

# Sizing System Tests for Estimating Test Execution Effort

Eduardo Aranha<sup>1,2</sup> and Paulo Borba<sup>1</sup>

<sup>1</sup>*Informatics Center – Federal University of Pernambuco  
PO Box 7851, Recife, PE, Brazil  
{ehsa,phmb}@cin.ufpe.br*

<sup>2</sup>*Mobile Devices R&D – Motorola Industrial Ltda  
Rod SP 340 - Km 128,7 A - 13820 000 Jaguariuna/SP - Brazil*

## Abstract

*In our researches, we developed a method to size tests based on their specifications. This measure, called execution points, can be used as input for test execution effort estimation models. Here, we present our method for sizing tests that is based on test specifications written in natural language.*

*We also presents the main functionalities of a tool developed for supporting our measurement method. In addition, we discuss how some techniques can be used for estimating test execution effort based on the proposed test size measure. Some interesting results of an empirical study run on the mobile application domain are also discussed. For instance, we verified in this empirical study a high linear correlation between test execution effort and execution points.*

## 1. Introduction

In the development of complex systems, planning is essential for achieving results within the schedule and budget. Several cost models were developed in order to support this planning, such as the well known COCOMO II family [1][2]. These models usually rely on software life cycles models and software size measures, achieving significant estimation accuracy when regarding software development effort. When observing the distribution of this effort, one of the most costly activities is testing.

Aiming at cost reduction and performance improvement, some organizations may have teams exclusively allocated for executing system tests for several development teams. Hence, test managers should plan their own test schedules and resources. However, test managers have difficulties using existing cost models, since the effort to execute tests are more related to the characteristics of the tests rather than characteristics of the software.

In our researches, we developed a method to size tests based on their specifications. This measure, called execution points, can be used as input for test

execution effort estimation models. For instance, we can define an estimation model that its input is the test specifications of a test suite and its output is the estimated effort required to execute all tests in the suite.

The proposed measure represents the test size and execution complexity of test cases. Test size means the amount of steps required to execute the test. In addition, test execution complexity is related to the relationship (complexity of interaction) between the tester and the tested product required during the test. These definitions are adaptations of the idea of size and development complexity for software products.

Considering test specifications written in a standardized way, we developed a method and a tool for supporting test size and execution complexity measurement based on a semi-automatic analysis of these test specifications. The measurement method is based on the evaluation of test actions according to a set of characteristics which weights are defined by expert judgment or historical data (Delphi assessment or Analysis of Variance).

After sizing the tests, we can use the measure in different estimation models. For instance, we can define a model in a similar way of the COCOMO, in which risk factors (related to testing) are used for adjusting the test effort according to some characteristics of personnel, test project, test environment and others.

The remaining of this paper is structured as follow. Section 2 presents our method for sizing tests that is based on test specifications written in natural language. Then, Section 3 lists some techniques to estimate test execution effort using our test size measure. After that, we show in Section 4 the main functionalities of a tool developed for supporting our measurement method. Also, we discuss in Section 5 some results of an empirical study run on the mobile application domain. Finally, we present our conclusions on Section 6.

