing effort baseline and calculate the effort for individual testing categories (such as test requirements, test execution etc.) using guidance values from table 6.

Overall Software Testing effort calculation - The overall end to end software testing effort is calculated by the sum of testing effort across all testing phases. The overall testing effort is given by the equation 6:

$$\begin{aligned} Test\_execution\_effort_s &= Test\_requirement\_effort_s \\ &+ Test\_strategy\_design\_effort_s \\ &+ Test\_execution\_effort_s \\ &+ Test\_monitoring\_effort_s \\ &+ Test\_quality_{effort} \end{aligned}$$

Equation 6

IV. RESULT

We used the "software testing estimation framework" (STEF) to predict the testing efforts for three complex long running testing projects. The table7provides the details of MRE and MMRE for each of the core testing activities.
The effort prediction of DPMEF for five digital maintenance project are given in table 7:

Table 7 Testing Effort prediction using STEF for 3 projects

Project	Testing project	Actual Effort (person days)	Predicted Effort (person days)	MRE
Test Requirements Gathering				
	Performance testing project	2.6	3.2	0.230769
	Digital commerce project testing	3.4	4.1	0.205882
	Digital portal project testing	1.6	1.4	0.125
Test strategy and design				
	Performance testing project	5.3	5.9	0.113208
	Digital commerce project testing	7.5	8.2	0.093333
	Digital portal project testing	3.2	4.3	0.34375
Test execution				
	Performance testing project	4.6	4.2	0.086957
	Digital commerce project testing	8.3	9.1	0.096386
	Digital portal project testing	4.1	3.9	0.04878
Test analysis, monitoring and reporting				
	Performance testing project	0.7	0.5	0.285714
	Digital commerce project testing	1.2	1.1	0.083333
	Digital portal project testing	0.4	0.6	0.5
Test quality improvement				
	Performance testing project	1.3	0.7	0.461538
	Digital commerce project testing	2.2	1.5	0.318182
	Digital portal project testing	1.1	0.9	0.181818

The MMRE is 0.211 and the pred (0.25) is 73% and pred (0.3) is 80%.

V. DISCUSSION

As we can see from the prediction tables above, the MMRE is 0.187, 0.183, 0.0773, 0.289 and 0.320 for Test Requirements Gathering, Test strategy and design, Test execution, Test monitoring and reporting Test quality improvement activities respectively. On an average, the Software testing estimation framework was able to predict with 21.1% deviation for all four categories.

The average Pred (0.3) for all four categories is 80% indicating that 80% of effort predictions from Software testing estimation framework are within 30% error margin. The MMRE was most optimal for test execution effort and the pred (0.25) is optimal for Test strategy and design and test execution activities. The STEF high effort prediction accuracy for Test strategy and design and test execution activities is due to the easily quantifiable activities and presence of accurate historical data for those activities.

We have also noticed that the prediction error margin is relatively high for the "Test quality improvement" category. The MMRE for "Test quality improvement" category is 0.32 which is highest among all the five categories and pred (0.3) is 30% which is lowest among all the four categories. The main reason for this relatively high error margin for "Test quality improvement" category is due to the variation in efforts of Automation, test case reusability, Productivity Improvement/Continuous improvement, and knowledge management activities from one project to another. Each project has its own goals and tasks under quality improvements category and as a result the historical data is not an accurate predictor of the future effort for quality improvement category.

Threats to validity - The sample testing effort guidance suggested by Software testing estimation framework uses historical data from 15 digital testing projects. The sample guidance values need to be fine-tuned using historical data from larger number of projects. The effort prediction for "Test quality improvements" has highest error margin due to availability of quality historical data. The framework needs to be tested against larger sample set and fine-tuned.

VI. FUTURE SCOPE OF IMPROVEMENT

The Software testing estimation framework needs to be tested against higher number of historical projects and the framework needs to be fine-tuned wherever required.

VII. CONCLUSION

In this paper we defined "Software testing estimation framework" that provides end to end estimation framework for estimating effort for various testing activities in the testing lifecycle stages. The Software testing estimation framework provided estimation template, effort estimation formula, categorized activities, complexity scale factors, adjustment factors for various testing stages such as Test Requirements Gathering, test strategy and design, Test execution, Test analysis, monitoring and reporting and Test quality improvement. The Software testing estimation framework was validated for three testing projects with pred (0.3) of 80% and MMRE of 0.2116.

VIII. REFERENCES

[1]. Ahn, Yunsik, et al. "The software maintenance project effort estimation model based on function points." Journal of Software: Evolution and Process 15.2 (2003): 71-85.

[2]. Ziauddin, S. K. T., & Zia, S. (2012). An effort estimation model for agile software development. Advances in computer science and its applications (ACSA), 2(1), 314-324.

[3]. Boehm, B., Abts, C., & Chulani, S. (2000). Software development cost estimation approaches—A survey. Annals of software engineering, 10(1-4), 177-205.

[4]. Sarro, F., Petrozziello, A., & Harman, M. (2016, May). Multi-objective software effort estimation. In Proceedings of the 38th International Conference on Software Engineering (pp. 619-630). ACM.

[5]. Shepperd, M., & MacDonell, S. (2012). Evaluating prediction systems in software project estimation. Information and Software Technology, 54(8), 820-827.

[6]. Jorgensen, M., & Shepperd, M. (2007). A systematic review of software development cost estimation studies. IEEE Transactions on software engineering, 33(1).

[7]. Kemerer C.F. (1987), "An Empirical Validation of Software Cost Estimation Models," Comm. ACM, vol. 30, no. 5, pp. 416 – 429, May

[8]. Sneed H.M., Huang S. (2007), "Sizing Maintenance Tasks for Web Applications," 11th European Conference on Software Maintenance and Reengineering, 2007. CSMR '07.

