Universidade Federal de Pernambuco
Centro de Informática

Pós-graduação em Ciência da Computação

Estimating Test Execution Effort Based on Test Specifications

Eduardo Henrique da Silva Aranha

Tese de Doutorado

Recife
Janeiro de 2009

Universidade Federal de Pernambuco
Centro de Informática

Eduardo Henrique da Silva Aranha

Estimating Test Execution Effort Based on Test Specifications

Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Doutor em Ciência da Computação.

Orientador: *Prof. Paulo Henrique Monteiro Borba*

Recife
Janeiro de 2009

Aranha, Eduardo Henrique da Silva

Estimating test execution effort based on test specifications / Eduardo Henrique da Silva Aranha. - Recife: O Autor, 2009.

xxii, 168 p. : il., fig., tab.

Tese (doutorado) – Universidade Federal de Pernambuco. CIn. Ciência da Computação, 2009.

Inclui bibliografia e apêndice.

1. Engenharia de software. 2. Medição de software. 3. Teste de software. I. Título.

005.1 CDD (22. ed.) MEI2009 - 156

Tese de Doutorado apresentada por **Eduardo Henrique da Silva Aranha** a Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título **"Estimating Test Execution Effort Based on Test Specifications"** orientada pelo **Prof. Paulo Henrique Monteiro Borba** e aprovada pela Banca Examinadora formada pelos professores:

Prof. Paulo Jorge Leitão Adeodato
Centro de Informática / UFPE

Profa. Renata Maria Cardoso Rodrigues de Souza
Centro de Informática / UFPE

Prof. Cristiano Ferraz
Departamento de Estatística / UFPE

Profa. Patricia Duarte de Lima Machado
Departamento de Sistemas e Computação / UFCG

Prof. Manoel Gomes de Mendonça Neto
Departamento de Engenharia e Arquitetura / UNIFACS

Visto e permitida a impressão
Recife, 21 de janeiro de 2009

Prof. FRANCISCO DE ASSIS TENÓRIO DE CARVALHO
Coordenador da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.

To my family.

Acknowledgements

✓ First of all, I would like to thank God, for giving me the strength required to complete this important stage of my life, and my parents, wife and daughter, for their support and patience.

✓ I would like to thank Paulo Borba, my advisor, for his great instructions and support during this research.

✓ Thanks to the professors Cristiano Ferraz, Renata Souza and Carla Monteiro, for their teaching and support given during this research.

✓ Many thanks to all professors, students and employees of the BTC Research Project and the Software Productivity Group for their valuable comments about my work and for their support during my empirical studies.

✓ Thanks to Luís Cláudio, Rogério Monteiro, Edson Fontes, André Lacerda, Alexandre Mendonça and the other employees and scholarship holders at the CIn/Motorola program that collaborated during this research.

✓ Thanks to the students Filipe de Almeida, Thiago Diniz, Vitor Fontes and Guilherme Carvalho, for their hard work during the implementation of tools and scripts used in this research.

✓ Thanks to the students Eduarda Freire, Raquel Cândida and Veristiana Carvalho, for their help during the execution of the empirical study ES4.

✓ Thanks to the students André Ribeiro, Fernando Souza and Ivan Machado, for their help during the execution of the empirical study ES5.

✓ Thanks to the student Roberto Ferreira, for their help during the evaluation of alternative experimental designs for the empirical study ES6.

✓ The suggestions of well-known researchers in the area, such as Barry Boehm, Emilia Mendes, Ricardo Valerdi, Andrea de Lucia and Carmine Gravino, as well as comments from the anonymous reviewers of our papers allowed us to identify and explore several opportunities in our research. We are grateful to them for their valuable comments.

✓ Thanks to the Motorola Brazil Test Center and the National Council of Technological and Scientific Development (CNPq) for their financial support during my doctorate.

Resumo

Em mercados competitivos como, por exemplo, o de celulares, empresas de software que liberam produtos com baixa qualidade podem rapidamente perder os seus clientes. A fim de evitar esse problema, essas empresas devem garantir que a qualidade dos produtos atenda a expectativa de seus clientes. Nesse contexto, testes é uma das atividades mais utilizadas para se tentar melhorar a qualidade de um software. Além disso, o resultado da atividade de teste está sendo considerado tão importante que em muitos casos é preferível alocar equipes exclusivamente para exercer atividades de teste. Essas equipes de teste devem ser capazes de estimar o esforço exigido para exercer as suas atividades dentro do prazo ou para solicitar mais recursos ou negociar prazos quando necessário. Na prática, as consequências de se ter estimativas ruins são onerosas para a organização: redução de escopo, atraso nas entregas ou horas extras de trabalho. O impacto dessas consequências é ainda maior em se tratando de execução manual de testes.

Visando uma melhor forma de estimar esforço de execução manual de casos de teste funcionais, esta pesquisa propõe e valida uma medida de tamanho de teste e de complexidade de execução baseada nas próprias especificações dos testes, bem como um método de medição para a métrica proposta. Além disso, diversos estudos de caso, survey e experimentos foram realizados para avaliar o impacto desse trabalho. Durante esses estudos, verificamos uma melhoria significativa proporcionada por nossa abordagem na precisão das estimativas de esforço de execução de testes manuais. Também identificamos fatores de custo relacionados a atividades de execução manual de testes utilizando julgamento de especialistas. O efeito desses fatores foram investigados através da execução de experimentos controlados, onde pudemos constatar que apenas alguns dos fatores identificados tiveram efeito significativo. Por fim, diversas ferramentas de suporte foram desenvolvidas durante essa pesquisa, incluindo a automação das estimativas de esforço de execução de testes a partir de especificações de testes escritas em linguagem natural.

Palavras-chave: Execução de testes, estimativa de esforço, especificação de testes, métricas de tamanho, fatores de custo, engenharia de software experimental.

Abstract

In competitive markets like the mobile phone market, companies that release products with poor quality may quickly loose their clients. In order to avoid this, companies should ensure that product quality conforms to their clients' expectations. A usual activity performed to improve quality is software testing, which is the act of exercising the software with the objective to detect failures during its execution. In addition, software testing is being considered so important that organizations can allocate teams exclusively for testing activities in order to achieve unbiased test results. These test teams are usually requested to test features of different software applications. Mainly 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 practice, the consequences of making poor estimates are costly to the organization: scope reduction, schedule overrun and overtime. The impact of these consequences are more evident when regarding manual test execution.

In order to better estimate the effort required to execute a given tests or test suite, this research proposes and evaluates a measure for test size and execution complexity based on the test specifications written in a controlled or standardized natural language. We also define and evaluate a measurement method for our proposed measure. Several empirical studies were executed to evaluate this work. The results suggests a significant accuracy improvement in effort estimates for manual test execution. We also identified and investigated cost drivers related to manual test execution through the use of expert judgement and designed experiments. Experts highlighted several cost drivers, but we observed that only few of them had significative effect on test execution effort in our studies. Finally, several tools were developed to support this research, including one to automatically estimate test execution effort based on test specifications written in natural language.

Keywords: Test execution, effort estimation, test specification, size measures, cost drivers, empirical studies, experimental software engineering.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

In competitive markets like the mobile phone market, companies that release products with poor quality may quickly loose their clients. In order to avoid this, companies should ensure that their products conform to the their specifications. A usual activity performed to improve quality is software testing, which is the act of exercising the software with the objective to detect failures during its execution [68].

Software testing is an important activity that usually requires a significant effort. There is a not so old 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 [104]. Although the development of new testing technologies reduced this impact over the time, software testing still demands significant effort.

For this reason, several techniques are being developed not only to improve the effectivity of the tests, but also to improve the test process efficiency. For instance, Model-Based Testing (MBT) can help us to improve test coverage by automatically generating tests from software specifications [46] [116] [109]. Although the use of MBT may reduce the costs of test design, the effort for executing the generated tests is still significant. For instance, the use of MBT or similar technologies to automatically generate tests usually achieves better test coverage (more tests and more paths covered), perhaps increasing the number of tests and the effort required to execute them, mainly when tests have to be manually executed due to restrictions such as:

- Short time to market, where the time required to automate tests is only acceptable for software maintainance, but not for the first software release.
- Tests of multimedia (sounds, video, etc.) and hardware features that have limited support of existing test automation frameworks.

In addition, software testing is being considered so important that organizations can allocate teams exclusively for testing activities in order to achieve unbiased test results [35]. These test teams are usually requested to test features of different software applications. Mainly 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.

A review of surveys [102] on estimation of software development effort showed the difficulty to have accurate estimates. Some of the practical consequences (observed in industry and reported by these surveys) of having poor estimates in testing projects are:

- Scope reduction: in order to keep the planned schedules and budget, only part of the tests are executed. This means that only parts of the software will be tested, increasing the chance of having escaped defects (defects not found during a test phase).

- Schedule overrun: in this case, the testing activities take more time than planned, increasing the costs and delaying the delivery of software releases into the market. In some high competitive markets, this option is not viable.
- Overtime: the testing phase is finished on schedule, but exceeding the regular working hours. Here, the cost may be high, as well the negative impact on team motivation.

All these consequences are costly and should be avoided.

1.1 Problem Statement and Research Hypotheses

Although the use of MBT or similar technologies can help us to reduce the test design effort, the effort to execute these tests is still significative, mainly when they have to be executed manually. To avoid the consequences of having poor estimates, the effort required for executing tests manually should be properly estimated by managers of independent test execution teams. For that, it is necessary to have good measures related to test execution effort and accurate estimation models.

Several estimation approaches and models have been proposed over the years for estimating software development effort [27], such as Function Point Analysis (FPA) [49] and COCOMO [28]. Regarding testing effort, Test Point Analysis [105] is an approach similar to FPA used for estimating the effort of the whole testing activities of a given information system. Although effort estimates for the whole development and testing activities are important, they are not much useful for managers of independent test execution teams, since these estimates are not appropriate to determine the effort that will be spent by test teams for providing their services: execution of specific set of tests, according to what need to be tested in the software, etc. For this reason, test execution effort estimates made by test managers are usually based on historical averages (usually imprecise) or expert judgment (generally subjective and not repeatable) [19], leading to the practical consequences mentioned above.

Also, there are several software development cost drivers (factors that have significant impact on development costs, such as developer experience and software complexity) reported in several papers and used by several software development estimation models to increase effort estimation accuracy, but few of them are related to test execution. In fact, some cost drivers for test execution can also be particular to specific industrial settings.

In summary, our research hypotheses are:

- It is possible to have a sound measure for the size and execution complexity of test specifications;
- The use of such measure will improve the estimation accuracy of manual test execution effort;
- By regarding cost drivers related to manual text execution, we will also improve the estimation accuracy of manual test execution effort;
- The measurement of test size and execution complexity and the estimation of manual test execution effort can be automated, reducing the costs of this approach.

1.2 Context

This work is part of a multi-disciplinary research initiative [136] in partnership with Motorola Industrial Ltda., an important software industry on the mobile application domain for developing new software testing technologies, such as the automatic generation of tests based on software specifications. In particular, the focus of this research is to increase the accuracy of test execution effort estimates and to automate the estimation process. In this way, we want to be able not only to automatically generate test specifications, but also to automatically estimate the effort required to execute tests manually.

In this work, we consider big organizations with dedicated and independent test execution teams (more than 10 testers per team, for instance) executing functional tests manually for several software development teams, since we observed that these teams are more impacted by the investigated problem and usually are responsible for most part of the total testing effort of a project. For this reason, we do not investigate in this work the effort estimation for other related testing activities, such as test design, setup of test environment and defect tracking. Also, each manual test execution on the mobile application domain usually requires the participation of only one tester, which means that the effort in man-hours required to execute a test is usually the time required to execute it.

Finally, we consider that software development teams request the services provided by test execution teams when they deploy internal or commercial software releases. Hence, test execution effort estimates should be performed when these requests arrive. We also consider that the tests for the product are already written in a more structured natural language (by model-based testing tools or by manual test design) and only a subset of the tests will be executed, according to the functionalities created and modified in the current software release.

1.3 Summary of Goals

This research has the objective of creating an estimation approach that can be used for estimating the effort to execute a given test case or test suite. This novel estimation approach is also intended to improve the estimation accuracy achieved for estimating manual test execution effort when using current estimation practices (historical averages, etc.).

To achieve our main objective, this research has the goal of proposing a measure for test size and execution complexity based on test specifications written in natural language. As size is usually the most important variable in estimation models [29], this proposed measure should improve the accuracy of test execution effort estimates. We also need to define and evaluate a measurement method for the proposed measure.

To better improve the accuracy of the estimation of manual test execution effort, we want to regard not only the size and execution complexity of test specifications, but also the effect of cost drivers related to test execution. Hence, we also have the goal of identifying these cost drivers and investigating their effects on test execution effort. Finally, we want to calibrate and evaluate our proposed estimation approach through empirical studies on the mobile application domain.

1.4 Methodology

Size is usually the most important and most used variable in effort estimation models. Hence, it is important to know how to measure software size. For this reason, we first review the existing measurement methods of software size. Since we want to propose a better way to estimate test execution effort, we also review the existing software effort estimation models described in the literature. Then, we define a size measure and a measurement method for test size and execution complexity. During our empirical studies, this measure is evaluated in terms of validity of construction, practical use and benefits for the accuracy of test execution effort estimation.

We define some test execution effort estimation models based on our proposed measure of test size and execution complexity and the existing estimation model techniques. After that, we run a sequence of empirical studies to identify the relevant variables to include in the proposed test execution effort estimation models, to evaluate them and the achieved accuracy. We evaluate the relevance of some cost drivers for executing tests by observing their impact on effort when considering different test scenarios. Finally, to achieve better precision in our empirical studies, we develop specific tools to support the collection and analysis of test execution data.

1.5 Summary of contributions

During this research, we produced the following contributions:

- The proposal and evaluation of a measure for test size and execution complexity that is based on test specifications written in natural language. This characteristic supports the estimation of the effort to execute a given set or subset of functional tests only based on their specifications, being applicable even for tests never executed before.
- The definition and implementation of an automated measurement method for sizing test specifications.
- The proposal and evaluation of an estimation approach based on our proposed measure for test size and execution complexity.
- Use of systematic methods for configuring the proposed measurement method for specific application domains, which includes the identification and weighting of characteristics in test specifications that better explain test execution effort.
- Use of systematic methods for identifying and weighting cost drivers related to test execution that can be particular to specific industrial settings.
- The configuration of the proposed measurement method and the investigation of cost drivers related to test execution for the mobile application domain.
- The development of a tool for measuring test size and for estimating test execution effort based on the test specifications.

1.6 Thesis organization

The remaining of this document is organized as follows:

- Chapter 2 presents a review of software size measurement methods. It shows the main software size measures and measurement methods that were proposed in the literature and used in well-known software estimation models.
- Chapter 3 presents a review of existing effort estimation models. This review is mainly based on books and papers that describe and compare estimation techniques, reporting their advantages and limitations.
- In Chapter 4, we present the overall planning of our research, as well the development of a measure for test size and execution complexity and estimation models for test execution effort based on the proposed measure.
- Chapter 5 presents the planning, execution and results of our empirical studies used to: evaluate our proposed size measure for test size and execution complexity; evaluate the accuracy of estimation models for test execution effort when using our proposed measure; identify cost drivers for test execution and investigate their effects on test execution effort.
- Chapter 6 relates this research to other works found in literature.
- Chapter 7 presents the final conclusions of this work, presenting opportunities for future work.

CHAPTER 2

Software Size Measurement Methods

Size is usually the most important and most used variable in effort estimation models [29]. In this chapter, we show the main software size measures and measurement methods that were proposed and used in software estimation models.

2.1 Analysis Criteria

To analyze the different software size measurement methods reported in the literature, we defined the criteria shown in Table 2.1. These criteria represent characteristics that should be considered when selecting a software size measure for estimating effort based on the project size.

Table 2.1 Set of criteria used to evaluate software size measurement methods.

Criterion	**Description**
Artifact	Type of artifact that is measured.
Attribute	What characteristic of the artifact is being measured.
Restriction	Any restriction that is applicable to the measured artifact.
Availability	Phase or time where the artifacts are available and the metric can be collected.
Standardized	Describe if there are well-defined measurement procedures and a consensus (standard) in the community.
Deterministic	Describe if different people can obtain different results when measuring the same artifacts.
Measurement cost	The cost to obtain the measure.
Calibration cost	The cost to calibrate the measurement method, if necessary.
Required background	Describe if there is any dependency of expertise for ensure the quality of the measure/measurement method.
Automation	Capability of the measurement method to be automated.
Validity	Describe if there is any critique about the validity of the measure.

The main sources used to find the presented software size measures and measurement methods were books, papers and surveys related to software metrics and effort estimation models.

2.2 Source Lines of Code (SLOC) Counting

There are two types of SLOC, the Physical SLOC and the Logical SLOC [113]. Physical SLOC depends on the style and format of the programming language. It may include comments and blank lines and it is easer to count. Logical SLOC attempts to measure only the number of statements and it is harder to count. Some advantages of SLOC are:

- The counting can be automated, although the tools are usually developed for specific languages due to their different syntaxes and structures. Examples of tools are the CodeCounter [39] and SLOCCount [130].
- It is an intuitive metric, since bigger programs have more SLOCs than smaller ones.

