Automated Test Execution Effort Estimation Based on Functional Test Specifications

Eduardo Aranha, Filipe de Almeida, Thiago Diniz, Vitor Fontes, Paulo Borba

Informatics Center – Federal University of Pernambuco (UFPE)
Recife – PE – Brazil

{ehsa,faa,tvsd,vmf,phmb}@cin.ufpe.br

Abstract. *A usual activity performed to ensure quality is software testing. Although important, testing is an expensive activity. For that reason, test teams should be able to estimate the required effort to perform their test activities on the schedule and to request more resources or negotiate deadlines when necessary. This is even more important when regarding regression testing, where the execution of hundred of tests may be required to verify if a software application still working after some change in the code.*

In order to estimate the effort to execute the tests of a given test suite, we proposed an automated effort estimation model that regards test size and execution complexity measured from functional test specifications written in natural language. This paper describes a tool that supports the measurement of test size and execution complexity, as well as the estimation of test execution effort. The tool was designed to be extended according to the customers needs, such as different estimation models, file formats of test specifications, parsers APIs, etc.

1. Introduction

In competitive markets (e.g., the mobile phone market), companies that release products with poor quality may quickly lose their 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 is a well-known rule of thumb saying that approximately 50 percent of the elapsed time and more than 50 percent of the total cost of typical software development project is spent in testing [1].

In addition, software testing is being considered so important that organizations can allocate teams exclusively for testing activities in order to achieve more test efficiency and unbiased test results. In such situations, test teams should be able to estimate the required effort to perform their test activities on the schedule and to request more resources or negotiate deadlines when necessary. In the practice, the consequences of poor estimates are costly to the organization: scope reduction, schedule overrun and overtime.

Several software development estimation models have been proposed over the years. COCOMO II [2] is a well-known example of model used for estimating the effort to develop software products based on their characteristics (lines of code, function

points, etc.). Regarding tests, Test Point Analysis [3] is a model used for estimating the effort to define, develop and execute functional tests.

However, these models do not estimate the effort to perform some specific testing activities (such as the execution of a given set of test cases), since their estimations are based on development complexity instead of test execution complexity. For this reason, estimates are usually made only based on expert judgment and historical data, which generally leads to a lack of precision.

In order to estimate the execution effort of a given functional test case or test suite (set of related test cases), we proposed a test execution effort estimation model [4][5] that regards test size and execution complexity measured from the functional test specifications written in natural language. This estimation model is especially useful when we periodically select existing tests to execute (regression testing, etc.), according to some testing objectives and available resources.

This paper describes a tool that we developed to support an automated estimation of functional test execution effort based on the measurement of size and execution complexity of tests or test suites. In Section 2, we overview the functionalities provided by the tool. After that, we present the architecture of the tool in Section 3. Finally, Section 4 shows our conclusions. In addition to the content of this paper, more information about tool license, examples of use and download of files are available at http://www.cin.ufpe.br/spg/TestExecEffortEstimation.

2. Functionalities

This section presents the main functionalities provided by our tool, which uses the model described in [4] and [5] to estimate the execution effort of a given test suite based on the test specifications.

Natural Language Processing. To automatically estimate the effort to execute test suites, the tool must be able to process test specifications, which are usually written in natural language. Figure 1 presents an example of test specification written in natural language.

	A	B	C
1	**Step**	**Description**	**Expected results**
2	1	Start the message center.	The phone is in message center.
3	2	Select the new message option.	The phone is in message composer.
4	3	Insert a recipient address into the recipients field.	The recipients field is filled.
5	4	Insert a SMS content into the message body.	The message body is populated.
6	5	Send the message.	The send message transient is displayed. The message is sent.
7			

Figure 1. Sample spreadsheet with test specification written in natural language.

Our tool has a natural language processing mechanism that identifies the test action represented by each test step (i.e., sentence in the test specification) based on the main verb of the sentence. Figure 1 shows some examples of test actions on the mobile application domain, such as the selection of a menu option (cell B2) and the insertion of content in form fields (cells B3 and B4).

Test Size and Execution Complexity Measurement. To estimate test execution effort, the tool measures the size and execution complexity of the test cases in execution points. The number of execution points of a test or test suite shows how big and complex it is. This measure is calculated by analyzing the test actions found in test specifications according to specific system characteristics. On the mobile application domain, examples of these characteristics are the number of screens to navigate the number of keys to press and delays in the application response time. More details on execution points and its measurement method are presented in [5].

These characteristics represent some general functional and non-functional requirements of the system under test that are exercised when the test is executed. The number of execution points assigned to a test action depends on its impact on the test execution complexity (Low, Average or High). Finally, the number of execution points of a test suite is the sum of execution points assigned to each test action found in the test specifications.

