# Measuring Test Execution Complexity

Eduardo Aranha[1,2]
*ehsa@cin.ufpe.br*

Paulo Borba[1]
*phmb@cin.ufpe.br*

[1]*Informatics Center*
*Federal University of Pernambuco*
*PO Box 7851, Recife, PE, Brazil*
*+55 81 2126-8430*

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

## Abstract

*Testing is an important activity to ensure software quality. Tests are usually run several times to certify that code maintenance did not accidentally insert defects into working parts of the software. In such situations, test teams shall be able to estimate the required effort to execute test cases in its schedules. This effort is directly related to the test execution complexity.*

*This paper presents a method for measuring the execution complexity of test cases based on its specifications written in a controlled natural language.*

## 1. Introduction

In competitive markets (e.g., the mobile phone market), companies that release products with poor quality may quickly lose its clients. In order to avoid this, companies should ensure that product quality has conformed to its client expectation. A usual activity performed to ensure quality is software testing.

There are many different testing strategies. Regarding functional testing, for example, it is a usual test strategy used to ensure product quality that should not be a bottleneck in the process, since its execution is usually done at the end of the development cycles. Otherwise, the risk to skip this activity due to short delivery schedules may increase significantly.

Software testing is been considered so important that organizations can assign teams only for testing activities. Testing is an important activity to ensure software quality. Tests are usually run several times to certify that code maintenance did not accidentally insert defects into working parts of the software. In such situations, test teams shall be able to estimate the required effort to execute test cases in its schedules and to request more resources or negotiate deadlines when necessary.

When regarding model-based testing approaches, a high number of test cases can be automatically generated. As team resources are limited, it may be not practical to execute all generated test cases. Their complexity usually determines the effort required to execute them and it can be used for planning test resources and test suites.

Several software development estimation models have been proposed over the years. However, these models do not estimate the effort for executing a given test suite, since their estimations are based on software development complexity instead of its execution complexity.

This paper presents an overview of how to measure the execution complexity of functional test cases based on its specifications written in a controlled language. It is organized as follows. In Section 2, we overview the controlled language used for test case specification. Next, Section 3 presents a new method for measuring test execution complexity. After that, we discuss related work in Section 4. Finally, our conclusions are presented in Section 5.

## 2. Test specification language

Test case specifications usually describe the tests in terms of test precondition, procedure (list of test steps with its inputs and expected outputs) and post-condition [3]. These specifications are commonly written in natural language, often leading to problems such as ambiguity, redundancy and lack of writing standard, which difficult not only test execution, but also its complexity estimation. However, the problems can be reduced using controlled natural languages [8].

A controlled natural language (CNL) is a subset of natural language with restricted grammar and lexicon,

in order to have sentences written in a more concise and standard way.

The test specifications considered by this work are written using a controlled lexicon and grammar described here in a simplified way. Each sentence (test step) in the specification conforms to the following structure: a verb and its arguments.

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 phrase "*Launch the message application*" has the verb "*launch*", indicating the action of launching an application, and the required argument "*the message application*", indicating the application to be launched.

This controlled language can have its lexicon and grammar extended for specific application domains. For example, the list of verbs (actions) and arguments may be different between the mobile and the Web application domains.

As the context of this work is related to testing mobile applications for Motorola Brazil Test Center site at the Informatics Center/UFPE, the considered controlled natural language reflects this domain [4].

## 3. Measuring test execution complexity

Regarding the test cases written in the controlled language cited in Section 2, we can measure their execution complexity based on their specification. In order to measure the execution complexity of a test case, we perform the following activities for each test step defined in its specification:

- Evaluate the complexity level for executing the test step based on the functional and non-functional system characteristics exercised by it.
- Rate the execution complexity level in execution points, a quantitative measure of execution complexity.

After that, the execution complexity of a test case is calculated by summing the execution points of each of its test steps. This process is illustrated by Figure 1 detailed next.

### 3.1. Evaluate Execution Complexity Levels

For each test case, we evaluate its execution complexity analyzing each test step defined in the test specification. This analysis regards the functional and non-functional characteristics of the tested system that are being exercised by the test steps.

Table 1 shows some of the functional characteristics that are analyzed. There we have the number of navigations between screens or group of information, the total number of characters entered and the number of items to verify during the test step execution.

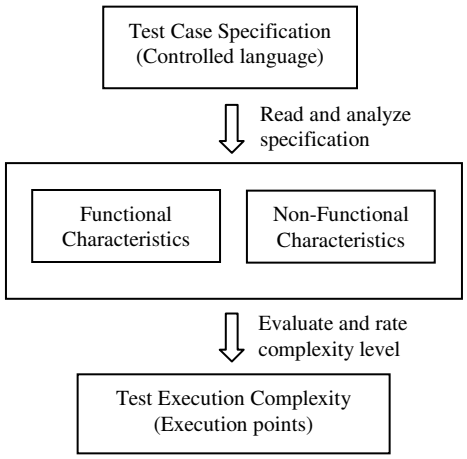


**Figure 1 – Measuring test execution complexity based on test specifications.**

For each characteristic, we have a score (low, average and high) indicating the complexity level for executing the test step when regarding only that characteristic.

