Journal of Software Engineering and Applications, 2013, 6, 47-52
http://dx.doi.org/10.4236/jsea.2013.610A006 Published Online October 2013 (http://www.scirp.org/journal/jsea)

A Survey of Software Test Estimation Techniques

Kamala Ramasubramani Jayakumar[1], Alain Abran[2]

[1]Amitysoft Technologies, Chennai, India; [2]École de Technologie Supérieure—University of Quebec, Montreal, Canada.
Email: jayakumar@amitysoft.com

Received August 30th, 2013; revised September 27th, 2013; accepted October 5th, 2013

ABSTRACT

Software testing has become a primary business for a number of IT services companies, and estimation, which remains a challenge in software development, is even more challenging in software testing. This paper presents an overview of software test estimation techniques surveyed, as well as some of the challenges that need to be overcome if the foundations of these software testing estimation techniques are to be improved.

Keywords: Software Testing; Testing Estimation; Software Estimation; Outsourcing

1. Introduction

Software testing has become a complex endeavor, owing to the multiple levels of testing that are required, such as component testing, integration testing, and system testing, as well as the types of testing that need to be carried out, such as functional testing, performance testing, and security testing [1]. It is critical that relevant estimation techniques be used in software testing, depending on the scope what the testing called for.

Furthermore, software testing has become an industry of its own over the years, with the emergence of independent testing services firms and IT services companies establishing testing services as a business unit. Test estimation consists of the estimation of effort and cost for a particular level of testing, using various methods, tools, and techniques. The incorrect estimation of testing effort often leads to an inadequate amount of testing, which, in turn, can lead to failures of software systems once they are deployed in organizations. Estimation is the most critical activity in software testing, and an unavoidable one, but it is often performed in haste, with those responsible for it merely hoping for the best.

Test estimation techniques have often been derived from generic software development estimation techniques, in which testing figures are as one of the phases of the software development life cycle, as in the COCOMO 81 and COCOMO II models [2].

Test estimation in an outsourcing context differs significantly from test estimation embedded in software development, owing to several process factors related to development organization and testing organization, in addition to factors related to the product to be tested. Distinct pricing approaches, such as time and material, fixed-bid, output-based, and outcome-based pricing, are followed by the industry based on customer needs. An adequate estimation technique is essential for all pricing models except for time and material pricing. The focus of the survey reported here is the estimation of effort for testing, and not the estimation of cost or schedule.

2. Evaluation Criteria and Groups of Estimation Techniques

For this study, the following criteria have been selected to analyze test estimation techniques:

1. **Customer view of requirements:** This criterion makes it possible to determine whether the estimation technique looks at the software requirements from a customer viewpoint or from the technical/implementation viewpoint: estimation based on the customer viewpoint assures the customer that he is getting a fair price (*i.e.* estimate) in a competitive market for what he is asking for in terms of quantity and quality, and that an increase or decrease in price (*i.e.* the estimated effort) is directly related to increases or decreases in the number of functions and/or levels of quality expected, and not on how efficient, or otherwise, a supplier is at delivering software using different sets of tools and with different groups of people.
2. **Functional size as a prerequisite to estimation:** Most estimation methods use some form of size, which is either implicit or explicit in effort estimation: when

size is not explicit, benchmarking and performance studies across projects and organizations are not possible. Functional size can be measured using either international standards or locally defined sizing methods.

3. **Mathematical validity:** Surprisingly, quite a few estimation techniques have evolved over the years, mostly based only on "feel good" factor and ignoring the validity of their mathematical foundations. This criterion looks at the metrological foundation of the proposed estimation techniques. A valid mathematical foundation provides a sound basis for further improvements.
4. **Verifiability:** The estimate produced must be verifiable, in order to inspire confidence. Verifiability makes the estimate more dependable.
5. **Benchmarking:** It is essential in an outsourcing context that estimates be comparable across organizations, as this can help in benchmarking and performance improvement. The genesis of the estimation techniques is looked at to determine whether or not benchmarking is feasible.