## **2. Sizing system tests**

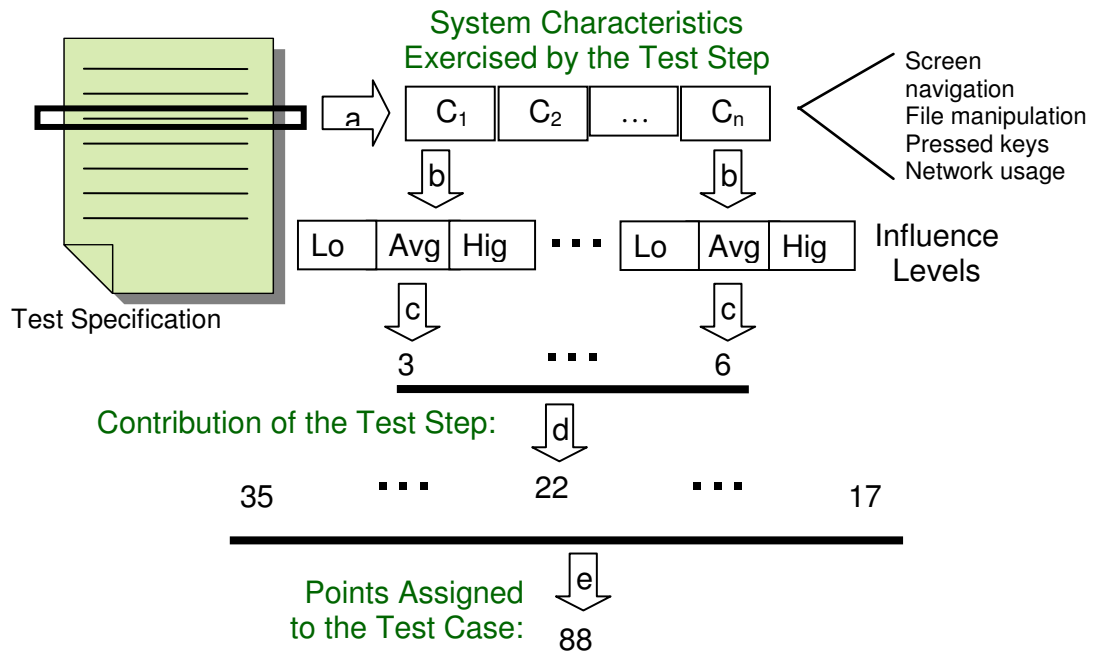
Here, we propose a method to measure the size and execution complexity of test cases in a test suite. Test size means the amount of steps required to execute the test. Test execution complexity is related to the relationship (complexity of interaction) between the tester and the tested product required during the test. These definitions are adaptations of the idea of size and development complexity for software products [3][5][9]. The proposed measure is given in execution points, a generic unit-of-work measure defined by this work.

In practice, the execution points count of a test case gives us a quantitative reference about its size and execution complexity. For instance, a test case rated with 700 execution points is bigger than others rated with 590 and 350. In addition, it allows us to better compare test productivity or capacity. For example, a tester that executed 5 tests rated with 500 execution points each one is faster than another that executed 15 tests rated with 100 execution points during the same amount of time.

### **2.1. The measurement method**

This section presents how we measure the size and execution complexity of a test case. All required information is extracted from the test specification. Although not essential, we consider in this paper that test specifications are written in natural language, as discussed in Section 2.2.

Figure 1 illustrates how our measurement method works. First, (a) we individually analyze each test step of the test specification. This step by step analysis was defined with the objective to support the method automation. We analyze each test step according to a list of characteristics ( $C_1$  to  $C_n$ ).



**Figure 1: Assigning execution points to test cases.**

These characteristics represent some general functional and non-functional requirements exercised when the test step is executed. Examples of possible characteristics are number of navigations between screens, number of pressed keys and use of network. The list of characteristics may not be the same for different application domains.

Each characteristic considered by the model has an impact in the size and execution complexity of the test and (b) this impact is rated using an ordinal scale (Low, Average and High). We have to create guidelines to help us to objectively choose the more appropriate impact level for each characteristic.

After that, (c) we assign execution points for each characteristic according to its impact level. The objective here is to transform the qualitative rate (impact level) into a quantitative value.

For instance, a characteristic  $C_i$  rated with the Low value could be assigned to 30 execution points. However, a more relevant characteristic rated with the same Low value

may be assigned to a higher number of execution points. Here, guidelines need to be provided for assigning the correct value for each possible characteristic value.

To calculate the total number of execution points of a test step, (d) we sum the points assigned for each characteristic. Then, (e) we measure the size and execution complexity of a test case by summing the execution points of each one of its test steps.

## 2.2. Test specification language

Tests are usually specified in terms of precondition, procedure (list of test steps with inputs and expected outputs) and post-condition [7]. These specifications are commonly written in natural language, often leading to problems such as ambiguity, redundancy and lack of writing standard. All these problems make difficult test understanding and execution complexity estimation. Nevertheless, they can be avoided using controlled natural languages.

A controlled natural language (CNL) [10] is a subset of natural language with restricted grammar and lexicon in order to have sentences written in a more concise and standard way. This restriction reduces the number of possible ways to describe an event, action or object.

The test specifications considered by this work are written in either NL (natural language) or the CNL described here. In a simplified way, each sentence (test step) in the specification conforms to the following structure: a main verb and zero or more arguments. Table 1 shows an example of test procedure written in a controlled natural language defined for the mobile application domain.