However, there are also some problems with SLOC:

- The number of SLOCs is dependent upon the skill level of the programmer. For instance, skilled programmer may write fewer lines of code for the same functionality when compared to a beginner programmer.
- For the same application, the support of the programming language (structures, instructions, etc.) affect its number of SLOCs.
- Lack of counting standards. A recent initiative is trying to define what to consider when counting SLOCs [108].
- It is difficult to have early estimates of the total number of SLOCs of the application to be developed [83][103].
- The use of code generators can automatically generate a great amount of source code, reducing the correlation between SLOC and development effort.

2.3 Function Points Analysis (FPA)

Function point [9] was developed by Allan Albrecht of IBM as a standardized metric for measuring the functions of a software application from the user's point of view. It provides a technology independent estimate of the size of the final program, since it is related to specifications rather than code. His first publication about FP was in 1979 [7] and an extended version was published in in 1983 [9]. Then, in 1984, the International Function Point Users Group (IFPUG) [57] was set up to clarify the rules, set standards, and promote their use and evolution.

In industry, Function Points Analysis (FPA) is one of the most known measurement process used for measuring software size based on the software specification [49] [82]. Since its initial description in 1979, this method was evolved as shown in Table 2.2. More details about the improvements found in each version can be seen in [5]. In addition, IFPUG Functional Points Analysis is an ISO certified functional sizing method.

FPA measures function points in 7 steps [49] that are described next.

Table 2.2 Official versions of the function points analysis.

Year	Version
79	Albrecht 79 [7]
83	Albrecht 83 [9]
84	GUIDE 84 [8]
86	IFPUG Release 1.0 [58]
88	IFPUG Release 2.0 [59]
90	IFPUG Release 3.0 [60]
94	lFPUG Release 4.0 [61]

Step 1: Determine the type of counting.

Three different types of function point counts were defined to represent the three major types of software projects:

- *Development:* this type of count is associated with the development of new software applications.
- *Enhancement:* this type of count tries to size the enhancements to be done in applications already in production.
- *Application:* this type of count is done on applications already developed, usually as a "baseline count".

Step 2: Identify the counting scope and application boundary.

The counting scope is determined by the purpose of the count, identifying the systems, applications or parts of applications that will be sized. The application boundary is the border between what is being measured and external applications and users.

Step 3: Identify all data functions and their complexity.

The data functions are classified as Internal Logical Files (ILF) and External Interface Files (EIF). ILF is a group of logically related data or control information that is perceived by the user and maintained inside the application boundary. EIF is a group of logically related data or control information that is perceived by the user but maintained inside the boundary of another application. For instance, an EIF counted for an application may be an ILF for another application.

FPA defines guidelines to identify ILFs and EIFs. Then, their complexity are calculated by counting the number of Data Element Types (DET) and Record Element Types (RET). DETs are unique user-recognizable non-repeatable fields or attributes. RETs are user-recognizable

subgroups of data elements contained whitin ILFs or EIFs. There are also guidelines defined to identify DETs and RETs in the FPA documentation.

The complexity of the data functions are given in a Low, Average and High scale, according to the counts of DETs and RETs (see Table 2.3). Then, these ordinal values are transformed in numerical values as presented in Table 2.4. To illustrate this transformation, let us suppose the counts RETs=7 and DETs=3 for a given data function. According to Table 2.3, the complexity of this data function is rated as Average. For an ILF data function, this result corresponds to the weight 10 (see Table 2.4).

Table 2.3 Complexity matrix for ILFs and EIFs.

RETs	DETs		
	1-19	20-50	> 50
1	Low	Low	Average
2-5	Low	Average	High
> 5	Average	High	High

Table 2.4 Weights used for data functions.

Rating	Weights	
	ILF	EIF
Low	7	5
Average	10	7
High	15	10

Step 4: Identify all transaction functions and their complexity.

Transaction functions perform update, information retrieval, output and so forth. They are classified in External Inputs (EI), External Outputs (EO) or External Inquiries (EQ). EIs are elementary processes that process data or control information entered from the outside of the application boundary. EOs are elementary processes that generate data or control information to outside of the application boundary. EQs are elementary processes that retrieve data or control information from ILFs or EIFs to the outside of the application boundary.

DETs and File Types Referenced (FTR) are used to determine the transaction function complexity. FTR is the number of files referenced or updated in the transaction. There are other tables (see [49]) similar to Tables 2.3 and 2.4 with the complexity matrix and weigths for EIs, EOs and EQs.

Step 5: Determine the unadjusted function point count.

The unadjusted function point count is the sum of the points assigned to data functions and transactions functions.

Step 6: Determine the value adjustment factor.

The value adjustment factor (VA) summarizes 14 General System Characteristics (GSC) defined by FPA. When applied, the value adjustment factor can adjusts the functional size by a maximum adjustment of $\pm 35\%$ to produce the adjusted function points (AFP). This is considered one limitation of the FPA measurement method.

The list of GSCs includes complex processing, reusability, installation ease and the development in multiple sites. Each GSC is evaluated on a scale of 0 to 5. This value is called degree of influence. The sum of all GSC rates is called Total Degree of Influence (TDI). VAF is calculated based on the following formula:

$$VAF = (TDI * 0.01) + 0.65 \tag{2.1}$$

Step 7: Calculate the adjusted function point count.

Finally, the adjusted function point count is calculated by multiplying the number of unadjusted function points by the value adjustment factor (VAF).

According to Garmus and Herron [49], counting points with FPA has the following advantages:

- It is a language and implementation-independent metric.
- Available at an early stage of the software development cycle, since it is based on the software specification.
- It has an international and independent group promoting its use and evolution.

However, there are also some problems in this approach:

- The reasons for the assignments of specific values (weights) are not clear.
- Lack of a clear definition. As we can see in [4], different authors had different interpretations of FP: a measure of size, productivity, complexity, functionality or user deliverables.
- The expertise in the assessment of complexity, the interpretation of the specification, value judgments, perception of the object boundaries and other aspects may causes a variation in the counts. Experience in the application of function points is then an important factor in their successful application [83].

2.3.1 Critiques about FPA model construction

The validity of Function Points Analysis (FPA) with respect to the measurement theory was questioned by several researchers. In [4], the authors suggested that the function points interpretation should be reconsidered from a measurement perspective, addressing issues in the expert judgments and measurement scales and transformations throughout the measurement steps.

Barbara Kitchenham discussed in [72] about problems in the model construction. She pointed that absolute scale counts are reduced to ordinal scale measures and, for this reason, you can no longer apply other transformations to make them interval or ratio scale. Formally, we cannot add the label “simple" to the label “complex", even when using numbers (3, 5, etc.) as a synonym for them. Also, since we cannot multiply or divide a ordinal value, we cannot convert them to productivity measures.

Another reported problem with function points is the technology adjustment factor, that is based on a subjective assessment of 14 project factors on a six-point ordinal scale. In addition, some researchers found correlations between Albrecht function point elements. Actually, [73] and [66] found different correlations, suggesting that predictive models based on the sum of the elements will not be stable for different datasets [72]. Also, Kitchenham and Känsälä identified that some function point elements were not related to effort [73].

For instance, they verified that an effort prediction model based on only two function point elements (input function points and output function points) had a similar performance of an effort model based on total function points. Also, the authors achieved almost the same result using stepwise regression and raw counts of the number of files instead of the total function points.

Another criticism is about the low interrater reliability of FP counts. Interrater reliability is the extent to which two or more individuals agree. In this case, the extent to which two individuals measuring the size of the same system using FPA would generate the same FP count. However, the function points measurement interrater reliability appears to be sufficiently high, posing no practical barrier to their adoption in the industry [71].

2.4 COSMIC

COSMIC [41] [133] is a method for measuring COSMIC Function Points (CFP) that is considered the new generation of functional sizing [137]. It was developed in the COSMIC (Common Software Measurement International Consortium) project and it is an attempt to solve the problems of its main predecessor, FPA.

Its official documentation [41] includes the types of software for which the method has been designed to measure functional size (domain of applicability), the software models used for measurement and the COSMIC measurement process. COSMIC Functional Points Analysis is also an ISO certified functional sizing method [62].

Using COSMIC, we measure the functional size of a piece of software in three distinct and related phases:

1. Setting the measurement strategy using the principles of the Software Context Model

(Table 2.5), establishing:

- The purpose of the measurement.
- The scope of each piece of software to be measured;
- The functional users and the boundary of each piece of software.
- The required level of granularity for the measurements.

2. Mapping the artifacts of the software to be measured onto the Generic Software Model (Table 2.6), which includes the following steps:
   - Identify the events of the functional users that the software must respond and then the functional processes.
   - Identify the data movements (Entries, Exits, Reads and Writes) of each functional process and data groups that are moved.
3. Measuring the specific elements of this model using a defined set of rules and processes.

Basically, the COSMIC size measurement is based on the following principle:

> "The functional size of a piece of software is directly proportional to the number of its data movements." (COSMIC Method, Version 3.0 [41])

Also, the measurement rules and process follows the characteristics listed in Table 2.7.

In addition to the same advantages of FPA, the COSMIC measurement method of functional size does not classify data using an ordinal scale. Also, it does not include weights to represent software complexity. For these reasons, it does not have most of the problems reported against of FPA. However, we did not found any published work that demonstrated the validity of the measure with respect to measurement theory.

2.5 Test Points Analysis (TPA)

Test Point Analysis (TPA) is method to measure the volume of testing work to be undertaken in a software development project [105]. This volume of work is expressed in a unit-of-work called Test Points (TP). The TPA measurement method is an extension of Function Point Analysis for the testing development phase.

The total number of test points of an information system is calculated by adding the number of test points necessary for testing the dynamic and the static measurable (testable) quality characteristics of the system.

Dynamic Test Points

The number of test points necessary for testing the dynamic measurable quality characteristics of the system assigned to each function of the system includes:

- The number of function points assigned to it using an adaptation of FPA or other two alternative models (FP).
- Influential factors divided into two categories:
  - Function-dependent (D_f): user-importance (Ue), usage-intensity (Uy), interfacing (I), complexity (C) and uniformity (U).
  - Quality requirements related to the dynamic quality characteristics to be tested (Q_d): suitability, security, usability and efficiency.

All function-dependent factors are rated in Low, Average or High according to some guidelines. Different weigths are associated for each factor. The following formula is used to calculate D_f:

$$D_f = \frac{Ue + Uy + I + C}{20} * U \tag{2.2}$$

where 20 is the sum of all nominal rate values of Ue, Uy, C and I.

The factors related to quality requirements are rated in 0, 3, 4, 5 or 6, according to is importance in the project. Q_d is calculated as follows. The rating for each factor is divided by four (the nominal rating), then multiplied by a weighting factor. Finally, Q_d is the sum of all these values added together.

Finally, the number of dynamic points of a function (TP_f) is obtained by the following formula:

$$TP_f = FP_f * D_f * Q_d \tag{2.3}$$

where FP_f is the number of function points assigned to the function f of the information system.

Static Test Points

The static measurable quality characteristics (Q_i) are based on the ISO 9126 quality characteristics [63]. For each quality characteristic to be observed during the test, the value sixteen is added to the Q_i factor rating.

Total Test Points

Finally, the total number of test points of an information system is calculated by the following formula:

$$TP = \Sigma TP_f + \frac{FP * Q_i}{500} \tag{2.4}$$

TPA includes an effort estimation model similar to the used in Function Point Analysis. It defines environment factors (test tools, test environment, development environment, etc.)

that should be rated according some guidelines. The sum of these environment factors is then multiplied by the total number of test points. This adjusted counting is then multiplied by the current test productivity (number of test hours required per test point). The author suggested that the productivity is generally a value between 0.7 and 2.0.

The result of a TPA is an estimate for the complete test process, excluding test planning. An effort distribution between phases can be done using historical data. According to his experience, the author reported the following effort distribution:

Although the author makes not clear how this model was built and calibrated, this model appears to have the same problems of FPA cited in 2.3.1. Also, we did not found any published work reporting the use of this model and presenting results from empirical studies, achieved accuracy, etc. For instance, there is no reference about in which situations this model was applied. For instance, the effort distribution shown in Table 2.8 may vary significatively when using MBT [116], since the test design effort can be reduced.

2.6 Other Measurement Methods

There are other extensions of function points analysis. MkII FPA [64] and NESMA FSM [107] are variations of FPA that are also certified by ISO as international functional size measurement standards. FAST Function Points [128] is an extension that tries to optimize the counting steps in order to reduce counting effort.

The Use Case Point Analysis (UCP) [101] is an extension of FPA and estimates the size of a system based on use case specifications. Object-Counts and Object-Points are object-based output measures [23].

Finally, we can count the number of tests in a test suite and the number of test procedures in test specifications. These two measures can be used to estimate respectively the size of a test suite and the size of a test specification.

2.7 Final Considerations

This chapter presented the software size measurement methods more related to this work, which includes SLOC Counting, Function Points Analysis, COSMIC, Test Point Analysis and others. Each metric has its specifics advantages and limitations. For instance, SLOC is difficult to estimate in an early stage of development and FPA has several critiques about its construction process. Table 2.9 summarizes the main characteristics of these measurement methods according to the criteria defined in Section 2.1.

In the next chapter, we review the existing estimation models and after that we present our proposed test size measure and its measurement method in Chapter 4.

Table 2.5 List of principles of the software context model defined in COSMIC.

Principles of the Software Context Model
• Software is bounded by hardware. • Software is typically structured into layers. • A layer may contain one or more separate 'peer' pieces of software and any one piece of software may further consist of separate peer components. • Any piece of software to be measured, shall be defined by its measurement scope, which shall be confined wholly within a single layer. • The scope of a piece of software to be measured shall depend on the purpose of the measurement. • The functional users of a piece of software shall be identified from the functional user requirements of the piece of software to be measured as the senders and/or intended recipients of data. • A piece of software interacts with its functional users via data movements across a boundary and the piece of software may move data to and from persistent storage within the boundary. • The functional user requirement (FUR) of software may be expressed at different levels of granularity. • The level of granularity at which measurements should normally be made is that of the functional processes. • If it is not possible to measure at the level of granularity of the functional processes, then the FUR of the software should be measured by an approximation approach and scaled to the level of granularity of the functional processes.

Table 2.6 List of principles of the generic software model defined in COSMIC.

Principles of the Generic Software Model
• The application receives input and produces output (or another outcome) for the functional users. • Functional user requirements can be mapped into unique functional processes. • Each functional process consists of sub-processes. • Sub-processes are data movements or a data manipulations. • Events cause functional users to trigger a functional process by an Entry data movement. • A data movement moves a single data group. • A data group consists of a unique set of data attributes that describe a single object of interest. • There are four types of data movement: – An Entry moves a data group into the software. – An Exit moves a data group out of the software. – A Write moves a data group from the software to persistent storage. – A Read moves a data group from persistent storage to the software. • A functional process shall include a minimum of two data movements, including at least one Entry data movement and either a Write or an Exit data movement. • For measurement purposes, data manipulation subprocesses are not separately measured. It is assumed to be accounted for by the data movement with which it is associated.

Table 2.7 Characteristics of the rules and processes used in the COSMIC measurement method.

Characteristics
• 1 CFP (COSMIC Function Point)is equivalent to a single data movement. • The functional size of a functional process is defined as the arithmetic sum of the number of its constituent data movements . • The functional size of any required functional change(s) to a piece of software is by convention the arithmetic sum of the number of its data movements that must be added, modified and deleted as a consequence of the required change(s). • The minimum functional size for a single functional process is 2 CFP, because the smallest functional process must have at least one Entry (as input), and either one Exit (as output) or one Write (as an alternative useful outcome). As a change may affect only one data movement, it follows that the minimum size of a change to a functional process is 1 CFP.

Table 2.8 Test effort distribution according to the author of Test Point Analysis.

PHASE	EFFORT DISTRIBUTION
Preparation	10%
Specification	40%
Execution	45%
Completion	5%
TOTAL	100%

Table 2.9 Summary of the software size measurement methods reviewed in this chapter.

	SLOC	FPA	COSMIC	TPA	No. of Tests	No. of steps
Artifact	Source code	Requirements	Requirements	Requirements	Test suite	Test specification
Attribute	Application size and complexity	Application size and complexity	Application size and complexity	Volume of test	Test suite size	Test size
Restriction	Language specific			Requirements written as use cases		
Availability	After implementation	After requirement specification	After requirement specification	After requirement specification	After test specification	After test specification
Standardized	There is some work in progress	Yes	Yes	Yes	Yes	Yes
Deterministic	Yes	No, result depends on the expertise in the measurement method	No, result depends on the expertise in the measurement method	No, result depends on the expertise in the measurement method	Yes	Yes
Measurement cost	None for languages supported by tools	High, since there is a lot of manual effort	Very high, since there is a lot of manual effort and it is more abstract method	High, since there is a lot of manual effort	None when supported by tools	None when supported by tools
Calibration cost	Not required	Significant, but not required	Significant, but not required	Significant, but not required	Not required	Not required
Required backgound	None when supported by tools	Knowledge in the measurement method and measured application	Knowledge in the measurement method and measured application	Knowledge in the measurement method and measured application	None	None
Automation	Yes	No	No	No	Yes	Yes
Validity	Same application can be implemented with significant different SLOC, depending on the language, developer, etc.	There are several critiques (see Section 2.3.1)	Result depends on who is measuring	Problems similar to FPA	Tests can have significantly different sizes and complexity	Steps can have significantly different execution complexities. Same test can have different measures depending on the level of detail used by the test designer

CHAPTER 3

State of Art in Effort Estimation

As presented in Chapter 1, we aim to develop and evaluate models that can be used for estimating test execution effort. This chapter presents the state of art in estimation models, including a general classification for the existing models. Due to the large number of models, we comment here only the models more related to our context.