Estimation techniques are based on a number of different philosophies. For the purposes of this survey, these techniques have been classified in to the following groups:

1) techniques based on judgment and rules of thumb,
2) techniques based on analogy and work breakdown,
3) techniques based on factors and weights,
4) techniques based on size,
5) fuzzy and other models.

There are also, of course, several variations of techniques in each group. In this paper, we present a few representative techniques from each group to illustrate their basis for estimation, as well as their strengths and weaknesses.

3. Survey Findings

Using the five criteria described in the previous section, **Table 1** presents a high level analysis of each of the above groups of techniques for the estimation of software testing. Comments specific to each group of testing techniques are presented subsequently.

3.1. Judgment and Rule of Thumb Techniques

Delphi [3]: a classic estimation technique in which experts are involved in determining individual estimates for a particular set of requirements based on their own earlier experience. Multiple iterations take place during which the experts learn the reasoning from other experts, and rework their estimates in subsequent iterations. The final estimate is picked from the narrowed range of values estimated by experts in the last iteration.

Wide-band Delphi [4]: a technique enabling interaction between experts to arrive at a decision point. A quick estimate is provided by experts knowledgeable in the domain and in testing, but the resulting estimate will be very approximate, and should be applied with caution. Estimates are not verifiable in this case, and benchmarking is possible. These techniques mostly take the implementation view of requirements, and functional size is often ignored.

Rule of Thumb: estimates which are based on ratios and rules pre-established by individuals or by experienced estimators, but without a well documented and independently verifiable basis.

Typically, functional size is not considered in this group of testing estimation techniques. They are not based on the analysis of well documented historical data, and benchmarking is not feasible.

3.2. Analogy and Work Breakdown Techniques

Analogy-based [3]: techniques involving comparison of the components of the software under test with standard components, for which test effort is known based on historical data. The total estimate of all the components of the software to be tested is further adjusted based on project-specific factors and the management effort required, such as planning and review.

Table 1. Summary analysis of strengths and weaknesses of estimation techniques for testing.

Criteria → / estimation techniques ↓	Customer view of requirements	Functional size as a prerequisite	Mathematical validity	Verifiable	Bench marking
1) Judgment & rule of thumb	NO	NO	Not applicable	NO	NO
2) Analogy & work breakdown	NO	NO	YES	YES	Partially, and only when standards are used
3) Factor & weight	NO	NO	NO-units are most often ignored	YES	NO
4) Size	YES	YES	Varies with sizing technique selected	YES	YES
5) Fuzzy & other models	Partially	Most often, No	YES, in general, but at times units are ignored	Partially	Partially, and only when standards are used

Proxy-based (PROBE) [5]: a technique for estimating development effort, which can be adapted for estimating testing efforts for smaller projects. When the components of the product to be tested are significantly different from the standard components, a new baseline has to be established. Validation based on historical data is weak and fuzzy in this case. Functional size is not considered in such techniques, and benchmarking is not feasible.

Task-based [3]: a typical work breakdown-based estimation method where all testing tasks are listed and three-point estimates for each task are calculated with a combination of the Delphi Oracle and Three Point techniques [6]. One of the options offered by this method for arriving at an expected estimate for each task is a Beta distribution formula. The individual estimates are then cumulated to compute the total effort for all the tasks. Variations of these techniques, such as **Bottom-Up** and **Top-Down**, are based on how the tasks are identified. These techniques can work in a local context within an organization, where similar types of projects are executed. Benchmarking is not possible, since there is no agreed definition of what constitutes a task or work breakdown.

Test Case Enumeration-based [3]: an estimation method which starts with the identification of all the test cases to be executed. An estimate of the expected effort for testing each test case is calculated, using a Beta distribution formula, for instance. A major drawback of this technique is that significant effort has to be expended to prepare test cases before estimating for testing. This technique can work in a context where there is a clear understanding of what constitutes a test case. Productivity measurements of testing activities and benchmarking are not possible.