Test Execution Effort Estimation. Several models can be used to estimate test execution effort based on the number of execution points and cost drivers [4]. These cost drivers are factors usually related to the testing team and environment, such as team experience and tool support. Our tool allows the use of different estimation models, such as regression models or productivity-based estimation models. Regression models are equations that can regard execution points and the effect of the cost drivers to estimate the test execution effort. A productivity-based estimation is based on the average time per execution point spent during the tests execution (execution time / execution points).

Model Configuration. Before estimating test execution effort, it is necessary that the user configures the tool with the characteristics that should be used to measure the size and execution complexity of test specifications. In addition, the user needs to indicate which cost drivers should be used when estimating test execution effort. For both characteristics and cost drivers, the user has to indicate their impact (influence levels and weights) with respect to test execution complexity and effort. All this information may be specific to some application domains and it can be determined through Delphi panels [6]. Figure 2 shows an example of the configuration of system characteristics for the mobile application domain.

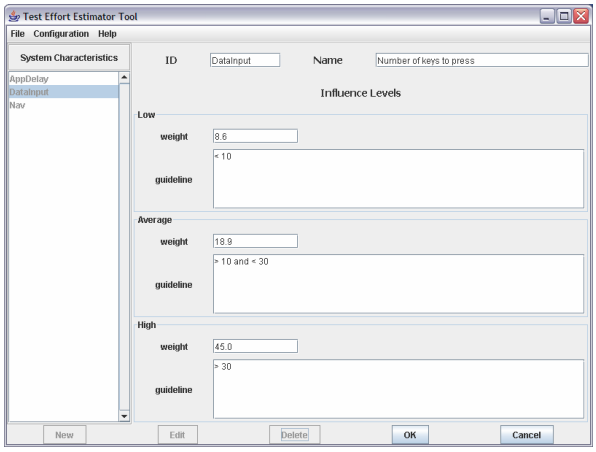

Figure 2. Configuration of the characteristics used to measure test size and execution complexity.

3. Tool Architecture

We developed the tool in Java using the architecture shown in Figure 3. In summary, the tool consists of the following components: Reader, Natural Language Processing, Model Configuration, Effort Estimation and Persistence Manager. These components are detailed next.

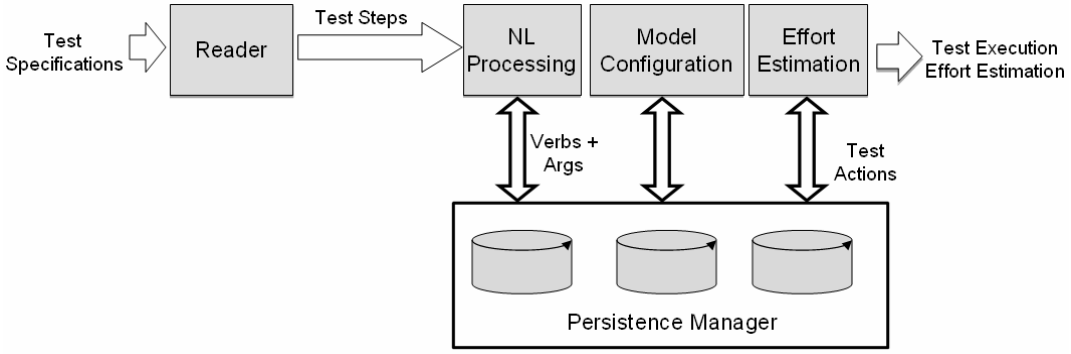

Figure 3. Architecture of the automated test execution effort estimation based on test specifications.

Reader: It is responsible for reading the files containing test specifications, extracting the test steps from each test specification. For that, this component has to manipulate the internal structure of the files.

Natural Language Processing: It is responsible for identifying the test action represented by each test step. First, the tool parses the sentence and identifies its main verb. Then, the tool converts the verb to its infinitive form, which is used to identify the test action. Next, the verb in the infinitive form is validated against a database that stores thousands of existing verbs of the English language. If the verb is not recognized, we consider that either the parsing or the conversion to infinitive form has failed. Since the English language is dynamic, the user is always enabled to add new verbs to the database. Validated verbs are then considered as test actions. When the tool finds a test action, it verifies if the test action is already stored in the test action database. This database contains all test actions previously processed by the tool. If the test action is not found in the database, it is stored.

Model Configuration: It is responsible for supporting the configuration of all information required to estimate the test execution effort, such as information about cost drivers, system characteristics, influence levels, weights and guidelines. In addition, the user has to evaluate the new test actions found in the test specifications, according to the system characteristics previously defined.

Effort Estimation: It is responsible for estimating the execution effort of a given suite of tests processed by the tool. For that, this component computes the number of execution points of the test suite and uses this measure to estimate the required test execution effort. All the information to compute execution points (characteristics,

weights, etc.) and to estimate test execution effort (estimation model parameters, cost drivers, etc.) are retrieved by the Persistence Manager.