3.1 A General Classification for Effort Estimation Models

There are several papers and books in the literature classifying and describing existing estimation models and techniques, such as [27], [67], [102], [110] and [125]. As discussed in [27], no single estimation technique can be considered the best for all situations. For this reason, it is important to know the strengths and limitations of these techniques to better select them to produce more realistic estimates.

We present in Table 3.1 a general classification that we considered appropriate for classifying existing software estimation models. Then, we overview the most related estimation models in the next sections, mainly those more close to our research: average effort, regression analysis, Delphi and COCOMO models.

3.2 Productivity-Based Models

A simple way to estimate effort is to analyze productivity in historical databases, which is calculated observing the relation between the outputs (what is produced) and inputs (demanded effort). Although very limited, this method for making estimates is still being used in the practice, probably due to its simplicity and lack of viable alternatives.

This estimation approach has a significant limitation. Estimates are accurate only if the variance of the test productivity is small, that is, the productivity is almost constant. In other words, there is no other variables, such as the environment factors, influencing the effort.

3.2.1 Average Effort or Conversion Factor

This method uses historical average effort for doing activities (usually mean effort) as the basis for new estimates. For instance, we can estimate the time for executing tests based on the test productivity in previous projects using the following equation:

Table 3.1 General classification of existing estimation models.

Classification	Description	Example of Models
Productivity-Based	Simple models that are based only on the historical relation between output and input (e.g., SLOCs written per hour)	Average effort Conversion factor
Parametric	Models with mathematical algorithms or parametric equations. The parameters are usually related to the project under estimation.	COCOMO 81 SLIM
Statistical	Models created by using statistical techniques.	Regression models
Probabilistic	Models created by using probabilistic techniques.	Bayesian networks
Learning-Oriented	Models create by using machine learning techniques	Decision tree learning Case-Based Reasoning Rule Induction Neural Networks SVM
Expert-Based	Uses expert judgement for making estimates.	Delphi Wideband Delphi Planning Game Planning Poker WBS
Combined Techniques	Models created by a combination of techniques.	COCOMO II MP5 algorithm

$$Effort = N_{current} * \frac{ExecutionTime_{previous} N_{previous}}{}$$

where:

– $Effort$ is the estimated effort to execute the tests of the current project.

– $N_{current}$ is the number of tests of the current project.

– $N_{previous}$ is the number of tests of the previous project.

– $ExecutionTime_{previous}$ is the effort spent on executing tests of the previous project.

In this case, the effort is estimated by regarding the mean effort to execute a test. The generated estimates can be accurate only if productivity is almost constant.

This is also the approach used by Test Point Analysis (TPA) [105] and some other function points-based models. A conversion factor is calculated based on the inverse of productivity and it is used according to the following equations:

$$Effort = Points * CF$$

$$CF = \frac{Effort_{previous}}{Points_{previous}}$$

where:

- $Effort$ is the estimated effort to run the project.
- $Points$ is the number of points calculated for the current project.
- $Points_{previous}$ is the number of points calculated for the previous project.
- $Effort_{previous}$ is the effort spent on running the previous project.

3.3 Statistical Models

3.3.1 Regression Models

Regression analysis [53] is a statistical technique that investigates and models the relationship between dependent and independent variables. An independent variable is an input to an estimation model and it is also known as a predictor variable. In its turn, a dependent variable is an output of an estimation model. It has this name because its value depends on the value of the predictors variables. Dependent variables are also known as response variables.

The relationship between dependent and independent variables is defined as mathematical model called regression equation. The most simple type of regression is the simple linear regression. The general form of the mathematical model in this case is:

$$y_i = \alpha + \beta x_i + \varepsilon_i$$

In this linear model, y_i is the actual response when x_i occurs, α is the intercept and β is the slope of the linear equation, and ε_i is the error term (called residual), which represents the unpredictable part of the response variable y_i. Figure 3.1 illustrates a data set and the resulting regression model.

In the practice, it is common to use multiple linear regression, which is the generalization of simple linear regression to include more than one independent variable. In this case, we have the following equation for n independent variables:

$$y_i = \alpha + \beta_1 x_1 + \ldots + \beta_n x_n + \varepsilon_i$$

The response variable is modeled as a random variable because of the uncertainty of its value (we usually do not know the value of ε_i in advance). The regression equation estimates

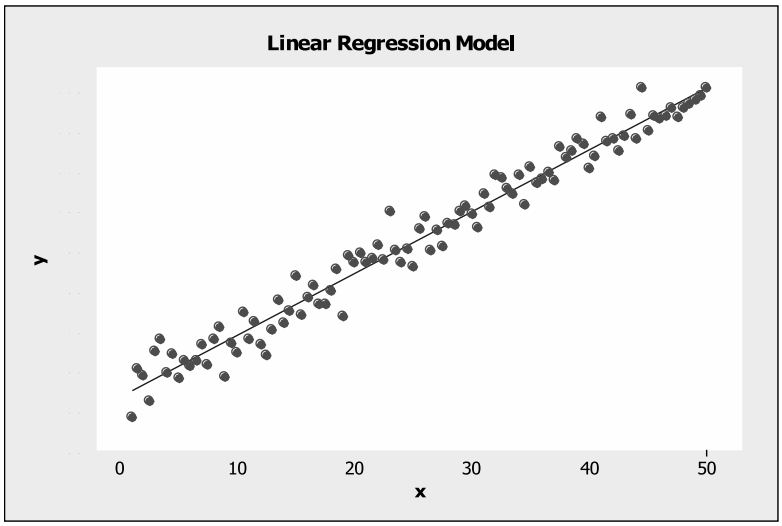


Figure 3.1 The linear regression on a data set.

hypothesized regression parameters that are constants (α and β_1 to β_n). These estimates are constructed using sample data for the variables. The estimates of these coefficients measure the relationship between the response variable and each independent variable.

There are several ways to perform regression analysis. With respect to the coefficient estimates, the Least Squares is the most common form of regression analysis. This technique choose the line that minimizes the sum of the squared errors. Least-Median-Squares is an alternative technique that determines outlier values prior to the final regression, enabling the analyst to discard or weight appropriately any outlier observations [84].

We also can use automatic techniques for variable selection, such as Stepwise Regression, which allows the computer to experiment and evaluate different combinations of independent variables. There are different approaches for selecting the independent variables when using stepwise regression, such as:

- Forward selection: starts with no variables in the model, then starts to include the variables, one by one, regarding only the ones that appears to be statistically significant.
- Backward selection: starts with all variables in the model, then starts to discard the variables that do not appear to be statistically significant.
- Forward and backward selection, where variables can be included or discarded in the model at any time.

Another question about regression analysis is that it is appropriated for using only with independent continuous variables. However, we can find a lot of relevant categorical variables in the practice. In this cases, we can make use of the dummy variables [85]. Basically, we create *n - 1* dummy variables, where *n* is the number of possible values of the categorical variable. We leave one possible value out of this process to avoid multicollinearity. These variables takes

values of zero or one, indicating the absence or presence of a given situation. For instance, if we have the categorical variable Q representing the quality of a product with possible values *Low*, *Average* and $High$, then we can create two dummy variables, such as Q_{Low} and Q_{High}. In this case, if the variable Q has the value *Low*, then Q_{Low} will have value 1 and Q_{High} will be set to 0. To represent the value *Average*, we set both variables to 0.

All we have discussed until now is considering a linear relationship between dependent and independent variables. However, this is not be true in some situations. For instance, development effort may not have a linear relationship with software size. In these cases, we can use a non-linear regression model [121] or we can transform the data. Concerning effort estimation, we usually opt for transforming the data. Since we usually have lot of data, the transformation techniques works fine. Also, non-linear regression models are harder to apply.

The transformations most used when regarding estimation models are the Box-Cox transformation. This technique tries to impose linearity and the data is expected to approximate to a normal distribution after the transformation. The Box-Cox transformation [124] has the following function:

$$\tau(Y;\lambda) = \begin{cases} \frac{Y^{\lambda}-1}{\lambda} & \text{if } \lambda \neq 0 \\ ln(Y) & \text{if } \lambda = 0 \end{cases}$$

where λ is a parameter calculated by statistical packages based on the analysis of the data to be transformed. As we can see, the logarithmic transformation is the specialization of the Box-Cox transformation when λ is 0. Actually, there is a more general form of this transformation:

$$\tau(Y;\lambda;\alpha) = \begin{cases} \frac{(Y+\alpha)^{\lambda}-1}{\lambda} & \text{if } \lambda \neq 0 \\ ln(Y+\alpha) & \text{if } \lambda = 0 \end{cases}$$

This general form includes a shift parameter α. This alternative function should be used when Y can assume the value 0 or a negative value, since the logarithmic function only accept positive values.

Finally, regression analysis is based on several important assumptions that should be checked to validate the model:

- The correct equation is being used (proper variables and functional form).
- Variables are measured accurately.
- Independent variables are independent from each other.
- The analyzed data is a random sample.
- The residual error term follows a normal or an approximate normal distribution.

If any of these assumptions are violated, the resulting regression equation may not be valid. Also, the Least Square regression model assumes that:

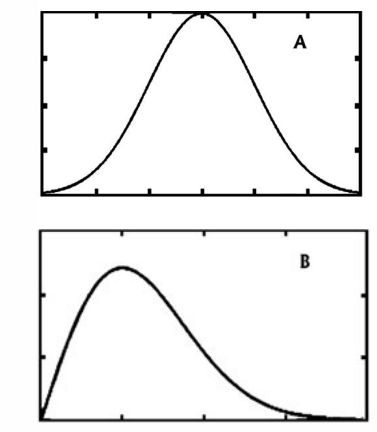


Figure 3.2 The normal (A) and the Rayleigh (B) distributions.

- The mean of the residuals is zero.
- The error term associated with one observation is not correlated with the error terms of the other observations (no autocorrelation).
- The variability of the error term is not related to the response variable (homoscedastic errors).
- The error term is not correlated with the response variable.

As presented by Jorgensen and Shepperd in [67], regression analysis is a tradicional method for creating estimation models in software engineering. Regression analysis can be used for estimating software effort, such as presented by Costagliola et al. [42], and also for estimating lines of code in early stages of software development [134].

3.4 Parametric Models

3.4.1 Putnam's Software LIfe-cycle Model (SLIM)

In the late 1970s, Larry Putnam of Quantitative Software Measurement [118] developed the Software Life-cycle Model (SLIM) [117]. SLIM describes the time and effort required to finish a software project of a specified size. Putnam noticed in his projects that software staffing profiles followed the well-known Rayleigh distribution (see Figure 3.2).

In the practice, the effort is calculated by the following equation:

$$Effort_{total} = \frac{Size}{Productivity}^{3} \frac{1}{Time^4}$$

where:

– $Effort_{total}$ is the estimated effort to run the project.

– *Size* is a sizing measure for the project.

– *Productivity* is the team productivity.

– *Time* is the schedule for complete the project.

This method is highly sensitive to uncertainty in both project size and team productivity estimates. Also, we need to define the schedule to estimate the project effort.

3.5 Probabilistic Models

3.5.1 Bayesian Networks

Bayesian Networks [123] are probabilistic graphical models that represents a set of variables and their probabilistic independencies. For example, a Bayesian network can represent the probabilistic relationships between effort and project characteristics. In [92], Mendes presents the use of this technique for estimating the effort to develop Web applications.

To use this approach, we have to develop the structure of the bayesian network. Nodes of the network and causal relationships should be identified. For instance, the node Effort can have relationship with nodes representing the type of the project and team experience. Then, the conditional probabilities should be estimated using expert judgement or based on historical data. For example, the probabilities of the Effort be Low, Average and High according to the possible combination of project types and team experience levels.

This approach requires the discretization of any continuous variable. It also do not produce a numerical estimate. Figure 3.3 shows an example of bayesian network based on another work of Mendes [90]. All nodes that affects the effort are measured qualitatively and the total effort is estimated as low, average, high and so on. Hence, the historical average of each category is used to estimate the effort in hours.

3.6 Machine Learning Approaches

3.6.1 Decision Tree Learning

In data mining and machine learning, a decision tree is a predictive model [100] [123]. Each internal node of the tree represents a conditional rule that guide to a leaf node. A decision tree can be a classification tree or a regression tree. In classification trees, each leaf node predicts a categorical attribute (classification) of the data that reach that node. In regression trees, the leaf nodes groups the data in a way that a specific numerical attribute (response variable) of the data is more homogeneous [119].

To construct a decision tree based on an example set, we first select an attribute to place at the root node and make one branch for each of its possible values. Each observation in the

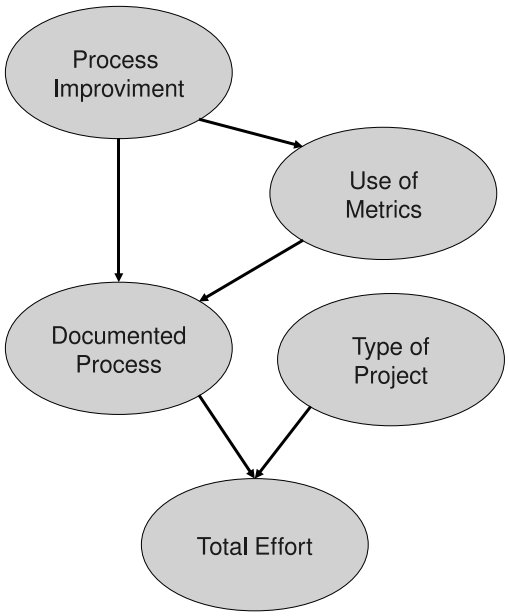


Figure 3.3 The linear regression on a data set.

example set is assigned to a unique child node. In this way, we split up the example set into subsets, one for each possible value of the attribute.

Then, this process is repeated recursively for each branch of the tree, using only those instances that actually reach the branch. The process stops developing a branch if all instances at a node have the same classification. At the end, each internal node of the tree will represent a splitting rule and each observation in the example set was assigned to a unique leaf node of the tree.

As we seek small trees, the determination of the attribute to be used in each step is based on a "purity" function calculated from the data assigned to a node (subset of the example set). We choose the attribute that produces child nodes with the purest data. The data is considered to be pure when it contains observations from only one class.

Some advantages of constructing decision trees are:

- They are simple to understand and interpret, increasing the chances to be validated and accepted by the user.

- Important insights can be generated, even when there are small data sets.

- Can be combined with other techniques. For instance, we can apply the decision tree to select relevant variables, to find more homogeneous data and to individually analyze them using other techniques.

3.6.2 Case-Based Reasoning (CBR)

Case-Based Reasoning (CBR) [139] is a technique to solve new problems by adapting the solutions of old problems. The CBR process finds the most similar case(s) in its database and tries to reuse it (them) to solve the new problem. In our context, a case is an abstraction of a software project. In this way, we can estimate effort of new projects based on historical information about completed projects with known effort.

To use CBR, we have to set parameters, such as the similarity function, which indicates how to measure the similarity between projects, the number of analogies to be used and the adaptation technique. For effort estimation, for example, we have to define how many similar projects we will use and how we will estimate the effort of the new project based on the past information (e.g., using mean effort, etc.). In [95], Mendes and Mosley investigate the use of CBR and Stepwise Regression to predict Web development effort.

3.6.3 Rule Induction (RI)

Rule induction (RI) is a particular form of inductive learning in which algorithms produce rules as a result of the analysis of a set of observations [100] [123]. By inductive learning we mean the process of acquiring general concepts from specific examples. The more common way to represent the output of this inductive learning is through IF-THEN rules. For instance, we can have the following rules for effort estimation:

IF FunctionPoints > 600 THEN Complexity = Average

IF FunctionPoints > 230 AND ProjectType = Critical THEN Complexity = Average

IF Complexity = Average THEN Effort = 2000-2600 Man-hours

As we can see, the rules can predict single value or a range of values representing the uncertainty.

3.6.4 Fuzzy Systems

The main assertion underlying the Fuzzy Logic approach [89] is that entities in the real world do not fit into simple categories. For instance, a project may not be either small, medium, or large, being in fact something in between. The idea is to assign membership values to the observations varying between 0 and 1. These extreme values respectively represent the absolute falsity and absolute truth.

For instance, we can assign a value 0.8 for the statement “the project is large”. This value is not the probability of project to be large. It just means that the project is almost large, because it also has some characteristics of medium projects.

This approach can be used to create rule induction, fuzzy-analogy and neuro-fuzzy systems to estimate effort [125] [76] [56]. Also, the fuzzy values can be used as independent variables in the models [50].

3.6.5 Artificial Neural Network (ANN)

The Artificial neural networks (ANNs) [51] [123] were inspired by the architecture of biological neural networks. Basically, an artificial neural network is a model composed of simple interconnected units called artificial neurons. Each neuron computes a weighted sum of its inputs and generates an output if the sum exceeds a certain threshold. This output then becomes an input to other neurons in the network or an output of the neural model.

To create such neural model, we have to define an appropriate layout of neurons. This process includes defining the number of layers of neurons, the number of neurons within each layer and how they are all linked. Then, the neural network need to be trained based on sample data. Back propagation is the most common learning algorithm that has been used by software metrics researchers.

Some limitations of this approach is the required expertise (not common in industry) to build the neural nets and its "black-box" model, which provide no explanation about its estimates.

3.6.6 Support Vector Machines

Support Vector Machines (SVMs) are a set of related supervised learning methods used in many areas, such as classification and regression [132] [36]. Instead of attempting to minimize only the empirical error, SVMs also maximize the geometric margin (maximum separation between two classes). In [45], the Support Vector Regression (SVR) was proposed for regression problems. There are few works in effort estimation using SVR, such as the study performed by Oliveira [111].