3.3. Factor and Weight-Based Estimation

Test Point Analysis [7]: a technique in which dynamic and static test points are calculated to arrive at a test point total. Dynamic test points are calculated based on function points, functionality-dependent factors, and quality characteristics. Function-dependent factors, such as user importance, usage intensity, interfacing requirements, complexity, and uniformity are given a rating based on predefined ranges of values. Dynamic quality characteristics, such as suitability, security, usability, and efficiency, are rated between 0 and 6 to calculate dynamic test points. Static points are assigned based on the applicability of each of the quality characteristic as per ISO 9126. Each applicable quality characteristic is assigned a value of 16 and summed to obtain the total number of static test points. The test point total is converted to effort based on ratings for a set of productivity and environmental factors.

While this technique appears to take into account several relevant factors, these factors do not have the same measurement units, which makes it difficult to use them in mathematical operations. Functionality- and quality-dependent factors are added together to obtain Dynamic Test Point, and subsequently to arrive at a Test Point size, which again is the sum of three different quantities with potentially different units of measurement. The basic mathematical principles of addition and multiplication are forgotten in the Test Point Analysis method. As such, it can be referred to as a complex "feel good" method.

Use Case Test Points: a technique suggested [7] as an alternative to Test Points and derived from Use Case-based estimation for software development. Unadjusted Use Case Test Points are calculated as the sum of the actors multiplied by each actor's weight from an actors' weight table and the total number of use cases multiplied by a weight factor, which depends on the number of transactions or scenarios for each use case. Weights assigned to each of the technical and environmental factors are used to convert unadjusted use case points to adjusted use case points. A conversion factor accounting for technology/process language is used to convert adjusted use case points into test effort. This is another example of a "feel good" estimation method: ordinal scale values are inappropriately transformed into interval scale values, which are then multiplied with the intention of arriving at a ratio scale value, which exposes the weak mathematical foundation. Although this technique takes a user's viewpoint of requirements, it is not amenable to benchmarking.

Test Execution Points [8]: a technique which estimates test execution effort based on system test size. Each step of the test specifications is analyzed based on characteristics exercised by the test step, such as screen navigation, file manipulation, and network usage. Each characteristic that impacts test size and test execution is rated on an ordinal scale—low, average, and high—and execution points are assigned. Authors have stopped short of suggesting approaches that can be adopted for converting test size to execution effort. The task of developing test specifications prior to estimation requires a great deal of effort in itself. Since test effort has to be estimated long before the product is built, it would not be possible to use this technique early in the life cycle.

Test team efficiency is factored into another variation of the estimation model for test execution effort. The **Cognitive Information Complexity Measurement Model** [9] uses the count of operators and identifiers in the source code coupled with McCabe's Cyclomatic Complexity measure. The measures used in this model lack the basic metrological foundations for quantifying complexity [10], and the validity of such measurements for estimating test execution effort has not been demonstrated.

3.4. Software Size-Based Estimation

The **Size-based**: an estimation approach in which size is used and a regression model is built based on historical data collected adopting standard definitions. Here, size is one of the key input parameters in estimation. Some of the techniques use the conversion factors to convert size into effort. Regression models built using size directly enable the estimation of effort based on historical data.

Test Size-based [3]: an estimation technique proposed for projects involving independent testing. The size of the functional requirements in Function Points using IFPUG's Function Point Analysis (FPA) [11] (ISO 20926) is converted to unadjusted test points through a conversion factor. Based on an assessment of the application, the programming language, and the scope of the testing, weights from a weight table are assigned to test points. Unadjusted test points are modified using a composite weighting factor to arrive at a test point size. Next, test effort in person hours is computed by multiplying Test Point Size by a productivity factor. FPA, the first generation of functional size measurement methods, suffers from severe mathematical flaws with respect to its treatment of Base Functional Components, their weights in relation to a complexity assessment, and the Value Adjustment Factor used to convert Unadjusted Function Points to Adjusted Function Points. The mathematical limitations of FPA have been discussed in [10]. In spite of these drawbacks, this technique has been used by the industry and adopted for estimating test effort. Performance benchmarking across organizations is possible with this technique for business application projects.