In addition, Table 1 shows rules for defining the complexity level for each characteristic. For instance, a test step requiring less than five navigations is considered a low-complexity step for the "*# of navigations*" characteristic. These rules were defined using expert judgment.

**Table 1. Rules for defining complexity levels based on functional characteristics.**

| Functional Characteristics | Low | Average | High |
|---|---|---|---|
| # of navigations | < 5 | 5 - 15 | > 15 |
| Sum of the size of each input | < 20 | 20 - 40 | > 40 |
| # of items to verify | < 3 | 3 - 8 | > 8 |

The same process is done using the non-functional characteristics (see Table 2). In this case, we have characteristics such as system performance and network use. For example, the test step "*Take a picture*" has an average complexity associated for the "*system performance*" characteristic, since it has a slower performance than others test steps have (i.e., scrolling menus, etc.).

At the end of this activity, all test steps have complexity levels assigned to each functional and non-functional characteristic.

**Table 2. Rules for defining complexity levels based on non-functional characteristics.**

| Non-Functional Characteristics | Low | Average | High |
|---|---|---|---|
| System Performance | No influence | Taking a picture, Saving images, ... | Playing a long music, waiting for alarms, … |
| Use of Network | No influence | Sending/ receiving small messages, dialing a number, ... | Sending/ receiving multimedia messages, ... |

All the relevant characteristics and its impact in the complexity for executing a test step were defined here using intuition and expert judgment.

### 3.2. Rate the Execution Complexity Levels

Once we have evaluated the complexity level for each characteristic of each test step, we rate them into quantitative values. These values are called execution points, a generic unit for measuring test execution complexity. As we show later, the number of execution points helps us to compare the complexity of different test cases.

For rating the complexity levels, we assign weights for each functional and non-functional characteristic exercised by the test steps. These weights quantify the impact of each characteristic in the test execution complexity.

In Table 3, we present the weights for each complexity level of each functional characteristic. For instance, if the number of navigations performed by test step has a low-complexity level, this characteristic will contributes with 10 execution points for the total execution complexity.

**Table 3. Weights assigned to the complexity levels of each functional characteristic.**

| Functional Characteristics | Low | Average | High |
|---|---|---|---|
| # of navigations | 5 | 10 | 15 |
| Sum of the size of each input | 35 | 80 | 120 |
| # of items to verify | 10 | 20 | 30 |

The weights assigned to the non-functional characteristics are presented in Table 4. In this case, for example, if we have a test step using the network with an average-complexity level, this characteristic will contributes with 80 execution points for the total execution complexity.

**Table 4. Weights assigned to the complexity levels of each non-functional characteristic.**

| Non-Functional Characteristics | Low | Average | High |
|---|---|---|---|
| System Performance | 0 | 50 | 150 |
| Use of Network | 0 | 80 | 200 |

In this way, the execution complexity of a test step is determined by summing the execution points contributed by the rate of each complexity level, regarding the functional and non-functional characteristics. Table 5 shows an example of how to calculate this sum.

**Table 5. Summary of execution complexity evaluation and rating of a test step.**

| Functional Characteristics | Value | Complexity Level | Rating |
|---|---|---|---|
| # of navigations | 3 | Low | 5 |
| Sum of the size of each input | 60 | High | 120 |
| # of items to verify | 2 | Low | 10 |
| System Performance | No influence | Low | 0 |
| Use of Network | Sending a small message | Average | 80 |
| Total | | | 215 |

### 3.3. Calculate the Test Case Execution Complexity

After we have calculated the number of execution points of each test step of a test case, we calculate the total execution complexity of this test case by summing the execution points of each of its test steps. Table 6 presents this calculus for a sample test case. The execution complexity of the presented test case is 160 execution points.

The number of execution points of a test case gives us a reference about a test case execution complexity. More complex test cases should have higher numbers of execution points. Actually, small changes in a test specification may not change its number of execution points due to ranges used in rules for defining complexity levels. This is not a problem when

considering that the same test case executed several times would have similar but different runtimes due to environment influences.

**Table 6. Execution complexity of a test case.**

| Test Step | Rating |
|---|---|
| Launch message application. | 15 |
| Scroll to inbox folder | 10 |
| Open a unread message | 15 |
| … | ... |
| Delete the message | 30 |
| Total | 160 |

In addition, the number of execution points allows us to compare the execution complexity of two or more test cases. For instance, a test case rated with 700 execution points is more complex to execute than others two rated with 590 and 350. We also say that the first test case is approximately 20% more complex than the second and 50% more complex than the third.