3.7 Expert-Based Approaches

3.7.1 Delphi

The Delphi method is a systematic and interactive method for obtaining consensus about a subject from a panel of independent experts [81]. The selected experts answer questionnaires in two or more rounds. After each round, a facilitator provides an anonymous summary of the experts' answers from the previous round, as well as the reasons they provided for their judgments. In this way, participants are able to review their previous answers now considering the answers of the other experts.

After each round, the range of the answers is expected to decrease and it is believed that the group will converge towards the "correct" answer. This process is stopped after satisfying some pre-defined stop criterion (number of rounds, achievement of consensus, stability of results, etc.).

Some of the advantages of this method are:

- The use of the experts knowledge.
- The facilitator avoids the negative effects of face-to-face panel discussions and solves usual problems of group dynamics.

- Prevention of participants to stick to previously stated opinions.
- Allows participants to freely express their opinions and to revise earlier judgments.
- Avoids participants dominating others using their authority or personality.
- Panel can be performed asynchronously. For instance, questionnaires and answers can be sent by e-mail.
- Tends to be more accurate than when using unstructured groups of experts.

Although it is a simple method, the Delphi panel can also fail. Some of the reasons for the failure of a Delphi panel are:

- Assuming that Delphi can be a surrogate for all other human communications in a given situation, not allowing for the contribution of other perspectives related to the problem.
- Imposing the coordinator's view and preconceptions of a problem upon the respondent group.
- Poor summarization and presentation of the group response and not ensuring the common interpretations of them.
- Generation of an artificial consensus, in which disagreements are ignored.
- Lack of experts or motivated people for the Delphi panel.
- Underestimating the demanding nature of a Delphi.

3.7.2 Wideband Delphi

Wideband Delphi estimation method is an adaptation of the Delphi technique explained in Section 3.7.1. It is a consensus-based estimation technique for estimating effort involving greater interaction and more communication between those participating. This method was popularized by Barry Boehm in [26] and the main steps to arrange and perform the expert's panel are:

1. A coordinator gives a specification and an estimation form for each expert.
2. The coordinator set up a group meeting. In this meeting, the experts discuss estimation issues with the coordinator and each other.
3. Experts fill out forms anonymously according to their judgment.
4. Coordinator collects responses and distributes a summary of the estimates.
5. Coordinator may call new group meetings focusing on points where their estimates vary widely. Steps 3 to 5 are then iterated for as many rounds as necessary to achieve a consensus.

3.7.3 Other Approaches

Planning Game [25] is the software project planning method from Extreme Programming (XP), a lightweight development methodology developed by Kent Beck in the 1990s. The Planning Game has a highly iterative process, in which the Development and Business work together to interactively write “user stories" on cards. We give estimates to each story in weeks. This process is repeated continuously throughout the project.

The *planning poker* [40] [115] works a similar way of the Wideband Delphi to achieve a consensus. However, it uses cards instead of questionnaires for given estimates. These cards follow an uncommon sequence, in which higher numbers have less granularity. This encourages the team to split large stories into smaller ones. There are also special cards, such as the coffee cup card, which means that you are too tired to think and need a short break, and the question mark, which means you have no idea about what is being discussed.

Another traditional way to estimate effort based on expert judgment is using a *WBS* [26]. WBS is a way of organizing project elements into a hierarchy, simplifying the cost estimation and control. A software WBS actually consists of two hierarchies. One represents the components of the software product and the other represents the activities needed to build that product.

3.8 Combined Techniques

3.8.1 COCOMO II

The first version of COCOMO (COnstructive COst MOdel) cost and schedule estimation model was first published in 1981 [26]. Then, the COCOMO II research effort was started in 1994 at USC to address several issues, such as:

- Non-sequential and rapid development process models;
- Reengineering;
- Reuse driven approaches;
- Object oriented approaches;
- Etc.

The COCOMO model has three types of inputs, the size of the system to be developed, the effort multipliers and the scale factors. Effort multipliers are cost drivers having linear relationship with effort. The scale factors having non-linear relationship with effort, determining what is called economies/diseconomies of scale of the software under development.

There are three submodels defined in COCOMO II [28]:

- Applications Composition Model: used for prototyping and evaluate potential high-risk issues, such as user interfaces, software/system interaction, performance or technology

maturity. In this model, object points are used for sizing rather than the LOC metric. This size measure is determined by counting the number of screens, reports, and other components that will be used in the application.

- Early Design: involves the exploration and evaluation of alternative system architectures and concepts of operation. Function points (or lines of code when available) are used as input of the model, as well a set of five scale factors and 7 effort multipliers.
- Post-Architecture: this model is used when you have completed the top level design of the system and detailed information about the project is already available. It uses SLOC and/or Function Points as the size input parameter. A set of 17 effort multipliers and a set of 5 scale factors.

The Early Design and the Post-Architecture models use the same approach for sizing products and for scale factors. Part of this process is summarized next.

The effort in persons-month is given by the following equations:

$$PM = A \times Size^{E} \times \prod_{i=1}^{n} EM_i$$

$$E = B + 0.01 \times \sum_{j=1}^{5} SF_j$$

where:

- PM is the effort estimate.
- $A = 2.94$ and $B = 0.91$.
- EM_i is the ith effort multiplier.
- SF_j is the jth scale factor.

If $E < 1$, the project exhibits economies of scale. If $E = 1$, the economies and diseconomies are in balance. If $E > 1$, the project exhibits diseconomies of scale.

The list of scale factors and effort multipliers considered by the COCOMO II model is shown in Table 3.2. The 5 scale factors are rated according the guidelines shown in Table 3.3 and the respective weights assigned to them are presented in Table 3.4. The evaluation of the effort multipliers are done in a similar way.

The Post-Architecture Model was calibrated to a database of 161 projects, which data was collected from Commercial, Aerospace, Government and non-profit organizations. This calibration process included the use of: data analysis using the log-log transformation and linear regression analysis; expert judgment (Delphi panel) to create a model based on expert opinion; a bayesian regression approach to combine the information based on the data and the experts.

Although some works supported the hypothesis of the existence of both economies and diseconomies of scale in software development [22], there are also some critiques about these economies/deseconomies of scale considered in the construction of COCOMO [74].

With respect to tool support, there are several commercial and free tools available on the Internet that implement this model.

Table 3.2 Cost drivers considered for COCOMO II.

Scale Factors	Precedentedness Development Flexibility Architecture/Risk Resolution Team Cohesion Process Maturity	
Effort Multipliers	Product Attributes	Required Reliability Database Size Product Complexity Required Reuse Documentation
	Platform Attributes	Execution Time Constraint Main Storage Constraint Platform Volatility
	Personnel Attributes	Analyst Capability Programmer Capability Personnel Continuity Applications Experience Platform Experience Language and Toolset Experience
	Project Attributes	Use of Software Tools Multisite Development Required Development Schedule

3.8.2 M5P

MP5 algorithm [138] can learn piecewise linear models, combining decision trees and linear regression techniques. In this algorithm, the decision tree is adapted to a regression problem. The M5P algorithm adapts decision trees to find more homogeneous data and individually analyze them using linear regression analysis.

3.9 Final Considerations

In the previous chapter we presented the most used software size metrics. In this chapter we classified and reviewed several estimation models. Some of these models were presented in more detail, mainly the ones that we believe that can be used for test execution effort estimation.

The presented estimation models use metrics related to code or requirements, but they do not consider metrics related to test artifacts, limiting their accuracy for estimating testing execution effort. In the next chapter, we present how we defined a new test size measure and how we can use it to estimate test execution effort.

Table 3.3 Scale Factors for COCOMO II Early Design and Post-Architecture Models [28].

Scale Factors	Very Low	Low	Nominal	High	Very High	Extra High
PREC	thoroughly unprecedented	largely unprecedented	somewhat unprecedented	generally familiar	largely familiar	throughly familiar
FLEX	rigorous	occasional relaxation	some relaxation	general conformity	some conformity	general goals
RESL	little (20%)	some (40%)	often (60%)	generally (75%)	mostly (90%)	full (100%)
TEAM	very difficult interactions	some difficult interactions	basically cooperative interactions	largely cooperative	highly cooperative	seamless interactions
PMAT	SW-CMM Level 1 Lower	SW-CMM Level 1 Upper	SW-CMM Level 2	SW-CMM Level 3	SW-CMM Level 4	SW-CMM Level 5

Table 3.4 Weights of Scale Factors for COCOMO II Early Design and Post-Architecture Models [28].

Scale Factors	Very Low	Low	Nominal	High	Very High	Extra High
PREC	6.20	4.96	3.72	2.48	1.24	0.00
FLEX	5.07	4.05	3.04	2.03	1.01	0.00
RESL	7.07	5.65	4.24	2,83	1.41	0.00
TEAM	5.48	4.38	3.29	2.19	1.10	0.00
PMAT	7.80	6.24	4.68	3.12	1.56	0.00

CHAPTER 4

Test Execution Effort Estimation

In previous chapters, we overviewed several existing software measures, measurement methods and effort estimation techniques. As we could observe, there are several measures and estimation models for software development, but none of them are apropriate for estimating test execution effort. In this chapter, we present the planning of this research and the development of a measure for test size and execution complexity. We also show how to estimate test execution effort based on the proposed measure.

4.1 Research Plan

The planning of this research is structured in terms of research goals, questions, and hypotheses, as described next.

4.1.1 Goals

The main goal of this research is the development and evaluation of a measure for test size and execution complexity that is based on test specifications. Also, we want to evaluate its adequacy for estimating manual test execution effort. In summary, we aim to create an estimation approach for using in the mobile application domain with the following characteristics:

- Accurate estimates;
- Based on the size and execution complexity of test specifications, enabling the estimation of manual test execution effort for any given set or subset of existing functional tests;
- Based on cost drivers related to manual test execution (team experience, test environment, stability of the tested product, etc.);
- Good estimation accuracy for existing tests and for new ones;
- Low estimation cost.

We believe that these characteristics should improve the accuracy of effort estimation for manual test execution. To achieve our main goal, we have the following secondary goals:

- Define and evaluate an automated method for sizing test specifications;

- Identify cost drivers related to manual test execution and investigate their impact on test execution effort;
- Define, calibrate, and evaluate our estimation approach through empirical studies on the mobile application domain.

4.1.2 Research Questions

The assessment of our research goals will be performed by answering the following research questions on the mobile application domain:

RQ1: Can we measure the size and execution complexity of a test specification?

RQ2: Is the estimation accuracy of manual test execution effort improved by regarding the size and execution complexity of test specifications?

RQ3: Is the estimation accuracy of manual test execution effort improved by regarding cost drivers related to manual test execution?

RQ4: What is the cost for estimating manual test execution effort based on the size and execution complexity of test specifications, and based on cost drivers related to manual test execution?

Questions RQ1 to RQ3 were not answered in previous works, since no existing effort estimation approach regards the size of test specifications and cost drivers related to manual test execution. Also, the cost of calibrating and using an estimation model may be prohibitive. The research question RQ4 is related to this issue.

In addition, we derived question RQ1 into more concrete ones, making easier to answer them during our empirical studies on the mobile application domain:

RQ1.1: What are the relevant characteristics that should be considered when sizing a test specification?

RQ1.2: What is the weight of each relevant characteristic with respect to manual test execution effort?

RQ1.3: Can we define a soundness measure for test size and execution complexity?

RQ1.4: Can we automate the measurement of test size and execution complexity?

RQ1.5: What are the relevant cost drivers for estimating manual test execution effort?

RQ1.6: What is the impact (weight) of each relevant cost driver on manual test execution effort?

4.1.3 Research Hypotheses

Based on our research questions and our findings from the review of literature, we formulate our research hypotheses as presented next:

RH1: It is possible to determine the size and execution complexity of a test specification by observing characteristics of each test action found in the specification.

RH2: The use of a measure for test size and execution complexity will significantly improve the estimation accuracy of manual test execution effort.

RH3: The use of cost drivers related to manual test execution will significantly improve the estimation accuracy of manual test execution effort.

RH4: It is possible to automate the measurement of test size and execution complexity, reducing the costs of our approach.

These research hypotheses are our assumptions that should be supported or rejected at the end of this work.

4.1.4 Main Activities

To answer our research hypotheses and to achieve our research goals, we structured this work in the following main research activities:

1. *Analyze the problem.*
    - Analyze test specifications and expert opinion to identify characteristics in test specifications that can be used to determine test size and execution complexity.
2. *Propose a measure for test size and execution complexity.*
    - Identify how relevant characteristics for the size and complexity of test specifications can be measured. That is, define a measure and a measurement method for test size and execution complexity based on test specifications.
    - Identify how to verify if the proposed measure is sound.
    - Define how to configure the measurement method for different application domains, which includes the identification of relevant characteristics and their weights, either based on:
        - Expert opinion, to create an initial configuration, which is specially useful when there is no historical data available.
        - Analysis of historical data, which can result in more precise results for analyzing the relevance and weights of each characteristic.
    - Identify how the proposed measure can be used to create an estimation model for test execution effort.
3. *Evaluate the proposed measure and measurement method.*
    - Evaluate if the proposed metric is sound and that its measurement method is feasible (not too costly, etc.).

- Evaluate the adequacy of the proposed measure for estimating test execution effort by observing the resulting estimation accuracy.

4. *Identify and evaluate cost drivers related to manual test execution.*

   - Based on expert opinion, identify possible cost drivers for executing tests manually.
   - Determine the relevance of some of the identified cost drivers, as well as their effect (weight) on test execution effort.
   - Evaluate the resulting estimation accuracy when considering the proposed measure and relevant cost drivers.

5. *Develop supporting tools.*

The remaining of this chapter and the next one present the results of these activities.

4.2 Test Size and Execution Complexity

In this section, we present our proposed measure for test size and execution complexity [12][11], as well as its measurement method based on test specifications. We start presenting how test specifications are usually written in industry and what is necessary to enable better interpretation and automatic processing. After that, we present our proposed measure and then its measurement method.

4.2.1 Test Specification Language

Tests are usually specified in terms of pre-conditions, procedure (steps, inputs and expected outputs) and post-conditions [68]. These specifications are commonly written in natural language, often leading to problems such as ambiguity, redundancy and lack of writing standard (level of detail, structure, etc.). All these problems make difficult to understand and estimate test size and execution complexity. Nevertheless, they can be avoided using controlled natural languages.

A controlled natural language (CNL) [126] 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. In a simplified way, each sentence (test step) of a test specification conforms to the following structure: a main verb and zero or more arguments. Table 4.1 shows an example of test procedure written in a controlled natural language that tests a feature from the mobile application domain.

For each sentence, the verb identifies the test action to be performed during the test. The verb arguments provide additional information for the test action represented by the verb. For instance, the sentence *Start the message center* has the verb *start* (action of starting an application) and the argument *the message center* (application to be started). A CNL usually have its lexicon and grammar extended for specific application domains. For example, the list of

Table 4.1 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.

possible verbs and arguments can be different between the mobile and the Web application domains. Since the context of this work is related to the test of mobile applications, the considered controlled natural language reflects this domain [135] [78].

There are several advantages in using CNL, such as the support to write unambiguous, complete and consistent specifications [126]. However, its adoption may not be easy due to the lack of tools used to check the correctness of the specifications, the costs and skills required to define the controlled language, etc. Also, a testing organization can already have several existing test cases written in natural language. In these cases, our proposed test size and complexity measurement method should be applied to specifications written in natural language (NL) with sentence structures similar to the CNL described here. From now on, we call this structured NL as a standardized natural language.

In this work, we consider that tests are written in a CNL or a standardized natural language. The details of our proposed measure and its measurement method are presented next.

4.2.2 Execution Points (EP)

Test teams may not have enough resources or time to execute all tests of an application when features are being developed or changed. For this reason, only a subset of existing tests is selected for execution. To enable the effort estimation for executing any given set of tests, we need a method for sizing tests based on their specifications.

In this work, we propose a measure for the size of test specifications that are weighted by their execution complexities, an approach used in other existing measures [29] that should increase the correlation between the proposed measure and manual test execution effort. By test size, we mean the amount of steps required to execute the test. We consider test execution complexity as the difficulty of interaction between the tester, the tested product, and the test environment that is required during the test. These definitions are adaptations of the idea of size and development complexity for software products [112] [32] [48].

The proposed measure, called Execution Points (EP), should reflect the amount of work need to manually execute tests. Basically, this measure is based on the amount of test actions (user actions and observed results) found in a test specification, considering the functional (data

input, screen navigations, etc.) and non-functional (use of network, etc.) characteristics of the applications and system environments exercised by the test actions.

Execution points should have a high linear correlation with the effort to manually execute tests. For this reason, the number of execution points should be useful not only for estimating test execution effort, but also for comparing and prioritizing tests. For example, considering the same environment conditions, a test with 1000 execution points should require approximately twice the effort needed to execute a test with 500 execution points. In addition, execution points allow us to better compare testers's productivity and test team capacity. For instance, a tester that executed 5 tests with 500 execution points each one can be considered faster than another that executed 15 tests with 100 execution points during the same amount of time.

4.2.2.1 Measure Validation

Since we are proposing a measure for size and execution complexity of a test specification, it is important to have an intuitive understanding of this test specification attribute. This leads us to the identification of empirical relations between tests with respect to their size and execution complexity:

- The relation *bigger than* indicates that one test has a bigger size and execution complexity than another.
- The relation *similar to* indicates that one test has a similar size and execution complexity when compared to another.