**Table 1: Example of a test procedure written in a controlled natural language.**

Step	Description	Expected Results
1	Start the message center.	The phone is in message center.
2	Select the new message option.	The phone is in message composer.
3	Insert a recipient address into the recipients field.	The recipients field is filled.
4	Insert a SMS content into the message body.	The message body is populated.
5	Send the message.	The send message transient is displayed. The message is sent.

The verb identifies the action of the test step to be performed during the test. The arguments provide additional information about the action represented by the verb. For instance, the sentence *Start the message center* has the verb *start* (action of starting an application) and the required argument *the message center* (application to be started).

The CNL can have its lexicon and grammar extended for specific application domains. For example, the list of possible verbs and arguments may be different between

the mobile and the Web application domains. The CNL simplifies the use of our model and also efficiently supports a high level of automation of our measurement method.

### **3. Using test size for estimating test execution effort**

Several techniques can be used to estimate the effort to execute the test cases based on our measure of test size and execution complexity. We are investigating and comparing some of these techniques:

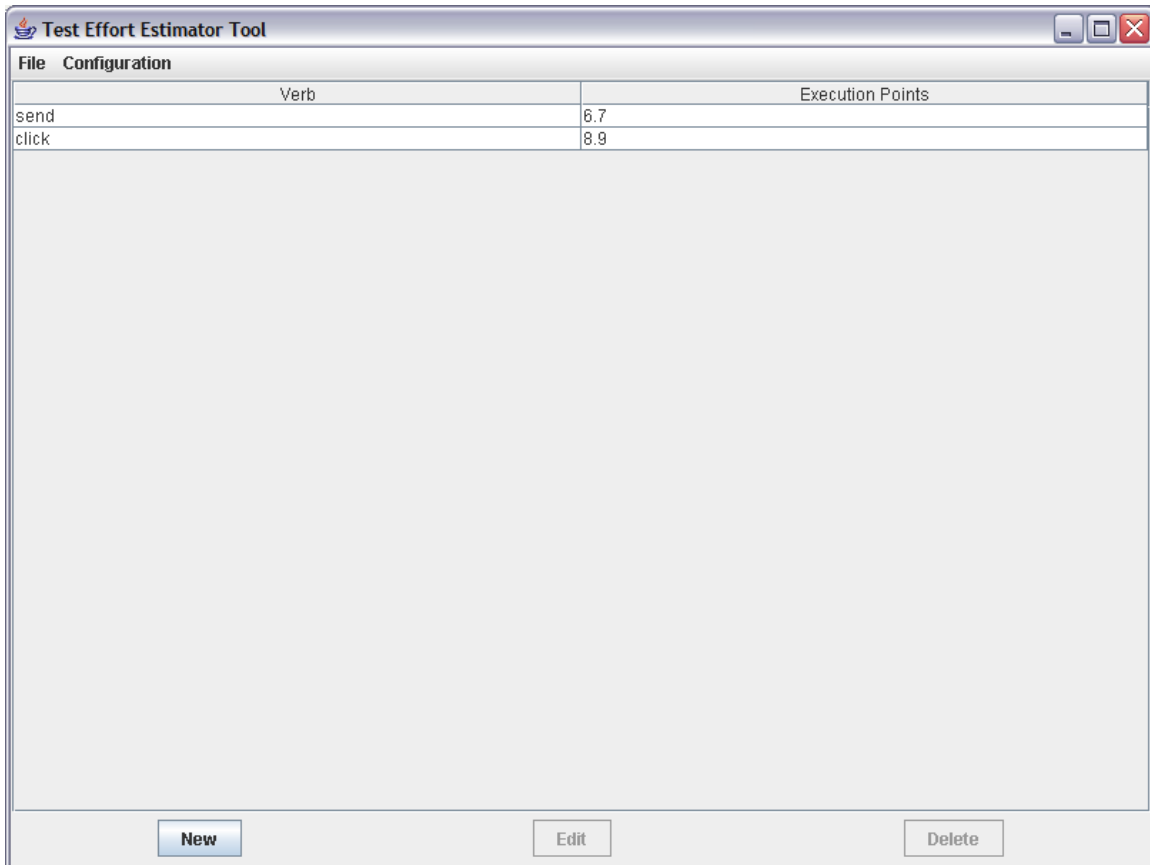
- COCOMO-based model: we are investigating the creation of a model similar to the COCOMO model [1], where cost drivers and scale factors related to test execution effort are used.
- Stepwise Regression Analysis (SWR): the stepwise regression analysis [6] can find an equation relating the size of the tests and other variables considered relevant by this technique.
- Case-Based Reasoning (CBR): using this technique [11], we can estimate effort based on similar past projects. The size of the tests is one of the most important attribute to be used when comparing test projects.
- Classification And Regression Trees (CART): this is a tree-building technique [4], in which some attributes of the project (test size, cost drivers, etc.) are selected to create the nodes (if-clauses) in order to split the data. Then, each leaf of the tree groups test projects with similar effort.