[9]. Boehm, B., Abts, C., Brown, A. W., Chulani, S., Clark, B. K., Horowitz, E., Madachy, R., Reifer, D., and

[10]. Steece, B. 2000. Software cost estimation with Cocomo II. New Jersey: Prentice-Hall.

[11]. Boehm, B. W. 1981. Software engineering economics. New Jersey: Prentice-Hall.

[12]. Boehm, B. W. 1984. Software engineering economics, IEEE Transactions on Software Engineering 10(1): 4–21.

[13]. Shepperd, M., Shofield, C., and Kitchenham, B. 1996. Effort estimation using analogy. In International Conference on Software Engineering, pp. 170–178, IEEE Comput. Soc. Press, Los Alamitos, CA, USA, Berlin, Germany.

[14]. Reifer, D. J. 2000. Web development: Estimating quick-to-market software. IEEE Software 17(6): 57–64.

[15]. Jørgensen, M. 2004. A review of studies on expert estimation of software development effort. Journal of Systems and Software 70(1–2): 37–60.

[16]. Idri, A., Abran, A., Khoshgoftaar, T., 2001. Fuzzy Analogy: a New Approach for Software Effort Estimation, In: 11th International Workshop in Software Measurements, pp. 93-101.

[17]. Pfleeger, S. L., Wu, F. & Lewis, R.(2005), Software cost estimation and sizing methods: issues, and guidelines, RAND corporation.

[18]. Jones, C. (2007). Estimating software costs: Bringing realism to estimating. New York: McGraw-Hill Companies.

[19]. Chemuturi, M. (2011). Analogy based software estimation. Chemuturi Consultants.

[20]. Highsmith, J. 2001. History: The Agile Manifesto, http://agilemanifesto.org/history.html.

[21]. Jørgensen, M., and M. Shepperd. 2007. A systematic review of software development cost estimation studies. IEEE Transactionson Software Engineering 33 (1):33-53.

[22]. Lang, M., Conboy, K., & Keaveney, S. (2013). Cost estimation in agile software development projects. In Information Systems Development (pp. 689-706). Springer, New York, NY.

[23]. Basha, S., & Ponnurangam, D. (2010). Analysis of empirical software effort estimation models. arXiv preprint arXiv:1004.1239.

[24]. Buglione, L., & Abran, A. (2007, May). Improving Estimations in Agile Projects: issues and avenues. In

Proceedings of the 4th Software Measurement European Forum (SMEF 2007), Rome (Italy) (pp. 265-274).
[25]. Abrahamsson, P., & Koskela, J. (2004, August). Extreme programming: A survey of empirical data from a controlled case study. In Empirical Software Engineering, 2004. ISESE'04. Proceedings. 2004 International Symposium on(pp. 73-82). IEEE.
[26]. Bhalerao, S., & Ingle, M. (2009). Incorporating Vital Factors in Agile Estimation through Algorithmic Method. IJCSA, 6(1), 85-97.
[27]. Ziauddin, S. K. T., & Zia, S. (2012). An effort estimation model for agile software development. Advances in computer science and its applications (ACSA), 2(1), 314-324.
[28]. Geraci, A., Katki, F., McMonegal, L., Meyer, B., Lane, J., Wilson, P., ... & Springsteel, F. (1991). IEEE standard computer dictionary: Compilation of IEEE standard computer glossaries. IEEE Press.
[29]. Lientz, B.P., and Swanson, E.B. Software Maintenance Management, Addison-Wesley Publishing Company, 1980.
[30]. Aranha, E., & Borba, P. (2007, September). An estimation model for test execution effort. In Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on (pp. 107-116). IEEE.\
[31]. Suresh Nageshwaran, Test Effort Estimation Using USE CASE Points. Quality Week 2001, San Francisco,California USA, 2001
[32]. Erika R. C De Almeida, Bruno T. de Abreu, Regina Moraes. An Alternative Approach to Test Effort Estimation Based on Use Case. IEEE-International Conference on Software Testing Verification and Validation, 2009
[33]. Eduardo Aranha, Filipe de Almeida, ThiagoDiniz, VitorFontes, Paulo Borba. Automated Test Execution Effort Estimation Based On Functional Test Specification. Proceedings of Testing Academic and Industrial Conference Practice and Research Techniques, MUTATION 2007.
[34]. AjithaRajan, Michael W Whalen, Mats P. E. Heimdahl.2007. Model Validation Using Automatically Generated Requirements- Based Tests. 10th IEEE- High Assurance Systems Engineering Symposium, 2007.
[35]. Aranha E, Borba P. Test Effort Estimation Model Based On Test Specifications. Testing : Academic and Industrial Conference- Practice and Research Techniques,IEEE Computer Society, 2007
[36]. AnielGuerreiro e Silva, Bruno T. de Abreu, Mario Jino. A Simple Approach For Estimation of Execution of Function Test Case. IEEE-International Conference on Software Testing Verification and Validation, 2009
[37]. Myers GJ (1979) The Art of Software Testing, 1st Edition, John Wiley and Sons, USA.
[38]. Harrold MJ (2000) Testing: A Roadmap, Proc. of 22nd International Conference on Software Engineering,
[39]. Future of Software Engineering Track, Limerick, Ireland.
[40]. Beizer B (1990) Software Testing Techniques, 2nd Edition, Van Nostrand Reinhold Company Limited, UK.
[41]. Nageswaran S (2001) Test Effort Estimation using Use Case Points, Quality Week, San Francisco, California, USA.
[42]. H. Rubin (1995) Worldwide Benchmark Project Report, Rubin Systems Inc.