These relations were defined intuitively by analyzing how experts create associations between test specifications with respect to their size and execution complexity. Also, we consider that a test t_1 is *bigger than* a test t_2 only if t_1 is not *similar to* t_2. This assumption reflects the difficulty to intuitively compare similar tests with respect to their size and execution complexity.

Let us call T as the set of all existing test specifications. The set containing the identified empirical relations (*bigger than* and *similar to*) is called R. Then, we call (T, R) as the empirical relation system for the attribute size and execution complexity of a test specification [48]. To measure test size and execution complexity that is characterized by (T, R), we must define a mapping M of (T, R) into (E, P), in which test cases in T are mapped into numbers (called execution points) in E and empirical relations in R are mapped into numerical relations in P. In this way, we can evaluate our measure by demonstrating empirically that the mapping M is valid for the attribute size and execution complexity.

The set E of all possible numbers of execution points consists of nonnegative integers and the set of numerical relations P consists of the relations $>_{ep}$ and $\approx_{ep}$ defined as follows.

$$a >_{ep} b = \begin{cases} false & \text{if } a \approx_{ep} b \\ a > b & \text{otherwise} \end{cases} \tag{4.1}$$

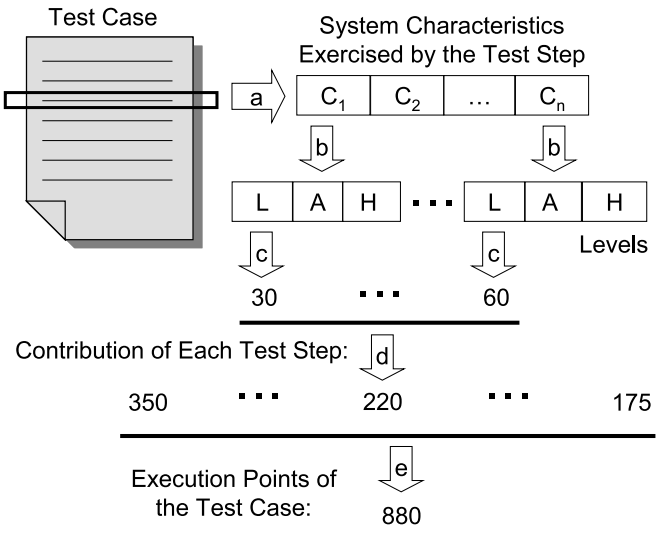


Figure 4.1 Assigning execution points to a test case.

$$a \approx_{ep} b = \begin{cases} true & \text{if } \frac{|a-b|}{a} \leq \frac{p}{100} \text{ and } \frac{|a-b|}{b} \leq \frac{p}{100} \\ false & \text{otherwise} \end{cases} \tag{4.2}$$

As we can see in Equation 4.1, the expression $a >_{ep} b$ is equivalent to the expression $a > b$, except in the case of similar numbers of execution points ($\approx_{ep}$). The definition of $\approx_{ep}$ in Equation 4.2 shows that numbers a and b are considered similar if the absolute value of $a - b$ is less than or equal to p percent of a and of b. The value of p is discovered empirically, as discussed later in Section 5.3.3. The relations of R and of P are mapped following the order of their presentations in this section.

During the empirical studies presented in Chapter 5, we also evaluate the practical use of execution points. For instance, we evaluate the estimation accuracy and cost achieved when using execution points in comparison to other measures.

4.2.3 Execution Points Measurement Method

Since all information required to compute execution points is extracted from test specifications, we only consider test specifications written in CNL or in a standardized NL, as discussed in Section 4.2.1. This restriction simplifies the use of our approach, improves its accuracy and also supports a high level of automation. Fig. 4.1 illustrates how we measure execution points. First, (a) we individually analyze each test step (sentence) of the test specification. This step by step analysis was defined with the objective to support the measurement method automation. We analyze each test step according to a list of characteristics (C_1 to C_n).

The characteristics C_1 to C_n 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. Each character-

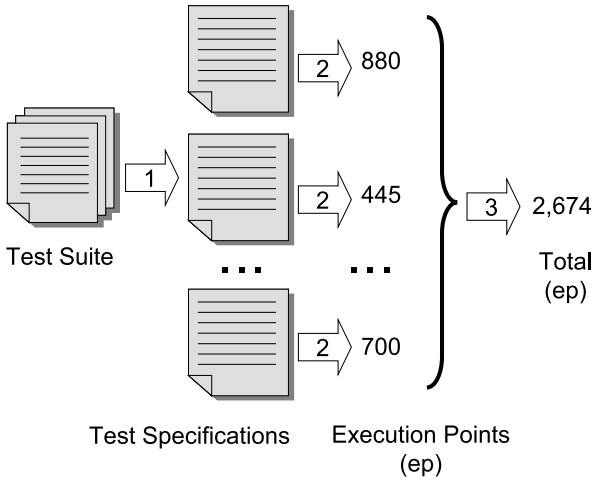


Figure 4.2 Measuring the number of execution points of a test suite.

istic considered by the method has an impact on the test size and execution complexity and, consequently, on the effort required to execute the test. (b) This impact is rated using an ordinal scale (Low, Average and High). As described later in Section 4.2.3.1, a Delphi approach can be used to choose the set of relevant characteristics and to define guidelines to help the selection of the more appropriate impact level for each characteristic.

After that, (c) we assign execution points for each characteristic according to its impact level, transforming the qualitative rate (impact level) into a quantitative value. For instance, a characteristic C_1 rated with the Low value can be assigned to 30 execution points. However, a more relevant characteristic rated with the Low value may be assigned to a higher number of execution points. Section 4.2.3.1 also discusses about guidelines 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 by summing the execution points of each one of its test steps. In practice, we need to measure the size and execution complexity of test suites (set of related tests). For that, we measure the execution points of each test specifications. Then, the sum of these points represents the size and execution complexity of the whole test suite.

In the practice, we have to measure the size and execution complexity of test suites (set of related test cases). Figure 4.2 illustrates this process. First, we measure the execution points of each test specifications. Then, the sum of these points represents the size and execution complexity of the whole test suite.

4.2.3.1 Configuration

The measurement method of execution points uses a model that should be configured according to the target application domain in order to maximize the estimation accuracy. This section presents what, why and how to configure our estimation model.

Controlled or Standardized Natural Language

Test specifications are the input of our approach and they can be written in a CNL or a standardized natural language. In the case of using CNL, as shown in Section 4.2.1, its grammar and lexicon are defined according to the target application domain. For example, we have the verb *take* on the mobile application domain that accepts the term *picture* as argument. Hence, a possible test step is *Take a picture*.

The list of verbs and possible arguments can be constructed by analysing requirement documents and existing test specifications. Besides, new verbs and terms can arises over the time due to the specification of new requirements, changes in technology, etc. The CNL grammar and lexicon can be stored in a database to be updated whenever necessary.

System Characteristics

During the test execution effort estimation, all test steps are analyzed according to a list of characteristics. These characteristics represent some general functional and non-functional requirements exercised when a test action is executed. The list of characteristics to be considered may depend on the target application domain. For instance, the pressing of phone keys and the usage of mobile network are examples of characteristics exercised on mobile applications under test. However, these characteristics are not valid for Web and other different application domains.

We use the Delphi method [81] for obtaining a consensus from a group of experts about the list of relevant characteristics to consider. The Delphi panel consists of 3 to 7 experienced testers invited from different teams for attending two or more rounds. In each round, they have the opportunity to add or remove characteristics from the list.

Examples of real test specifications are provided as a source for identifying different types of test actions, software configurations, use of tools or of specific hardware, and other characteristics that can impact the size and execution complexity of a test. All this process is anonymous and, in each round, a moderator provides the participants with a summary of the experts' decisions and their reasons for that.

An alternative technique that can be used to identify relevant characteristics is the survey [114]. When we have several testers in the organization, we can survey them about the relevant characteristics using questionnaires or other survey instruments.

Guidelines

Once the experts have defined the list of characteristics to be considered by the estimation model, the experts continue attending the Delphi panels, but with different objectives. First, they have to define the possible values for each identified characteristic. For example, if the type of camera is selected as a relevant characteristic, its possible values would be automatic shooting, manual zoom, use of flash, etc.

After identifying the possible values of each relevant characteristic, the experts group these values into three impact levels (low, average and high). The choices are made based on the impact of each value in the test size and execution complexity. This part of the guideline will

help us to objectively choose the more appropriate impact level of a test step according to each characteristic.

Finally, the experts must define for each characteristic the number of execution points to be assigned for each one of its impact levels. The experts can proceed as follows. Each characteristic is weighted from 1 to 10. These weights indicate the significance of each characteristic for the test size and execution complexity. Then, the experts give a weight from 1 to 10 for the levels Low, Average and High of each characteristic. These weights indicate the significance of each level for the characteristic. Then, the number of execution points assigned for a level is calculated by multiplying its weight by the weight of its characteristic.

An alternative procedure is to estimate the increase of time caused by each characteristic level with respect to manual test execution effort. Then, the estimated increase of time is used as the weight of each characteristic level.

The list of relevant characteristics, their values and weights can be updated periodically by one or two experts.

4.2.3.2 Measurement Method Automation

One objective of this work is the development of an estimation approach that can be automated. This automation is important for supporting the development of new test generation and test selection tools. In practice, companies may not be able to execute all tests generated by such tools, since their resources are limited. For this reason, test execution effort should be taken in consideration for test selection. A test generation tool, for instance, can consider a minimal requirement coverage and a maximum execution effort as its stop criteria.

The use of CNL or a standardized natural language for specifying tests supports the development of an estimation tool that automatically reads and interprets these specifications. In addition, all the information required for measuring execution points, such as the list of characteristics, guidelines and weights can be stored in a database. Actually, the CNL grammar and lexicon can also be stored in the database [135].

During the analysis of the first test cases, the estimation tool asks the user to rate the characteristics of each different test action (verb). This information is stored in the database. Since number of possible ways (verbs) to describe a test action is reduced (see Section 4.2.1), it is reasonable that the same test action occurs many times in the same test specification and in different ones. For this reason, the need for manual assistance during the estimations tends to be reduced as much as we process tests.

4.2.3.3 Automatic Calibration of Characteristic Weights

Although the experienced testers can calibrate the execution points measurement method (define the weight of each characteristic level), this procedure can be automated when we have historical test execution data stored with a high level of detail. Basically, we need to have the time spent to execute each test action (sentence) of each test specification. This detailed information can be collected by using ManualTEST [17] (see Appendix D), a tool that we developed for collecting test execution data with a high level of detail and accuracy.

The time spent to execute each sentence is also the time spent to execute each test action

(verb or verb + arguments). In Table 4.2, we are considering the verb as the identifier of the test action. For some test actions with larger execution time variation, such as Select, we can also consider the identifier of the test action as the verb + arguments, increasing the accuracy of this calibration process.

Table 4.2 Storing the time to execute each sentence (test action) of test specification TS-1.

Step	Sentence	Test action (verb)	Time
1	Go to Message Center	Go	00:00:10
2	Select Email Msgs	Select	00:00:05
3	Accept the dialog	Accept	00:00:21
4	Exit notification	Exit	00:00:14
5	Exit Account Folder	Exit	00:00:09
6	Go to Account Folder	Go	00:00:06

The next step is to join information about execution times and characteristics rates for each test action (see Table 4.3, which consider the use of three characteristics). Each test action is usually associated to different execution times, since we have an inherent variability, different verb arguments, different levels of tester experience and effects related to the testing environment. For this reason, it is important to analyze data collected in controlled test execution environments and from the same tester or from testers with similar expertise.

To analyze the data presented in Table 4.3, we use the following regression model:

$$Time_i = \beta_0 + \beta_1 ScreensLow_i + \beta_2 ScreensAvg_i + \beta_3 ScreensHigh_i + \beta_4 KeysLow_i \quad (4.3)$$

$$+\beta_5 KeysAvg_i + \beta_6 KeysHigh_i + \beta_7 DelayLow_i + \beta_8 DelayAvg_i + \beta_9 DelayHigh_i + \varepsilon_i$$

where:

- $Time_i$ is the time spent to execute the ith test action.
- $ScreensLow_i$, $ScreensAvg_i$ and $ScreensHigh_i$ are dummy variables for characteristic Screens with respect to the ith test action.
- $KeysLow_i$, $KeysAvg_i$ and $KeysHigh_i$ are dummy variables for characteristic Keys with respect to the ith test action.
- $DelayLow_i$, $DelayAvg_i$ and $DelayHigh_i$ are dummy variables for characteristic Delay with respect to the ith test action.
- β_0 to β_9 are coefficients defined during the regression analysis.
- ε_i is the error term for the ith test action.

Table 4.3 Joining information about execution time and characteristics levels for each test action.

Test/Step	Test Action	Time	Characteristics		
			Screens	Keys	Delay
TS-1/3	Accept	00:00:21	N/A	Low	Low
TS-3/3	Accept	00:00:23	N/A	Low	Low
...	...	...	...	...	...
TS-1/4	Exit	00:00:14	Low	Low	N/A
TS-1/5	Exit	00:00:09	Low	Low	N/A
TS-2/5	Exit	00:00:08	Low	Low	N/A
TS-3/4	Exit	00:00:04	Low	Low	N/A
...	...	...	...	...	...
TS-1/1	Go	00:00:10	Low	Low	N/A
TS-1/6	Go	00:00:06	Low	Low	N/A
TS-2/1	Go	00:00:08	Low	Low	N/A
TS-3/1	Go	00:00:07	Low	Low	N/A
...	...	...	...	...	...
TS-2/3	Reject	00:00:05	N/A	Low	N/A
...	...	...	...	...	...
TS-1/2	Select	00:00:05	N/A	Low	N/A
TS-2/2	Select	00:00:04	N/A	Low	N/A
TS-2/4	Select	00:00:34	N/A	Low	N/A
TS-3/2	Select	00:00:04	N/A	Low	N/A
TS-3/5	Select	00:00:22	N/A	Low	N/A
TS-3/6	Select	00:00:14	N/A	Low	N/A
...	...	...	...	...	...
...	...	...	...	...	...

As we can see, we create dummy variables for each characteristic. To avoid problems of multicolinearity [85], we create only $n-1$ dummy variables for each characteristic, where n is the number of possible values for each characteristic (e.g., N/A, Low, Avg and High). As we can see in Equation 4.3, we did not create a dummy variable for the value N/A (Not Applicable), which is a reference (weight=0) with respect to the levels Low, Avg and High. Finally, the weights for the levels Low, Avg and High of each characteristic are the values of their respective β coefficients calculated by the regression analysis.

In summary, the weight of each characteristic level is the increase of time caused by it. The correct identification of relevant characteristics should result in positive values for all β coefficients. In this way, this calibration procedure can require changes in the set of characteristics considered relevant if problems during the regression analysis arises.

4.2.3.4 Measurement Validity

The measurement method of Execution Points (EP) is similar to Function Points Analysis (FPA). Hence, we have to observe if the critiques to FPA (see Section 2.3.1) are applicable to the measurement of execution points. Most of the critiques are related to problems in the model construction: issues in the expert judgments and measurement scales and transformations throughout the measurement steps.

In FPA, absolute scale counts are reduced to ordinal scale measures. For instance, counts of Data Element Types (DET) and Record Element Types (RET) are reduced to the Low, Average and High scale. This scale transformation can be seen as a loss of information. When measuring EP, we do not have absolute counts, but only intervals. For instance, we do not know how many screen navigations the test action "go" will require, but we do know which interval (Low, Avg and High) has the higher probability to include this number. In this way, there is no loss of information in the EP measurement method.

Another problem with FPA is that ordinal values, such as Low and Average, are transformed to a ratio scale in order to enable their sum, multiplication, etc. For instance, what is the meaning of summing two ordinal values, such as Low + High? We have similar transformations in the measurement of EP. However, the transformation of the Low level of a characteristic to the weight 25 means that, in average, the value Low of that characteristic increases the test execution time in 25 units. This interpretation is more clear when we use the automatic calibration procedure presented in Section 4.2.3.3.

Another reported problem with function points is the technology adjustment factor, that is based on a subjective assessment of 14 project factors on a six-point ordinal scale. In our approach, the measurement method should be configured for each different application domain, having no standard configuration that can be used in any testing project.

In addition, some works found correlations between Albrecht function point elements, the first version of FPA. Also, some researchers observed that some function point elements were not related to effort in their studies [73]. These kind of problems can occur by using the EP measurement method calibrated by expert judgement. However, the statistical tests performed during the automatic calibration procedure (regression analysis) of the EP measurement method can identify and guide the solution of these problems.

4.3 Estimation of Test Execution Effort

The execution points counting of a test suite gives us a reference about its tests size and execution complexity. Based on the literature and in our experience, we believe that this is the most important information for estimating test execution effort. As presented next, we can use execution points to estimate manual test execution effort in different ways. Some of them also take in consideration the impact of cost drivers.

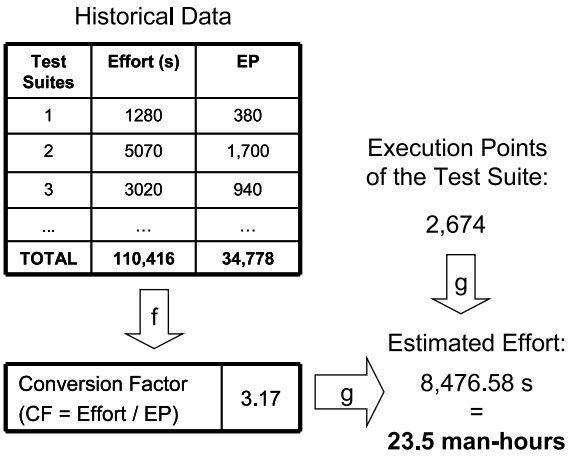


Figure 4.3 Using execution points and a conversion factor to calculate test execution effort.