We also can use the execution complexity measurement of a test case in order to estimate the required effort to execute it. For instance, testers can execute several test cases and observe the proportion between execution points and time spent for each test case. This information can then be used to calculate its test execution productivity. For example, the tester may verify a productivity of 120 execution points per minute. Using this productivity, a new test case with 160 execution points can be estimated to be executed in approximately 1 minute and 20 seconds.

Nevertheless, the test execution effort estimation is an even more complex activity, where environment conditions, team experience, the use of tools, the reuse of test setups and other factors must be considered. The test execution effort estimation is subject of further research.

## 4. Related Work

This work presents a method to measure test execution complexity. This metric is useful for estimating the effort to execute test suites. During the last few decades, several models and techniques were created for estimating software development size and effort.

Function Points Analysis (FPA) [2], for instance, gives a measure of the size of a system by measuring the complexity of system functionalities offered to the user. The size of system is determined in function points (FP), a unit-of-work measure, and this count is used for estimating the effort to develop it.

The Use Case Point Analysis (UCP) [6] is an extension of FPA and estimates the size of a system based on use case specifications. Both UCP and FPA regard the development complexity of a system, while the presented method regards the execution complexity of test cases.

The Constructive Cost Model (COCOMO) [1] converts size metrics such as FP and SLOC (source lines of code) into effort estimation for developing systems. This is done through the use of effort multipliers and scale factors, which represent the environment, the teams and the processes characteristics. We believe that a similar model can be defined to convert our execution complexity metric into the estimated effort to execute test cases.

Test Point Analysis [7] is another method that estimates the effort required to define and implement functional tests based on use case points. This model also takes in consideration the characteristics of the product being tested, but it estimates the effort of all test activities together, such as defining, implementing and executing tests.

## 5. Conclusions

This paper presents a method for measuring test execution complexity based on the use of test case specifications written in a controlled natural language. This information is especially useful when planning the test resources and test suites, where one important criterion is the effort to execute tests.

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

Our method does not require historical execution times of the test cases. This characteristic is extremely important in several situations, such as when test cases are new and different from any previous one. It is also important when you do not have reliable historical data or when you generate high numbers of test cases using model-based testing approaches.

The use of a controlled natural language reduces the ambiguity helping the complexity measurement. Actually, the number of possible ways to describe the same test step in a controlled language is minimal, and we also observed that a small but concise controlled language can support a high number of different test cases.

Based on these considerations, the method for measuring test execution complexity can be optimized as follows. The complexity evaluation of test steps are recorded and reused whenever possible in the complexity evaluation of other test cases. Over the time, the number of necessary evaluations tends to decrease.

The evaluations of similar test steps are also reused, since their execution complexity may be determined only by its verb. For instance, the act of launching an application has the same complexity for most applications. In this example, only the exceptions need to be evaluated.

With these optimizations and the possibility to automate all steps of the method, except the test step complexity evaluation, the costs for applying this method are significantly reduced.

The definition of the relevant functional and non-functional system characteristics exercised by the test cases was done using intuition and expert judgment. We observed the test cases specifications and its executions, and then we identified the relevant characteristics.

We also run an initial experiment aiming to validate the main concepts introduced by this method. Although we achieved interesting results suggesting the feasibility of the method, we have to run more expressive experiments in order to get more conclusive results [5] about the relevant characteristics, complexity levels and weights considered by the method.

We also intend to extend this work creating a test execution effort estimation in a similar way as done for COCOMO. This new test execution estimation model will regard test environment, processes and team experiences in order achieve better precision.

## Acknowledgments

We thank the anonymous reviewers, whose suitable comments helped improving the paper. 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.

## References

[1] Boehm, B., et. all. *Software Cost Estimation with COCOMO II*. Prentice Hall, 2000.

[2] Garmus, D., Herron, D. *Function Point Analysis, Measurement Practices for Successful Software Projects*. Addison Wesley, 2001.

[3] Jorgensen, P. *Software Testing, A Craftsman's Approach.* CRC Press, second edition, 2002.

[4] Leitão, D., Torres, D. and Barros, F. Nlforspec: Translating natural language descriptions into formal test case specifications. M.Sc. Thesis, 2006. In progress.

[5] Menzies T., et all. Prediction & verification: Validation methods for calibrating software effort models. In *Proceedings of the 27th international conference on Software engineering* (ICSE'05), pages 587-595. ACM Press, 2005.

[6] Mohagheghi P., Anda B., Conradi R. Effort estimation of use cases for incremental large-scale software development. In *Proceedings of the 27th international conference on Software engineering* (ICSE'05), pages 303–311. ACM Press, 2005.

[7] Nageswaren, S. Test Effort Estimation Using Use Case Points. In *14th International Internet & Software Quality Week 2001*. June, 2001.

[8] Schwitter, R. English as a formal specification language. In *Proceedings of the 13th International Workshop on Database and Expert Systems Applications (DEXA'02),* pages 228–232. IEEE Press, 2002.