### **4. Tool support**

We developed the Test Effort Estimator Tool for supporting the activities of sizing tests and estimating execution effort. Figure 2 presents a screen of the tool showing the points assigned to the verbs *send* and *click*. This tool has the following characteristics:

- Natural language processing: reads test cases written in natural language. All test steps are evaluated and the verbs are identified, evaluated and registered in a database.
- Management of exercised system characteristics: the user can set the system characteristics to be used during the test size and execution complexity measurement, as well all the guidelines and characteristic weights required by our method.
- Measurement of the size of test cases: the test cases read into the tool are automatically processed and the execution points of each test case are calculated.

- Estimations are performed based on the measured number of execution points and the reported risk factors.



The screenshot shows a window titled "Test Effort Estimator Tool" with a menu bar containing "File" and "Configuration". Below the menu bar is a table with two columns: "Verb" and "Execution Points". The table contains two rows of data: "send" with 6.7 execution points and "click" with 8.9 execution points. At the bottom of the window, there are three buttons: "New", "Edit", and "Delete".

Verb	Execution Points
send	6.7
click	8.9

**Figure 2: Execution points assigned to verbs.**

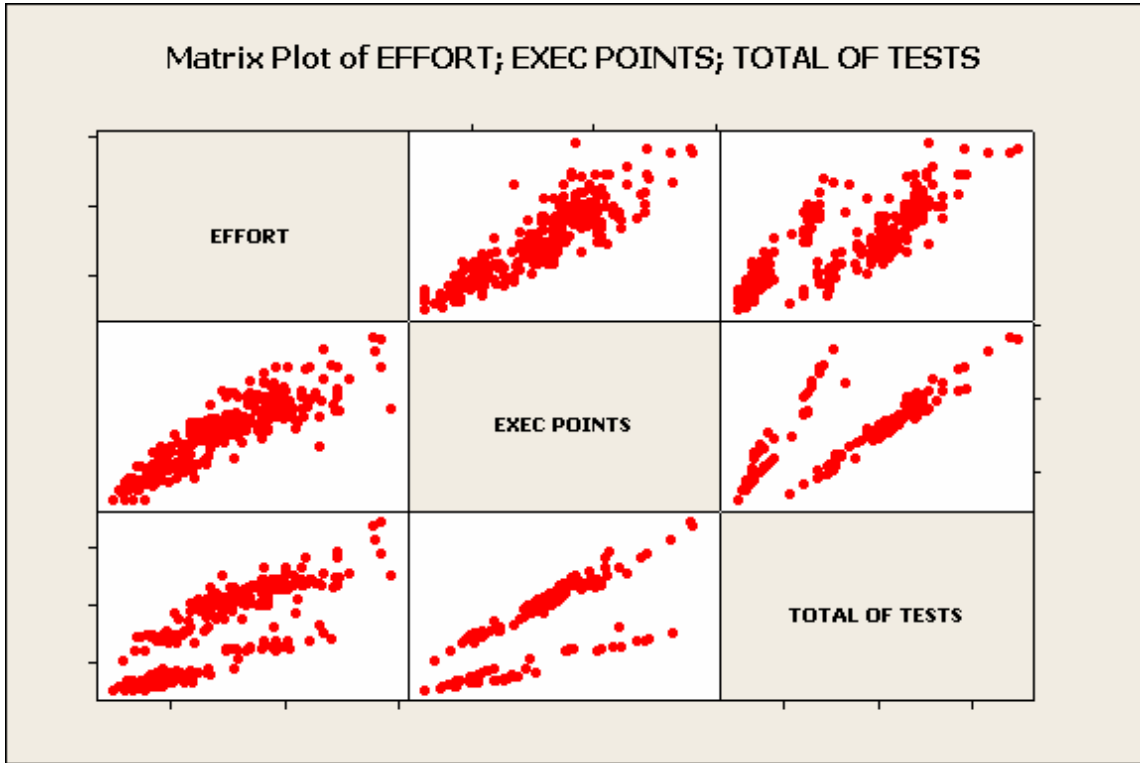
## **5. Empirical study on the mobile application domain**

This section overviews an empirical study we run using our test size measure on the mobile application domain. First, we configured our measurement method for the target domain. To define the list of characteristics to use, we invited 6 experienced testers. They identified the relevant characteristics and defined the guidelines in a Delphi panel that took only four hours (two sessions of two hours).