4.3.1 Test Productivity-Based Approach

The estimated effort can be calculated based on the execution point counting and a conversion factor (CF). This conversion factor represents the relation between test execution effort and execution points, which varies according to the productivity of the test team.

The conversion factor is given in seconds per execution point, indicating the number of seconds required to execute each execution point of a test. For calculating the conversion factor, testers can measure the execution points of some tests and collect their execution effort based on a historical database (when available) or by executing them. As illustrated by Figure 4.3, (f) the conversion factor is calculated by dividing the total effort by the total number of execution points. For estimating the execution effort of new test suites, we just (g) multiply their number of execution points by the calculated conversion factor.

In the example of Figure 4.3, the tester verified a conversion factor of 3.17 seconds per execution point. Using this value, a new test with 2,674 execution points is estimated to be executed in 23.5 hours. Similar approach is used by other existing estimation models [49] [101] [105] [112].

In this approach, we assume that test productivity and environment conditions are stable over time. For example, improvements in the test team, tools or environment may change the test productivity and consequently the conversion factor. In this case, the conversion factor should be recalculated using data collected after the improvements.

In summary, the conversion factor used in the estimations should properly represent the current situation. Additionally, a different conversion factor can be calculated and used for each different test team, since they can have significant different productivities due to the type of executed test, the tested product, etc.

4.3.2 COCOMO-Based Approach

COCOMO is a well-known effort estimation model that has several extensions for different purposes, such as the COQUALMO [38] and COSYSMO [30]. We verified that the COCOMO model can be used for estimating manual test execution effort. For that, we have to identify the cost drivers related to test execution effort to be included in the COCOMO general equation:

$$PM = A \times Size^{E} \times \prod_{i=1}^{n} EM_i \tag{4.4}$$

$$E = B + C \times \sum_{j=1}^{m} SF_j \tag{4.5}$$

where:

- PM is the effort applied in person-months. In our case, the effort to execute the tests.
- $Size$ is a measure of the project size, that is, the number of execution points of the test cases.
- EM_1 to EM_n are the effort multipliers, that is, cost drivers having linear relationship with effort.
- SF_1 to SF_m are the scale factors, that is, cost drivers having scale impact on effort.
- A, B and C are coefficients defined during a model calibration procedure.

As we can see, the cost drivers must be classified in effort multipliers and scale drivers. In our case, the effort multipliers should have linear relationship with test execution effort and the scale drivers should have an exponential impact on effort. The coefficient of each cost driver must also be defined using expert judgment, regression analysis or a combination of both methods.

4.3.3 Regression-Based Approach

In this approach, we use linear regression analysis [53] to build an estimation model, which is a mathematical equation relating independent (predictors) and dependent (response) variables. We can also use Stepwise Regression (SWR), a statistical technique that builds a regression model by adding and possibly removing independent variables one-at-a-time until some stopping rule is satisfied. Variables are removed from the model only if they become nonsignificant after the inclusion of other variables. Basically, the main goal is to find the best set of predictors (EP and cost drivers) and to build an equation to explain the variation in the response (manual test execution effort):

$$Effort = \alpha + \beta_0 EP + \beta_1 C_1 + \ldots + \beta_n C_n \tag{4.6}$$

where:

– $Effort$ is the effort to execute the test cases.

– EP is the number of execution points of the test cases.

– C_1 to C_n are the cost drivers selected as good predictors by the regression analysis.

– α and β_0 to β_n are coefficients defined during the regression analysis.

4.4 Final Considerations

In this chapter, we presented the planning of this research in terms of research goals, questions and hypotheses. We also presented our proposed measure of test size and execution complexity that is based on test specifications. Then, we presented different estimation approaches for manual test execution effort based on execution points. In addition to the presented estimation approaches, several others can be used, such as Case-Based Reasoning [139] and Bayesian networks [106].

In the next chapter, we present a sequence of empirical studies used to evaluate the adequacy of execution points for estimating manual test execution effort, to identify and investigate the effect of cost drivers for test execution, etc. For practical reasons, we investigate the use of only the productivity-based and the regression-based estimation approaches. We believe that these two approaches are already being used in industry and they can be highly accurate when correctly applied.

CHAPTER 5

Empirical Studies

The previous chapter presented how to estimate test execution effort based on test specifications and cost drivers. To evaluate our estimation approach, we run a sequence of empirical studies with the objective to create and calibrate estimation models for test execution effort, to evaluate our proposed size measure for test size and execution complexity, and to investigate cost drivers related to test execution. This chapter shows the planning, execution and analysis of these studies.

5.1 General Overview

Table 5.1 summarizes our empirical studies related to test execution effort estimation, showing for each empirical study (ES):

- A general classification for the study (experiment, survey, etc.).
- The relation between research questions, metrics, and main techniques and instruments used in the study:
  - Which research questions are investigated.
  - Which metrics are used to answer each question.
  - Which techniques and instruments produce the metrics used to answer our research questions.

The design of most of these studies are also discussed in [14].

5.1.1 Description of the empirical studies

In this section, we briefly overview the studies shown in Table 5.1. The details of these studies are presented in the next sections of this chapter.

ES1 – Expert Judgment: this first empirical study uses expert judgment through a Delphi panel to configure an initial version (characteristics and cost drivers, their values and weigths, etc.) of our test execution effort estimation approach. In this study, experts achieve a consensus about the list of characteristics to consider for measuring the size of test specifications on the mobile application domain. The experts are also asked to identify cost drivers that should be considered when estimating manual test execution effort. This study helps us to answer questions RQ1.1, RQ1.2, RQ1.5, RQ1.6 and RQ4.

Table 5.1 Empirical studies used to answer our research questions.

Goal	Question	Metric(s)	Empirical Study					
			ES1 [12] Expert Judgement	**ES2** [15] Case Study	**ES3** [18] Database Analysis	**ES4** Montecarlo Experiment	**ES5** Survey	**ES6** Designed Experiments
Based on test specifications	RQ1.1 – What are the relevant characteristics that should be considered when sizing a test specification?	Characteristics, $CValues_{c,l}$	Delphi panel			ANOVA		
	RQ1.2 – What is the weight of each relevant characteristic with respect to manual test execution effort?	$CWeight_{c,l}$	Delphi panel			Least Square		
	RQ1.3 – Can we define a soundness measure for test size and execution complexity?	$Inconsistences_1$, $Inconsistences_2$		Empirical demonstration				
	RQ1.4 – Can we automate the measurement of test size and execution complexity?	StepsNotAuto		Prototype development				
Cost drivers	RQ1.5 – What are the relevant cost drivers for estimating manual test execution effort?	CostDrivers, $CDValues_{cd,l}$	Delphi panel				Questionnaire	ANOVA
	RQ1.6 – What is the impact (weight) of each relevant cost driver on manual test execution effort?	$CDWeight_{cd,l}$	Delphi panel				Least Square	Least Square
Accuracy	RQ2 – Is the estimation accuracy of manual test execution effort improved by regarding the size and execution complexity of test specifications?	MMRE, PRED(.25)		Cross-validation/ Statistical test	Cross-validation/ Statistical test	ANOVA		
	RQ3 – Is the estimation accuracy of manual test execution effort improved by regarding cost drivers related to manual test execution?	MMRE, PRED(.25)						ANOVA
Low cost	RQ4 – What is the cost for estimating manual test execution effort based on the size and execution complexity of test specifications, and based on cost drivers related to manual test execution?	ConfigEffort, EvaluationEffort	Delphi panel					

ES2 – Case Study: in this study, tests are executed in a controlled environment in order to collect data and to perform different analyses: empirical demonstration of the soundness of execution points (question RQ1.3); automation of the execution points measurement method (question RQ1.4); evaluation of the accuracy of the estimation method configured in ES1 (question RQ2).

ES3 – Data Analysis: using a large database of test execution data, the objective of this study is to analyze and compare the accuracy of test execution effort estimates calculated by using historical execution times (a simple and common estimation method used in practice) and by using execution points (question RQ2).

ES4 – Montecarlo Experiment: the purpose of this experiment is to collect test execution data from different testers in a controlled environment and to verify the adequacy of different test size measures (execution points and others) for measuring test execution effort, helping to answer questions RQ1.1, RQ1.2 and RQ2.

ES5 – Survey: the purpose of this survey is to identify cost drivers not highlighted during the Delphi panel (ES1) through the application of a questionnaire to several testers of different test execution teams. (RQ1.5 and 1.6).

ES6 – Designed Experiments: this study has the objective of investigating the effect on test execution effort caused by cost drivers identified in ES5. For that, we design two experiments to be run in the industrial setting, but in a controlled environment. This study helps us to answer research questions RQ1.5, RQ1.6 and RQ3.

In addition, we run some other exploratory studies. One of them is presented in Appendix B, having the objective of analyzing the effect on test execution effort caused by cost drivers identified in ES5. For that, we analyze a historical test execution database and use expert knowlegde to gather information about the investigated cost drivers. Despite of the low cost of this study, the effect of confounding factors could not be controlled, creating the necessity of the designed experiment of ES6.

5.1.2 Definition of metrics

Following the Goal/Question/Metric method [24], we define the set of metrics to be collected during our empirical studies in order to answer our research questions.

- *MMRE*: Mean magnitude of the relative error.

$$MMRE = \frac{\sum MRE_i}{N} \tag{5.1}$$

  where:

  - $MRE_i = |RE_i|$
  - $RE_i = \frac{est_i - actual_i}{actual_i}$
  - N is the number of estimates.
  - est_i is the estimated execution effort of the ith test.
  - $actual_i$ is the actual execution effort of the ith test.

- $PRED(n)$: Percentage of estimates that are within n% of the actual values.

$$PRED(n) = \frac{\sum_{t=1}^{N}(1, ifMRE_t \leq \frac{n}{100}, 0, otherwise)}{N} \tag{5.2}$$

- *Characteristics*: List of characteristics of test specifications that are relevant to estimate test execution effort.

- $CValues_{c,l}$: List of values of the characteristic c that have a given influence level l (low, average or high) on test execution effort.

- $CWeight_{c,l}$: Weight (quantitative impact) assigned to the influence level l of the test characteristic c with respect to test execution effort.

- *CostDrivers*: List of cost drivers relevant to estimate test execution effort.

- $CDValues_{cd,l}$: List of values of the cost driver cd that have a given influence level l (low, average or high) on test execution effort.

- $CDWeight_{cd,l}$: Weight (quantitative impact) assigned to the influence level l of the cost driver cd with respect to test execution effort.

- $Inconsistences_1$: List of pair of tests (t_1, t_2), where t_1 and t_2 are intuitively similar with respect to their size and execution complexity, but with significantly different numbers of execution points.

- $Inconsistences_2$: List of pair of tests (t_1, t_2), where t_1 and t_2 are intuitively different with respect to their size and execution complexity, but with inconsistent numbers of execution points. For instance, a bigger test with a similar or a smaller number of execution points.

- $StepsNotAuto$: List of steps of the estimation model that were identified as difficult to automate.

5.1.3 Summary of main data analysis methods and techniques

Delphi [81] is a method that experts are invited to attend a panel for two or more rounds in order to achieve a consensus about a subject. The participants are coordinated by a facilitator who ensures their anonymity and structure the information flow between them. It promotes rapid consensus and avoid groupthink. We use this method to decide how to configure our estimation model based on expert judgment. For example, we use it to decide which cost drivers and test characteristics to consider in the estimation model.

Statistical hypothesis test is a statistical method used to test if a given hypothesis is true with a determined confidence level [131]. We use this kind of test to verify, for example, if a given characteristic or cost driver is relevant for estimating test execution effort.

ANOVA is a statistical method to test heterogeneity of means by the analysis of group variances [131]. In our research, we use ANOVA to identify the impact on the test execution

effort when varying the influence levels of characteristics and cost drivers. This information is also used to calculate the weight of these variables in our estimation approach.

Empirical demonstration is the confirmation that a given theory is correct by observing it in the practice [48]. For instance, we empirically demonstrate the soundness of execution points measured from test specifications.

Cross-validation [85] is a method for generalizing the results of a model evaluation. The main idea is to partition a sample of data into folds (subsets) such that you test the model on a single fold, while the other folds are used to build the model. The use of this method reduces the probability of obtaining results by chance.

Regression analysis [87] is a method used to create a mathematical model (called regression equation) that tries to explain the relationship between response and predictors variables. We use this method to identify the weights of the influence levels of each relevant characteristic or cost driver. We choose the regression model to be used (linear, quadratic, exponential, etc.) by doing linear and nonlinear correlation analyses between the effort and each cost driver. The Least Square Method is used to estimate the parameters of the regression model.

By Prototype development we mean the development of a tool that confirm the possibility to automate our proposed solution. Finally, Questionnaire [114] is a survey instrument that we use to gather information from experts.

5.2 Configuring a Test Execution Effort Estimation Model (ES1)

The first empirical study (E1) was run on the mobile application domain and it used expert judgement through a Delphi panel to instantiate a first version of the test execution effort estimation model. We did this panel with six experienced testers in two sessions of two hours.

In the first session, we explained to the participants how a Delphi panel works and the importance of that work for the organization and for themselves (better estimates avoid overtime, etc.). We also presented our approach for estimating manual test execution effort and the importance to calibrate it for the mobile application domain. Then, we asked the participant to answer questionnaires about the following questions:

- What characteristics of the tests (based on the specification) can be used to determine how big and complex the test is?
- In which situations these characteristics can be rated as having a Low, Average or High influence on the test size and execution complexity?
- Regarding the test size and execution complexity, what is the importance (weight) of each characteristic when compared to the others?
- For each characteristic, what is the importance (weight) of each of its influence Level to the test size and execution complexity?

We achieved a consensus in two rounds. The results are presented in Table 5.2, which shows the possible values for each characteristic, their weights and the guidelines showing the situations in which each value should be assigned.

Table 5.2 List of characteristics identified in the Delphi panel.

Functional characteristics				
ID	Description	Guideline / Value to Assign		
		Low	Average	High
1	Average number of navigations between screens.	Up to 3	More than 3	
		9	13	
2	Average size of date inputs.	Less than 30	30 to 100	More than 100
		15	60	150
3	Software configuration.		e-mail or IM account configuration	
			40	
4	File manipulation.	save files		Move files from different memories types or transfer files though network
		20		50
5	List manipulation.	search list entries, delete entries	add new entries	
		15	20	
6	Multimedia manipulation.	Standard sound/picture/animation manipulation	camera, emoticon	specific sound/picture/animation, edit picture
		21	26	38
7	Server access type.		Standard	Authenticated
			40	60
8	Type of screen items to be verified.	Valid characters	Invalid characters	Number of pixels
		18	35	50
Non functional characteristics				
ID	Description	Guideline / Value to Assign		
		Low	Average	High
1	Use of Bluetooth.	File transfer, print messages, use headset	Application data synchronization	
		30	50	
2	Use of network.	Sending IM messages or short e-mails or short messages	Sending large messages or large e-mails	
		25	55	

In the second session of the panel, we asked the experts about the following questions:

- What factors related to the test team, tested product, test environment and test process has influence in the effort to execute tests?
- In which situations these factors can be rated as having a Low, Average or High influence on the effort?
- What is the importance (weight) of each factor with respect to the others?
- For each factor, what is the importance (weight) of each influence Level?

After two rounds, we achieved a consensus about these questions. The results are presented in Table 5.3.

5.3 Evaluating Accuracy Improvement and Validity of EP (ES2)

As reported in [15], the main goal of this study was to analyze the accuracy of test execution effort estimates when using the estimation model proposed in this work. In order to do that, we compared the estimates given by our model with the ones calculated using historical averages of test execution times, a simple and common estimation method used in practice.

5.3.1 Planning

Following the goal-question-metric approach [24], we refined our goal for this empirical study to the questions presented next:

Q1: Is the average estimation error lower when using our model rather than using historical execution times?

Q2: Is the average percentage of estimates within 20% of the actual values higher when using our model rather than using historical execution times?

The answer for Q1 will indicate if the use of our estimation model results in a small error when regarding all estimates together. In its turn, the answer for Q2 will indicate if the number of estimates within 20% of the actual values increased when using our estimation model, a usual measure for estimation accuracy found in literature.

For the study, we selected 33 test cases of a messaging application feature for mobile phone. These test cases were written in a controlled natural language and their size and complexity were measured using our method. We wanted to compare the precision of estimates made using historical information with estimates made using our approach. As both approaches require information related to test productivity, we split the collected execution times into two sets of data, one for training and other for testing. The tests were randomly split, where the training set contained approximately 65% of the tests. All test cases were then executed by an experienced tester.

The execution times were collected and stored in a spreadsheet for analysis. We used the test execution times of the training set to calculate:

Table 5.3 List of cost drivers identified in the Delphi panel.