Other estimation models, such as **COCOMO** and **SLIM**, and their implementation in the form of estimation tools, along with other tools such as **SEER** and **Knowledge Plan** [12], employ size based estimation of software development effort from which testing effort is derived, often in proportion to the total estimated effort. Usually, the size of software, measured in SLOC, is an input parameter for these models. There are provisions to backfire function point size and convert it to SLOC to use as input, although this action actually increases uncertainty in the estimate. These models use a set of project data and several predetermined factors and weights to make up an estimation model. The intention behind using these models and tools is interesting, in that they capture a host of parameters that are expected to influence the estimate. Abran [10] has observed that many parameters of COCOMO models are described by linguistic values, and their influence is determined by expert opinion rather than on the basis of information from descriptive engineering repositories. It should be noted that some of these models are built based on a limited set of project data and others with a large dataset, but they primarily use a black box approach. Some of the tools use predetermined equations, rather than the data directly. These models and automated tools can provide a false sense of security, when the raw data behind them cannot be accessed for independent validation and to gain additional insights.

ISBSG equations [13]: techniques derived from estimation models based on a repository of large project datasets. These equations are based on a few significant parameters that influence effort estimates, as analyzed through hundreds of projects data from the open ISBSG database. Such estimation equations are built as a white box approach, with the ability to understand the underlying data and learn from them. Using a similar approach, practical estimation models using functional size can be developed by the organizations themselves. Another study of projects in the ISBSG database has come up with interesting estimation models for software testing using functional size [14].

Capers Jones [15] mentions that FPA and COSMIC Function Points [16] provide interesting insights into the quantum of test cases (also referred to as "test volume"), and proposes various rules of thumb based on function points in order to calculate the number of test cases required and to estimate the potential number of defects. Estimate of test cases and defects lead to estimate of efforts for overall testing. However, the authors caution against using rule of thumb methods, indicating that they are not accurate and should not be used for serious business purposes.

Non functional requirements, like functional requirements, are quite critical in software testing. An approach for **estimating the test volume and effort** is proposed in [17], where the initial estimate based on functional requirements is adjusted subsequently by taking into consideration non functional requirements. This model uses **COSMIC Function Points**, the first of the 2nd generation functional size measurement methods adopted by the ISO [18], as it overcomes the limitations of the 1st generation of Function Point sizing methods. Estimates for non functional testing are arrived at based on graphical assessment of non functional requirements of project data. Estimation models developed by performing regression analysis between COSMIC Function Points and development effort have been successfully used in the industry [19]. Estimation models built using historical project data with a COSMIC Function Point size have a strong mathematical foundation, are verifiable, and take into account the customer view of requirements. In addition, they are amenable to benchmarking and performance studies.

3.5. Fuzzy Inference & Other Models

Fuzzy models have been proposed to account for incomeplete and/or uncertain input information available for estimation. One of the fuzzy logic approaches [20] uses COCOMO [2] as a foundation for fuzzy inference using mode and size as inputs. Even though this approach is proposed for estimating software development effort, it could be adopted for estimating testing effort as well. Another approach has been proposed by Ranjan *et al.* [21]. This technique again uses COCOMO as the basis on which KLOC is used as an input, and development effort is calculated using effort adjustment factors based on cost drivers.