In addition, we have the cost to evaluate the size and execution complexity of test steps. Although it will usually take less than a minute to evaluate a test step, there may exist hundreds of test steps to be evaluated. In our case, we had more than 2 hundred of verbs to evaluate, taking almost 5 hours of work. The list of verbs was automatically extracted from the test specifications.

After that, we used the developed tool for sizing the test specifications. Then, we analyzed the correlation between the execution points of the test cases and the effort to execute them. Since the number of test cases is commonly used to estimate test execution

effort, we also analyzed its correlation. As shown in Figure 3, the linear correlation between effort and execution points was very significant, achieving a Pearson's correlation of 0.89. This number was larger than the correlation between effort and number of tests (0.79).



**Figure 3: Correlation between test execution effort, execution points and number of tests.**

When using execution points instead of number of tests to calculate test productivity and to estimate test execution effort, we reduced the mean magnitude of the relative error (MMRE) in approximately 30% and increased the prediction at level 20% (PRED(.20)) in approximately 61%, with statistical significance verified using t-tests.

## 6. Conclusions

Existing estimation models in the literature are based on system specifications and they estimate the effort required to perform more activities than test execution, such as defining and implementing tests. Then, they cannot be used to estimate the execution effort of a given test case.

In this paper, we presented a method for sizing tests based on test specifications. Our method sizes tests by analyzing the test steps according to a list of system characteristics exercised during the test execution. Although it is not required, the use of a controlled natural language reduces the ambiguity, helping the test size and execution complexity measurement. Also, the method for measuring test execution complexity was automated by a supporting tool.

For the mobile application domain, we defined the relevant system characteristics exercised by the test cases and their weights. This definition used intuition and expert judgment through a Delphi panel. We also analyzed the effort to analyze the verbs according to the characteristics defined during the Delphi panel. We believe that this effort is compensated by the accuracy achieved when using our proposed test size measure. This accuracy was a result of the high correlation between execution points and test execution effort.

## Acknowledgments

The first author is partially supported by Motorola, grant BCT-0021-1.03/05, through the Motorola Brazil Test Center Research Project. The second author is partially supported by CNPq, grant 306196/2004-2. We would like to thank Filipe Almeida and Vitor Fontes for the development of the tool used in our study.

## References

- [1] B. Boehm, E. Horowitz, R. Madachy, D. Reifer, B. Clark, B. Steece, W. Brown, S. Chulani, and C. Abts. *Software Cost Estimation with COCOMO II*. Prentice Hall, 2000.
- [2] B. Boehm, D. Reifer, R. Valerdi. COSYSMO: A Systems Engineering Cost Model. *1st Conference on Systems Integration*, March 2003.
- [3] L. Briand, K. E. Emam, and S. Morasca. On the application of measurement theory in software engineering. *Empirical Software Engineering: An International Journal*, 1(1):61–88, 1996.
- [4] L. Brieman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth Inc., Belmont, 1984.
- [5] N. Fenton. Software measurement: A necessary scientific basis. *IEEE Transactions on Software Engineering*, 20(3):199–206, 1994.
- [6] R. Hocking. *Methods and Applications of Linear Models: Regression and the Analysis of Variance*. Wiley-Interscience, 2003.
- [7] P. Jorgensen. *Software Testing, A Craftsmans Approach*. CRC Press, second edition, 2002.
- [8] H. Linstone and M. Turoff. *The Delphi Method: Techniques and Applications*. <http://is.njit.edu/pubs/delphibook>, 2002.
- [9] C. Pandian. *Software Metrics: A Guide to Planning, Analysis, and Application*. CRC Press, Inc., 2003.
- [10] R. Schwitter. English as a formal specification language. *In Proceedings of the 13th International Workshop on Database and Expert Systems Applications (DEXA02)*, pages 228–232, 2002.
- [11] I. Watson. *Applying Case-Based Reasoning: Techniques for Enterprise Systems*. Morgan Kaufmann, 1997.