Risk factors				
ID	Description	Complexity Level / Guidelines / Rates		
		Low	Average	High
1	Schedule pressure		Short and tight schedules	Schedule overrun
			0.05	0.10
2	Tested system stability		Intermediate release, some tests performed	Initial release, almost not tested
			0.15	0.4
3	Network stability		Eventually not available	Frequently not available
			0.10	0.70
4	Server availability		Eventually not available	Frequently not available
			0.10	0.70
5	Support tools		Some eventual errors occur	Frequently errors occur
			0.05	0.30
6	Test setup reuse		Eventual	Frequently
			-0.20	-0.35
7	Test similarity		Eventual	Frequently
			-0.05	-0.15
8	Hardware performance	Lower than the average		Larger than the average
		0.20		-0.20
9	Tester capacity	Beginners		High experienced
		0.05		-0.05
10	Phone experience	Beginners		High experienced
		0.05		-0.05
11	Tested feature experience	Beginners		High experienced
		0.05		-0.05
12	Project precedence		It is similar to a previous one	It is a retest
			-0.05	-0.20

Table 5.4 Improvements achieved by using execution points.

Test	Reduction of MMRE	Increase of PRED(20)
1	36.75%	100.00%
2	36.12%	33.33%
3	17.19%	50.00%

- The average test execution time (for the historical data approach).
- The average time required to execute each execution point of a test case (the conversion factor for our proposed model).

With this information, we estimated the test execution effort of the testing set using both approaches. The following metrics were collected for answering Q1 and Q2.

- Mean magnitude of the relative estimation error.

$$MMRE = \frac{\sum_{t=1}^{T} MRE_t}{T}$$

where:

$MRE_t = abs(\frac{estimated_t - actual_t}{actual_t})$

$T = number\ of\ tests$

- Average percentage of estimates that were within 20% of the actual values.

$$PRED(.20) = \frac{\sum_{t=1}^{T}(1, if MRE_t \leq .20, 0, otherwise)}{T}$$

In order to avoid bias, we repeated the process two more times with different training and testing sets.

5.3.2 Execution and Analysis

We executed the case study as planned. The results are presented in Table 5.4, showing the improvements achieved for the metrics collected from the two analyzed estimation approaches. In all tests we achieved better or equivalent estimation precision. In the first test, for example, the number of estimates within 20% of the actual values increased by 100%. We also applied t-tests and confirmed the significance of the results.

5.3.3 Validity of Execution Points

In Section 4.2.2, we proposed a measure of the size and execution complexity attribute of a test case. We can evaluate the validity of this measure by demonstrating empirically that the mapping between the empirical relation system (T, R) to the numerical relation system (E, P) is valid [48].

During the experiment presented in the previous section, we mapped several test cases in T into execution points in E. It is necessary to verify that the mapped relations (in R and P) is valid considering the collected data.

We used expert judgement and effort information to identify similar tests and tests bigger than others with respect to their size and execution complexity. We verified that tests intuitively identified as similar tests had different measured numbers of execution points. However, the differences between these measures were within 20% of their values and this percentage value (p) can be used for identifying similar test cases from their number of execution points.

We also verified that tests identified as bigger than others had bigger measures for their size and execution complexity attribute. In summary, we demonstrated empirically that, for all t_a and t_b in T:

t_a *bigger than* $t_b \Leftrightarrow ep(t_a) >_{ep} ep(t_b)$

t_a *similar to* $t_b \Leftrightarrow ep(t_a) \approx_{ep} ep(t_b)$

where *ep(t)* is the number of execution points measured from t.

5.3.4 Discussion

In this empirical study, we observed the cost to use our proposed model, which was basically the time spent in the following activities:

- Define a controlled natural language;
- Identify relevant system characteristics;
- Define guidelines;
- Evaluate the size and execution complexity of test steps.

The cost to define a controlled natural language depends on the way we define it. For instance, we can only define general rules, such as: each test step must be an imperative sentence giving a direct command to the tester, the main verb in infinitive form that defines the test action must starts the sentence, etc. In this case, we do not have a significant cost.

However, we can also specify the list of all possible verbs that define test actions, their possible arguments, the vocabulary to be used, etc. In this case, we have to analyze existing requirement and test documents, a process that can be done incrementally. We also need support tools to store this information and to check the conformity of the test cases. In this way, we maximize the benefits of the controlled natural language: reduced grammar and lexicon, writing standard, etc.

Finally, the cost to evaluate each test action, with respect to size and execution complexity, was very small. It took less than a minute to evaluate each test step. However, there may exist hundreds of test steps to be evaluated. Fortunately, we did not need to evaluate all test steps, since it is common to have the same test step occurring several times in different test cases or even in the same test. After evaluating a test step, we just need to assign the same number of execution points to its other occurrences.

In our experiment, we observed that most of times we can evaluate a test step based only on the main verb of its sentence, independently of the verb arguments. For instance, the act of launching an application has the same complexity for most applications and only the exceptions need to have a specific evaluation.

Also, we use a controlled natural language that reduces the vocabulary and consequently increases the use of the same verb in different test steps that only change the verb arguments. For this reason, the number of test steps to evaluate (and our cost) is even smaller. In summary, the results of this empirical study suggests the feasibility of our model regarding the cost of using it.

5.4 Evaluating Accuracy Improvement Using Historical Data (ES3)

In this section, we present the details of an empirical study which main goal is the analyis and comparison of the accuracy of test execution effort estimates calculated by using historical execution times (a simple and common estimation method used in practice) and by using execution points.

5.4.1 Planning

Based on the goal of this study, we defined the following research question:

Q1: What is the impact on estimation accuracy when using execution points instead of historical execution times?

To answer this research question, we compare estimates from the model presented in Section 4.3.1 (test productivity-based model) with estimates calculated by multiplying the number of test to be executed by the average time spent in the past to execute a test. For this comparison, we consider the Mean Magnitude of the Relative estimation Error (MMRE) and average percentage of estimates that were within 25% of the actual values (PRED(.25)), two measures commonly used to assess estimation accuracy.

In ES2, we analyzed the accuracy improvement by analyzing a small set of tests. In this study, we use a large historical database of manual execution on the mobile application domain. For measuring the execution points, we use the characteristics and weights for the mobile application that were identified in ES1 through the Delphi panel with test experts. We use the two estimation approaches to estimate test execution effort of previous test projects (historical database) in a ten-fold cross-validation analysis.

5.4.2 Execution and Analysis

The analyzed database has information about the execution of 319 different test suites (sets of related tests) used to test the features of different types of mobile applications. Table 5.5 presents descriptive statistics of the analyzed data.

Table 5.5 Descriptive statistics of the historical database used in the study.

Test suite characteristic	Mean	Median	Min	Max	Std Dev
Number of tests	48.55	57	15	104	21.37
Execution points	14935.12	15553	6086	28174	4641.48
Effort (hh:mm)	07:05	06:56	02:28	14:40	02:35

During the cross-validation analysis, we observed a reduction (from 49.49% to 66.02%) of the Mean Magnitude of the Relative estimation Error (MMRE) and an increase (from 20.83% to 93.75%) of the average percentage of estimates that were within 25% of the actual values (PRED(.25)). We used a paired t-test to reject our null hypothesis (p-value$<$0.001, $\alpha = 0.05$), which says that the mean accuracy of both estimation approaches are the same according to MMRE. Table 5.6 shows the accuracy improvement achieved by the use of execution points during the cross-validation analysis.

Table 5.6 Accuracy improvement achieved by using execution points (EP) against of using historical execution times (HET).

	MMRE			PRED(.25)		
Fold	HET	EP	Reduction	HET	EP	Increase
1	0.30	0.12	60.37%	0.50	0.97	93.75%
2	0.28	0.11	60.65%	0.59	0.94	57.89%
3	0.27	0.12	53.83%	0.53	0.94	76.47%
4	0.28	0.10	66.02%	0.72	1.00	39.13%
5	0.30	0.14	52.70%	0.63	0.81	30.00%
6	0.36	0.17	53.00%	0.63	0.84	35.00%
7	0.30	0.11	64.11%	0.56	1.00	77.78%
8	0.30	0.12	58.91%	0.75	0.91	20.83%
9	0.33	0.12	62.93%	0.56	0.88	55.56%
10	0.31	0.15	49.49%	0.68	0.90	33.33%
Avg	0.30	0.13	58.20%	0.61	0.92	51.97%

We also analyzed the correlation between the execution points of the tests, number of tests and the effort to execute them. The linear correlation between effort and execution points was significative, achieving a Pearson's correlation of 0.888 (p-value$<$0.001). This num-

ber was larger than the correlation between effort and number of tests, which was 0.792 (p-value$<$0.001).

5.4.3 Limitations and Threats to Validity

This study has some limitations and threats to validity. First, we did not consider the variation that can occur when changing the group of experts used to define test characteristics, guidelines, weights, etc. As a consequence, we do not know if this variation can significantly change the observed results. In addition, we used historical data that may be affected by some uncontrolled factors, such as test environment conditions, time pressure, etc. Although the cross-validation reduced the impact of these factors, more precise results can be achieved by controlling all relevant factors.

Although the test suites in the historical database are different, we observed that several of them have tests in common with other test suites. This characteristic may increase the accuracy metrics of both investigated estimation models, since the same test (with similar test execution times) can occur in both training and test sets of the cross-validation. As we used the same training and test sets for both investigated models, this effect is reduced when we analyze only the accuracy improvement.

5.4.4 Conclusions

In this study, we observed a significant accuracy improvement caused by execution points when compared to average test execution times. In addition, we observed a high estimation accuracy for the both investigated estimation approaches. This can be an effect of having the same tests in both training and test sets of the cross-validation. Also, the tested features are similar to those used to configure the execution points measurement method. The use of features different from those used to configure the measurement method may lead to smaller accuracy improvements.

5.5 Evaluating the Adequacy of Test Size Measures (ES4)

The accuracy of test execution effort estimation strongly depends on the selection of adequate predictor variables. In Chapter 4, we propose the use of execution points to improve the accuracy of test execution effort estimation. In this study, we compare the adequacy of four test size measures for estimating test execution effort. All these measures are related to the size of test specifications. Three of them are based on the ideas presented in Chapter 4 and the other is a measure commonly used in the practice.

5.5.1 Research Objective

Following the goal template of the Goal/Question/Metric (GQM) method [24], we define the research objective of this experiment as:

> Analyze *test size measures* for the purpose of *evaluation* with respect to *its adequacy for estimating test execution effort* from the point of view of *test managers* in the context of

functional tests on the mobile application domain.

5.5.2 Size Measures Under Investigation

For this experiment, we only consider measures that we believe to be appropriate for sizing test specifications and to be used in the practice. These measures are detailed next.

5.5.2.1 EP calibrated by specialists (M1) and by OLS (M2)

These two measures (M1 and M2) are basically the Execution Points (EP) measured by two differently calibrated measurement methods. The former is a measure obtained by using a measurement method calibrated by using expert opinion (see empirical study ES1). The latter is a measure obtained by using a measurement method calibrated by analyzing test execution data (see Section 4.2.3.3), which uses the Ordinal Least Square (OLS) method to fit a linear model and to obtain the weights to use in the model.

5.5.2.2 Number of test steps (M3)

One way to measure the size of a test specification is counting its number of test steps. This simple measurement method can be easily automated and it does not require any calibration procedure, since it is considered that all steps in the test specification have the same weight. For instance, the test specification presented in Table 5.7 has the size of 5 test steps. The size of a test suite is then calculated as the sum of steps of each individual test specification of the suite.

Table 5.7 Example of a test procedure written in a controlled or standardized 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.

This measure is very simple to compute and it is used in the practice, although its usefulness strongly depends on the use of a standardized test specification languages (description in the same level of details, etc.). The tests specifications considered in this study have an homogeneous level of detail and follow writting standards defined by the industry involved in this research.

5.5.2.3 Multiple characteristics (M4)

We can see the execution points measurement method as a way for aggregating several characteristics observed in test specifications that are all related to test execution effort. A different method for estimating test execution effort is considering each of these characteristics as a different and complementary measure of test size. In other words, the characteristics Number of navigations between screens (Nav), Number of press keys (Keys), etc., are considered as size measures for estimating test execution effort. Then, all these multiple measures can be used to build a model for estimating test execution effort based. A similar approach was successuly used for estimating web development effort [95].

5.5.3 General Planning

In this section, we describe several information about the planning of our experiment, such as its goals, the participants, hyphotheses, experiment design, data analysis procedures and results, etc.

5.5.3.1 Goal

The major goal of this experiment is the evaluation of how useful the investigated test specification measures are to estimate test execution effort. The assessment of this goal is performed by answering the following research questions:

RQ1: How accurate are the estimates based on the investigated measures?

RQ2: What is the best measure to use for estimating test execution effort?

We analyze the accuracy of test execution effort estimation models that use the investigated measures (question RQ1) by using multiple and standard accuracy measures [129]:

MMRE: Mean magnitude of the relative error.

MdMRE: Median magnitude of the relative error.

PRED(25): Percentage of estimates that are within 25% of the actual values.

Then, the best investigated measure considered here (question RQ2) is the one that achieved the best results with the accuracy metrics (MMRE, MdMRE, PRED) and the following metric:

PRESS: Prediction Sum of Squares statistic, which is a leave-one-out refitting and prediction method [31]. It measures how a regression model is sensitive to particular data itens and it is calculated by:

$$PRESS = \sum_{i=1}^{n} (y_i - \hat{y}_{i,-i})^2 \tag{5.3}$$

where y_i is the actual y value for the point x_i and $\hat{y}_{i,-i}$ is the predicted value using a model calibrated by n - 1 points (all data points except x_i).

5.5.3.2 Participants

The participants of this experiment were students of a testing course at the Informatics Center/UFPE in partinership with an important cell phone manufactor. In this course, graduate and undergraduate students have class during half of the day and the remaining time is used to practice software testing in a real test site on the mobile application domain. All students of this testing course are required to participate in a research project, being evaluated according to the achieved results. Usually in groups of three, these students choose their research project from a predefined list of options according to their interest and some project prerequisites (experience of the participants, etc.).

In this context, the main motivation of the participants (three student volunteers) were the interest in the topic of study and the fulfilment of the testing course requirements (research project) with their participation in the experiment. These students worked as testers for approximately six months, having experience on executing functional tests on the mobile applications. All the participants had almost the same testing experience background, existing only some individual testing experiences with respect to specific features and types of mobile phones.

5.5.3.3 Experimental Material

The tests specifications considered in this experiment were obtained from a real test repository. These tests were previously designed to test a feature related to message applications running on mobile phones. The complete test suite contain 47 functional tests that exercise all the 7 requirements of the considered feature.

We selected these tests based on their availability, previous knowlegde of the participants about the tests (no training required) and the high quality of the specification writing, which reduced any confounding effect related to who wrote the tests. Also, we verified that these tests have the most common test actions found in functional tests for mobile applications.

5.5.3.4 Tasks

The task of the participants in this study is to execute a set of functional tests in a random order assigned to each one of them. For this activity, the test environment (hardware and software) is already prepared. Also, the testers are asked and monitored to run the tests without being affected by any external factor, such as time pressure, noisy environment, etc.

They executed the tests using ManualTEST [17] (see Appendix D), a tool that we developed to improve the accuracy of the data collected in our empirical studies and to provide information to locate and identify sources of problems occurred during the execution such studies. Hence, the participants have to follow the procedures to use the tool, which include the logging of problems during the test execution, such as any influence of confounding factors.

5.5.3.5 Hypotheses, Fixed Factors and Variables

For evaluating the adequacy of each measure under investigation for estimating test execution effort, we compare the mean accuracy obtained by their respectives estimation models. We verify the statistical significance of the differences between the means through the test of the

following statistical hyphoteses:

$$H_{0i} : \mu_{iM1} = \mu_{iM2} = \mu_{iM3} = \mu_{iM4}$$

$$H_{1i} : \mu_{iMj} \neq \mu_{iMk} \text{ for at least one } Mj \neq Mk, \quad \text{Mj,Mk=M1, M2, M3, M4}$$

Where i = MMRE, MdMRE, PRED(25) and M1 to M4 are the measures under investigation.

The test of these hypotheses shows if there is a statistical significant difference when using some of the measures under investigation for estimating test execution effort. If this difference exists, we identify the measure(s) that provides the best estimation accuracy by performing simultaneous inferences and multiple comparisons with respect to the resulting accuracy ($MMRE$, $MdMRE$ and $PRED(25)$).

There are several factors that can influence the accuracy of the test execution effort estimates. As we intend to analyse only the influence of the size measures on the accuracy of the estimates, the effect of these other factors have to be controlled. The value for most of these factors were fixed for the experiment, as presented in Table 5.8.

In addition, we present the variables defined for this experiment in Table 5.9. We have the independent variables $Size_{M1}$ to $Size_{M4}$, the factor $MeasureType$ representing the type of size measure to be used, the actual test execution effort ($ActEffort$) and the response variable $EstEffort$, which is the estimated test execution effort by using one of the investigated size measure.

5.5.4 Experiment Design

This experiment is organized in three stages. In the first stage, we collect data using three testers executing tests in a controlled environment. In the second stage, we use this data to create test execution effort estimation models based on the investigated size measures. The third stage consists in calibrating and evaluating the created models using a cross-validation analysis in order to analyze the accuracy provided by each investigated size measure.

5.5.4.1 Stage I: Collecting Data

In this first stage, three participants (testers) execute all the tests described in Section 5.5.3.3 in a controlled environment, each tester following a different and random test execution order. The objective of this stage of the experiment is to determine the actual effort required to execute each test. We reduce and control the effect on test execution effort caused by confounding factors through the following methods:

- Blocking. We block the testers who execute the tests, since they have similar but not equal previous experiences and others personal characteristics that affect their performance.

- Randomization. The order of test execution may affect the total execution effort. We assign a different and random order of test execution for each tester. This forces a more homogeneous contribution of the test execution order with respect to the execution effort.