Persistence Manager: It is responsible for the persistence of data, storing information about the model configuration (system characteristics, cost drivers, influence levels, weights) and the test actions found in the analyzed test cases.

3.1. Feature Model

The presented architecture makes easy the customization of our tool, according to the customer needs. The feature model [9] presented in Fig. 4 shows the possible extensions of each component of the tool architecture.

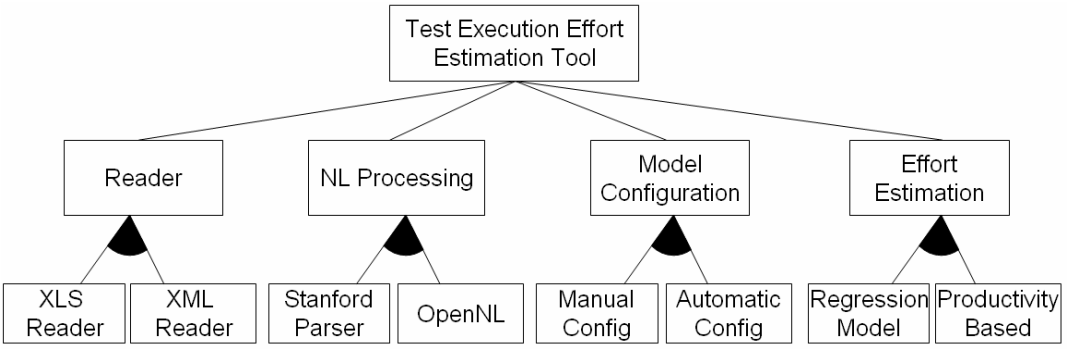

Figure 4. Feature model presenting how the tool can be extended.

At the time this paper was written, our tool was implemented as follows. The Reader component uses the Apache POI API [7] to read test specifications from Microsoft Excel files. The Natural Language Processing uses the Stanford Parser API [8] to parse test steps. The Model Configuration requires a manual configuration process based on expert opinion, although the tool can be extended by future works to automatically calibrate the model (define weights of characteristics and cost drivers) based on historical data. The Effort Estimation component supports both the regression and the productivity-based models.

4. Conclusion and Future Works

In this paper, we presented the characteristics of a tool developed to automate most part of a test execution effort estimation model that is based on test specifications. This automation support is essential to make feasible the test execution effort estimation based on the analysis of a high number of test specifications (may be hundreds) written in natural language.

We discussed the main functionalities and architecture components of the tool. Currently, the tool allows the processing of test specifications written in English to estimate test size and execution complexity, as well as test execution effort. This tool was already used to support some empirical studies, such as in [4], suggesting the viability of this tool.

Some of the drawbacks of this tool are the necessity to manually configure the measurement method of execution points and the estimation model based on regression analysis, for instance. In addition, it requires a manual evaluation of complexity for the

first time a test action is found in test specifications. However, previous empirical studies indicated that little effort is required for the manual configuration of the tool, since the vocabulary for writing test specifications is reduced (few verbs for hundreds of tests) [4].

There are some improvements in the tool that can be done in future works. One example is an automatic or semi-automatic model configuration, supporting the calibration of the estimation models through an automatic analysis of historical databases. The automation support of effort estimation provided by this tool can be integrated with model-based testing tools. This integration can support the use of execution effort as an additional criterion for limiting the number of test cases to be generated.

Acknowledgments

We thank the anonymous reviewers for their comments and suggestions. The first author is partially supported by Motorola, grant BCT-0021-1.03/05, through the Motorola Brazil Test Center Research Project. The last author is partially supported by CNPq, grant 306196/2004-2.

References

[1] Myers, G. J. (2004) "The Art of Software Testing". John Wiley Sons, Inc.

[2] Boehm, B., Horowitz, E., Madachy, R., Reifer, D., Clark, B., Steece, B., Brown, W., Chulani, S. and Abts, C. (2000) "Software Cost Estimation with COCOMO II". Prentice Hall.

[3] Nageswaren, S. (2001) "Test Effort Estimation Using Use Case Points". In: Proceedings of the 14th International Internet & Software Quality Week.

[4] Aranha E. and Borba P. (2007) "An estimation model for test execution effort", In: Proceedings of the 1st International Symposium on Empirical Software Engineering and Measurement.

[5] Aranha E. and Borba P. (2006) "Measuring test execution complexity", In: 2nd Intl. Workshop on Predictor Models in SE.

[6] Linstone, H. and Turoff, M. (2002) "The Delphi Method: Techniques and Applications". http://is.njit.edu/pubs/delphibook

[7] Apache POI: Java API To Access Microsoft Format Files. (2007) http://poi.apache.org/

[8] The Stanford Parser: A statistical parser. (2007) http://nlp.stanford.edu/software/lex-parser.shtml

[9] Riebisch, M., Streitferdt, D. and Pashov, I. (2004) "Modeling Variability for Object-Oriented Product Lines". Springer Berlin.