Francisco Valdès [22] designed a fuzzy logic estimation process in his Ph. D thesis. His approach is the purest form of the application of fuzzy logic to the software estimation context, since it does not use any of the other estimation techniques. The model allows experts to decide on the most significant input variables for the kinds of projects in which the model will be applied. The membership function is defined for the input variables, and the values are assigned based on expert opinion. This creates fuzzy values that are used in inference rule execution. Unlike other expert judgement-based methods, where the knowledge resides with experts, here the knowledge is captured in the form of inference rules and stays within the organization. The estimates produced by these models can also be verified. However, setting up a fuzzy rule base requires time and the availability of experts in the domain/technology for which estimation is being performed.

Software Development Estimation techniques using Artificial Neural Networks (ANN) [23,24] and Case-Based Reasoning (CBR) [25] are reported by researchers. There is an implementation of ANN for software testing [26], which inherits the drawbacks from Use Case based estimation. ANN and CBR have to be further investigated for arriving at practical and mathematically correct estimation of software testing.

4. Summary and Key Research Needs

Judgment & Rule of Thumb-based techniques are quick to produce very approximate estimates, but the estimates are not verifiable and of unknown ranges of uncertainty.

Analogy & Work Break-Down techniques may be effective when they are fine-tuned for technologies and processes adopted for testing. They take an implementation view of requirements, and cannot be used for benchmarking purposes.

Factor & Weight-based techniques perform several illegal mathematical operations, and lose scientific credibility in the process. They can serve as “feel good” techniques.

Functional Size-based techniques are more amenable to performance studies and benchmarking. They tend to produce more accurate estimates [13]. COSMIC Functional Size-based estimation overcomes the limitations of the first generation of the Function Point method.

Estimation Tools based on black-box datasets have to be used very judiciously, with an understanding of where the models come from and the kind of data used to build them.

Innovative approaches, such as **Fuzzy Inference, Artificial Neural Networks, and Case-based Reasoning**, are yet to be adopted in the industry for estimating testing effort.

The estimation techniques surveyed here are currently limited in their scope for application in outsourced software testing projects, and do not attempt to tackle estimates for all types of testing. The classification of test effort estimation techniques presented here is also applicable for software development effort estimation techniques with the same strengths and weakness.

Prior to pursuing further research to improve any one specific estimation technique, it would be of interest to develop a more refined view of the evaluation criteria identified in this paper, and to explore, in either software engineering or other disciplines, what candidate solutions or approaches could bring about the most benefit in terms of correcting weaknesses that have been identified or adding new strengths.

REFERENCES

[1] ISO, “ISO/IEC DIS 29119—1, 2, 3, & 4: Software and Systems Engineering—Software Testing—Part 1, 2, 3, & 4,” International Organization for Standardization (ISO), Geneva, 2012.

[2] B. W. Boehm, “Software Cost Estimation with COCOMO II,” Prentice Hall, Upper Saddle River, 2000.

[3] M. Chemuturi, “Test Effort Estimation,” 2012. http://chemuturi.com/Test%20Effort%20Estimation.pdf

[4] M. Nasir, “A Survey of Software Estimation Techniques and Project Planning Practices,” 7*th ACIS International Conference on Software Engineering* (*SNPD*’06), Las Vegas, 19-20 June 2006, pp. 305-310.

[5] W. Humphrey, “A Discipline for Software Engineering,” Addison Wesley, Boston, 1995.

[6] R. Black, “Test Estimation—Tools & Techniques for Realistic Predictions of Your Test Effort,” 2012. http://www.rbcs-us.com/software-testing-resources/library/basic-library

[7] M. Kerstner, “Software Test Effort Estimation Methods,” Graz University of Technology, Graz, 2012. www.kerstner.at/en/2011/02/software-test-effort-estimation-methods

[8] E. Aranha and P. Borba, “Sizing System Tests for Estimating Test Execution Effort,” Federal University of Pernambuco, Brazil & Motorola Industrial Ltd, *COCOMO Forum* 2007, Brazil, 2012.

[9] D. G. E. Silva, B. T. De Abreu and M. Jino, “A Simple Approach for Estimation of Execution Effort of Functional Test Cases,” *International Conference on Software*

Testing Verification and Validation, Denver, April 2009, pp. 289-298.

[10] A. Abran, "Software Metrics and Software Metrology," Wiley & IEEE Computer Society Press, Hoboken, 2010. http://dx.doi.org/10.1002/9780470606834

[11] IFPUG, "Function Point Counting Practices Manual, Version 4.2.1," International Function Points Users Group, New Jersey, 2005.

[12] S. Basha and P. Dhavachelvan, "Analysis of Empirical Software Effort Estimation Models," *International Journal of Computer Science and Information Security*, Vol. 7, No. 2, 2010, pp. 68-77.

[13] P. R. Hill, "Practical Project Estimation," 2nd Edition, International Software Benchmarking Standards Group (ISBSG), Victoria, 2005.

[14] K. R. Jayakumar and A. Abran, "Analysis of ISBSG Data for Understanding Software Testing Efforts," *IT Confidence Conference*, Rio, 3 October 2013.

[15] C. Jones, "Estimating Software Costs," 2nd Edition, Tata McGraw-Hill, New York, 2007.

[16] A. Abran, J.-M. Desharnais, S. Oligny, D. St-Pierre and C. Symons, "The COSMIC Functional Size Measurement Method Version 3.0.1, Measurement Manual, The COSMIC Implementation Guide for ISO/IEC 19761:2003," Common Software Measurements International Consortium (COSMIC), Montreal, 2009.

[17] A. Abran, J. Garbajosa and L. Cheikhi, "Estimating the Test Volume and Effort for Testing and Verification & Validation," *IWSM-Mensura Conference*, 5-9 November 2007, UIB-Universitat de les Illes Baleares, Spain, 2007, pp. 216-234. http://www.researchgate.net/publication/228354130_Estimating_the_Test_Volume_and_Effort_for_Testing_and_Verification__Validation

[18] ISO, "ISO/IEC 19761: Software Engineering—COSMIC—A Functional Size Measurement Method," International Organization for Standardization (ISO), Geneva, 2011.

[19] R. Dumke and A. Abran, "Case Studies of COSMIC Usage and Benefits in Industry, COSMIC Function Points—Theory and Advanced Practices," CRC Press, Boca Raton, 2011.

[20] P. V. G. D. Prasad Reddy, K. R. Sudha and P. Rama Sree, "Application of the Fuzzy Logic Approach to Software Estimation," *International Journal of Advanced Computer Science and Applications*, Vol. 2, No. 5, 2011, pp. 87-92.

[21] P. R. Srivastava, "Estimation of Software Testing Effort: An Intelligent Approach," 20*th International Symposium on Software Reliability Engineering* (*ISSRE*), Mysore, 2009. http://www.researchgate.net/publication/235799428_Estimation_of_Software_Testing_Effort_An_intelligent_Approach

[22] F. Valdès, "Design of the Fuzzy Logic Estimation Process," Ph.D. Thesis, University of Quebec, Quebec, 2011.

[23] A. S. Grewal, V. Gupta and R. Kumar, "Comparative Analysis of Neural Network Techniques for Estimation," *International Journal of Computer Applications*, Vol. 67, No. 11, 2013, pp. 31-34.

[24] J. Kaur, S. Sing, K. S. Kahlon and P. Bassi, "Neural Network—A Novel Technique for Software Effort Estimation," *International Journal of Computer Theory and Engineering*, Vol. 2, No. 1, 2010, pp. 17-19.

[25] Y. A. Hassan and T. Abu, "Predicting the Cost Estimation of Software Projects Using Case-Based Reasoning," *The* 6*th International Conference on Information Technology* (*ICIT* 2013), Cape Town, 8-10 May 2013.

[26] C. Abhishek, V. P. Kumar, H. Vitta and P. R. Srivastava, "Test Effort Estimation Using Neural Network," *Journal of Software Engineering & Applications*, Vol. 3, No. 4, 2010, pp. 331-340. http://dx.doi.org/10.4236/jsea.2010.34038