Table 5.8 Values set for fixed factors in this experiment.

Related to	Factor Name	Value
Environment	Tools	All availables testing tools are used.
	Resources	Mail servers are available and operating with the usual performance.
	Stability	Mobile network is available and the sign is stable.
Personnel	Participant level of expertise	Several months of experience on testing. Similar testing experience and academic performance.
	Familiarity of participants with the tested feature, phone and tools	Good familiarity.
	Current occupation	Subjects from a university course and half-time employee.
Test project	Software testing technique	Functional testing.
	Test size	Set of similar tests with size that is common on the mobile application domain.
	Application domain	Mobile applications.
	Application complexity	Feature with small complexity.
	Tested application	Feature related to mobile message applications.
	Tested phone	A stable phone model with regular performance.
	External pressures	Low level of schedule preassure to execute the tests.

Table 5.9 Variables defined for this experiment.

Name	Abbreviation	Class	Entity	Type of attribute	Scale type	Unit	Range or Scale points
Size of the test specifications determined by measure M1	$Size_{M1}$	Product	Test Specifications	Internal	Ratio	Execution point	[0, ∞]
Size of the test specifications determined by measure M2	$Size_{M2}$	Product	Test Specifications	Internal	Ratio	Execution point	[0, ∞]
Size of the test specifications determined by measure M3	$Size_{M3}$	Product	Test Specifications	Internal	Ratio	Test	Non-negative integers
Size of the test specifications determined by measure M4	$Size_{M4}$	Product	Test Specifications	Internal	Ratio	Step	Non-negative integers
Id of the tester that executed a test	TesterId	Process	Tester	Internal	Nominal	-	A, B, C
Type of measure and measurement method to be used for estimating the required test execution effort	MeasureType	Method	Size measurement	Internal	Nominal	-	M1, M2, M3, M4
Estimated effort required to execute the tests	EstEffort	Process	Test Execution	Internal	Ratio	Hour	[0, ∞]
Actual effort required to execute the tests	ActEffort	Process	Test Execution	External	Ratio	Hour	[0, ∞]

- Holding project and environmental conditions constant during the experiment. These are the fixed factors already described.

Hence, we consider that only the size of the testers and the testers's experience can significantly affect the test execution effort. Then, we record information about the test and the tester who executed each test in a database, as well as its actual test execution effort. As the test is executed by only one tester, test execution effort means the time spent to execute the test. The structure of this database is described in Table 5.10.

Table 5.10 Information collected in the first part of the experiment.

Name	Description
TestId	Identification of the test that was executed.
M1	Size of the test case determined by the M1 measurement method.
M2	Size of the test case determined by the M2 measurement method.
M3	Size of the test case determined by the M3 measurement method.
ActualEffort	Actual effort required to execute the test.
TesterId	Identification of the tester that executed the test.

All test size measures are automatically calculated by using a tool [20] (see Appendix C), avoiding measurement errors. The test execution data is collected by the ManualTEST tool, avoiding most of the data collection problems.

5.5.4.2 Stage II: Creating Estimation Models

We use regression analysis [53] for creating the estimation models used to investigate the adequacy of the size measures. We selected this technique since regression analysis is a well-known statistical technique largely used in industry and in the academia for creating estimation models. In summary, regression analysis a statistical technique that investigates the relationship between dependent variables (response variables) and independent variables (predictor variables). This relationship is defined by an regression model, which is in our case:

$$ExecTime_k = \beta_0 + \beta_1 Size_{mk} + \beta_2 T_{A_k} + \beta_3 T_{B_k} + \beta_4 T_{A_k} * Size_{mk} + \beta_5 T_{B_k} * Size_{mk} + \varepsilon_k \quad (5.4)$$

Where:

- $ExecTime_k$ is the time spent to execute the test of the kth observation.
- β_0 to β_5 are coefficients of the estimation model to be calculated by the regression analysis.
- $Size_{mk}$ is the size of the test executed in the kth observation determined by the measure m (M1, M2, M3 or M4).

- T_{A_k} is equal to 1 if tester A executed the test of the kth observation, otherwise it is equal to 0.

- T_{B_k} is equal to 1 if tester B executed the test in the kth observation, otherwise it is equal to 0.

- ε_k is the error term for the kth observation.

As we can see, we use a general linear model that includes the possible interactions between the test size ($Size_{mk}$) and the tester's experience (dummy variables [85] T_{A_k} and T_{B_k}). For each investigated measure, the regression analysis is applied to the database created in stage I with the objective of determining which β coefficients are different from 0. Also, transformations in the model can be performed to meet some regression analysis assumptions:

- The residual error term follows a normal or an approximate normal distribution.

- The residuals have a constant variance.

5.5.4.3 Stage III: Evaluating Estimation Accuracy

In this stage, we calibrate and evaluate the four models created in Stage II (one for each investigated measure). In summary, this stage has the objective of analyzing the accuracy provided by each investigated measure for estimating test execution effort. For that, we calibrate and evaluate the estimation models using a three-fold cross-validation arrangement [85].

In this three-fold cross-validation, the data collected in Stage I is randomly split into three folds of sizes as similar as possible. Then, each fold is used to verify the accuracy of each estimation model when they are calibrated based on the other folds. Since this experiment blocks the testers, this cross-validation analysis is executed three times, each time considering the data obtained from a different tester. We run this procedure one hundred times to reduce the experimental error variance. After that, we analyze the achieved accuracy, performing statistical tests to support our conclusions.

5.5.5 Execution and Analysis

This study was executed according to the planning presented in the previous sections. The details of this execution and its analysis are presented next.

5.5.5.1 Data Collected in Stage I

During the execution of some tests, the testers observed and logged the effect of some confounding factors, such as changes in the network performance. These problems were confirmed by analyzing the data recorded by the ManualTEST tool. For this reason, only 130 of the 141 observations were considered in the study, 44 from tester A, 43 from tester B and 43 from tester C.

5.5.5.2 Estimation Models Created in Stage II

As presented next, we created four estimation models, one for each investigated size measure. The details of the statistical analyses performed to create the models are presented in Appendix A.

Measure M1. We applied the regression analysis considering measure M1 (EP-Experts). The results of this analysis are presented in Figure A.1. As we can observe in the residual by predicted plot, the variance of the residuals is not constant. The distribution of the data is also suggesting that there may exist some non-linear relation between the dependent and independent variables.

To stabilize variance of the residuals, we applied the Box Cox [53] transformation on the variable effort. The results of this analysis are presented in Figure A.2. As we can observe in the residual by predicted plot, the variance of the residuals was stabilized. We also observed that there is some non-linear effect of size on the transformed effort that we approximated by a quadratic effect (see Figure A.3).

After that, we performed again the regression analysis (see Figure A.4). The inclusion of the quadratic effect of the measure M1 on test execution effort improved in some way the linear relationship between the actual and the predicted effort (see Figure A.5 and the changes in RSquare Adj from Figure A.2 to A.4).

Finally, we removed the terms that were not considered significant by the regression analysis. The final result from this analysis is presented in Figure A.6. Although the dummy variable for tester B was not considered significant, we kept it in the model, since the other dummy variable was significant. We also confirmed the normal distribution of the residuals by analyzing the data histograms and applying the Shapiro-Wilk W Test [131].

Measure M2. The results of applying regression analysis considering measure M2 (EP-Data) are presented in Figure A.7. As we can observe in the residual by predicted plot, the variance of the residuals is not constant. We transformed the effort data using the Box-Cox transformation. After that, we verified that all factors and interactions should be kept in the model, as presented in Figure A.8.

Measure M3. We applied the regression analysis considering measure M3 (Steps) and we verified the variance not constant of the residuals (Figure A.9). After transforming the effort data using the Box-Cox transformation method, we removed the interactions between the size measure and the testers (dummy variables), since they did not have significant effect (Figure A.10). The resulting model was considered the final model for measure M3 (see Figure A.11).

Measure M4. Before applying the regression analysis considering measure M4 (measures Keys, Screen, Delay, ListManip), we analyzed the correlation between the four measures represented by the multiple measure M4. We observed a high correlation between some of them: Keys and Screen and Keys and ListManip (see Table A.1). Using the partial correlation analysis, we decided to remove the variable Key.

Then, we applied the regression analysis and verified the variance not constant of the residuals (Figure A.12). We transformed the data using the Box-Cox transformation and verified

that some interactions have not significant effect (Figure A.13). To reach the final regression model, we first removed the Screen and Tester interaction term (Figure A.14) and then the ListManip and Tester interaction term (Figure A.15).

5.5.5.3 Estimation Accuracy Achieved in Stage III

We performed the Stage III of this study according to the planning by implementing scripts in the statistical package R [37]. The estimation accuracy (MMRE, MdMRE and PRED(25)) achieved in each run and by each estimation model is presented in Table A.2. The analysis of the MMRE distribution is presented in Figure A.16. The values observed for the MMRE are distributed between 0.2 and 0.9 and follow an approximate normal distribution. The MMRE means and their confidence intervals at $\alpha = 0.05$ are presented in Table 5.11.

Table 5.11 MMRE means and 95% confidence intervals.

Measure	lower 95% Mean	MMRE Mean	upper 95% Mean	Rank
M1	0.3924	0.4156	0.4387	3
M2	0.3264	0.3321	0.3378	1
M3	0.3890	0.3978	0.4066	3
M4	0.3502	0.3619	0.3737	2

As we can see, the statistical analysis of confidence intervals shows an intersection only between the confidence intervals of the MMRE means for M1 and M3. This result rejects the null hypothesis $H0_{MMRE}$ at $\alpha = 0.05$, which says that the choice between the investigated measures did not impact the achived estimation accuracy measured by MMRE. The column Rank shows which measures had the best results with statistical significance.

With respect to the MdMRE means, the analysis of their distributions is presented in Figure A.17. The values observed for the MdMRE also follow an approximate normal distribution. In addition, the results also reject the null hypothesis $H0_{MdMRE}$ at $\alpha = 0.05$, which says that the choice between the investigated measures did not impact the achived estimation accuracy measured by MdMRE. As we can see in Table 5.12, we observed statistical significant differences between some of the Median means. Again, column Rank shows which measures had the best results with statistical significance.

Table 5.12 MdMRE means and 95% confidence intervals.

Measure	lower 95% Mean	MdMRE Mean	upper 95% Mean	Rank
M1	0.2886459	0.2960826	0.3035194	2
M2	0.2377913	0.2434163	0.2490413	1
M3	0.3029377	0.3106087	0.3182798	2
M4	0.2371364	0.2437111	0.2502858	1

Finally, the analysis of the PRED(25) distribution is presented in Figure A.18. The values observed for the PRED(25) also follow an approximate normal distribution. The results reject the null hypothesis $H0_{PRED(25)}$ at $\alpha = 0.05$, which says that the choice between the investigated measures did not impact the achived estimation accuracy measured by PRED(25). Column Rank of Table 5.13 shows which measures had the best results with statistical significance.

Table 5.13 PRED(25) means and 95% confidence intervals.

Measure	lower 95% Mean	PRED(25) Mean	upper 95% Mean	Rank
M1	43.548274	44.477639	45.407003	2
M2	50.792641	51.581219	52.369796	1
M3	42.477479	43.300828	44.124177	2
M4	51.026752	51.904510	52.782268	1

5.5.6 Discussion

5.5.6.1 Evaluation of Results and Implications

We answer question RQ1 by using the calculated means for MMRE, MdMRE and PRED(25) for each estimation model used in the experiment. Different from what we expected, none of the models based on the characteristics of test specifications (M1, M2, and M4) achieved the great estimation accuracy observed in previous studies, that is, MMRE and MdMRE close to 0.2 and PRED(25) close to 80%. We believe that this decrease in the observed estimation accuracy for these models is due to specific characteristics of this feature (intensive use of network, etc.) that were not identified in empirical study ES1, since that study was based on different features. Also, it is difficulty to control the effect of some confounding factors (tester experience, network performance, etc.), which may reduce the accuracy of all investigated models. However, the comparison between the estimation models was not impacted, since the models were evaluated using the same conditions.

For answering question RQ2, we observed which of the investigated measures produced estimation models with better performance in the accuracy metrics (MMRE, MdMRE and PRED(25)) and in the PRESS metric. In the mean, this study shown with statistical significance that MMRE was smaller when we use the measure M2. Actually, the maximal MMRE value obtained by using M2 was almost the half of the maximal value obtained by using M4. However, the MdMRE and the PRED(25) were similar for mesaures M2 and M4 with respect to their means. With respect to the PRESS metric, measure M2 resulted in a smaller PRESS than the other measures (see Table 5.14). In summary, based on the MMRE and PRESS metrics observed in this study, we consider measure M2 as the best measure to use for estimating test execution effort.

Table 5.14 PRESS statistic for the models created by using each of the investigated measures.

Measure:	M1	M2	M3	M4
PRESS:	343933.25389	302215.05609	382953.04261	303771.93926

5.5.6.2 Threats to Validity

In this section, we describe threats to the validity of the experiments results, the impact of each threat and how we controlled them.

Construct validity: The execution points measurement method was configured in ES1 by testers that did not executed this feature. For this reason, some characteristics present only in test specifications of this feature may be missing in the measurement method, reducing the accuracy of the models based on characteristics of test specifications (metrics M1, M2, and M4). We asked the participants to report any characteristic that was missed, in their opinion.

Internal validity: Only one of the participants (testers) was expert in testing the feature used in the experiment. The other two testers knew the feature and have tested it before, but long time ago and only part of their requirements. We noticed that the experience of these two testers had different effect on test executin effort, according to each test. For instance, these testers had the effect of an experienced testers for tests similar to those frequently executed by them. However, they may had different performance for tests different from those frequently executed by them. This problem can increase the variance of the effort spent to execute each test and can reduce the effectiveness of our mechanism used to control the testers's experience effect (dummy variables, etc.). The possible consequence of this confounding effect is a reduction in the estimation accuracy of all investigated models. The control of this problem was performed through the analysis of outliers, where some observations were discarded.

In addition, the tests could not be executed in the same day, making difficulty the control of the testing environment. A possible consequence of this problem is an increase of the effects of confounding factors, reducing the estimation accuracy of all investigated models. To control this problem, we asked the testers to report any observed variation in the testing environment.

External validity: The participants and material used in the experiment limit the generalization of our findings. The use of three different testers, tests containing the most common test actions, and a cross-validation analysis repeated several times increases the probability to find results not by chance.

5.6 Identifying Cost Drivers Related to Test Execution (ES5)

The purpose of the survey was to identify what information is necessary to accurately estimate test execution effort on the mobile application domain. We defined the survey objective through the following research question with respect to the mobile application domain:

RQ: What cost drivers (test environment, team capability, etc.) significantly impact the effort to execute test cases?

This survey was planned to run during two business days of August, 2007. The last months before the application of this survey were considered common in terms of number and types of test projects. For this reason, we believe that the participants could remember the different test scenarios and environment conditions they have experienced before.

Next, we discuss about the main concerns about the survey administration.

5.6.1 Target population and sample size

The group of people able to answer our questions are the test architects, test managers, test designers and testers of the mobile application domain.

This work is part of the Motorola Brazil Test Center (BTC) Research Project, a partnership between Motorola Industrial Ltda. and the Informatics Center of the Federal University of Pernambuco that allowed us to have access to real test projects, teams and environments of Motorola BTC. We consider this industrial setting very representative in the mobile application domain, due to the high number of mobile applications that are developed by Motorola.

We were able to apply the survey to two Motorola test sites in Brazil (called here as test sites A and B). These sites together have more than 150 people working on testing mobile applications. We considered that a sample size of approximately 40 respondents were enough to get the information we needed and also feasible to run within our resources.

Using our previous experiences working with these test sites, we believed that our survey would achieve a response rate of 80%. For this reason, we decided to use a sample size of 50 people, as the analysis of a bigger sample would be too costly.

5.6.2 Participant selection and motivation

We wanted our sample to be representative. This means that it should ideally include people not only from all positions, but also from all different test teams with respect to the type of test and tested technology, since they may have different necessities and test environments. Both Motorola sites have similar organizational structure with respect to teams and positions. Hence, we selected 25 participants from each site using two different approaches.

For the test site A, the participants were randomly selected according to their positions, including test managers, test architects, test designers, testers and others. We split its list of employees into subgroups of the same position and then sample them separately.

For the test site B, the participants were randomly selected according to their test teams. We split its list of employees into subgroups of the same test teams and then sample them separately.

This stratified random sample has more chance to have participants from all positions and all test teams, since there are more testers than any other position. The number of members selected from each subgroup was proportional to the size of the subgroup.

Aiming to achieve a high response rate, we motivated the participants in different ways. First, we presented the goals of the survey in the beginning of the questionnaire, as well the benefits that an improvement in the accuracy of test execution effort estimations may result for the respondents.

Also, we designed the questionnaire to take 10 to 20 minutes to be answered. We believed

that most of the participants would not have more than 30 minutes to spend answering the questionnaire. The test managers were also asked to reinforce the importance of the survey.

We verified which participants responded the survey at the end of each day. We sent an e-mail to those that did not respond in order to remind the importance of their answers. We also approached the unresponsive participants four hours before the response deadline.

5.6.3 Questionnaire design

We designed a questionnaire having three parts. The first part is the survey description, which presents the survey goals, benefits and recommendations about how to answer the questions. For instance, we recommended the respondents to spend some time remembering previous experiences that impacted in the test execution effort.

The second part of the questionnaire regards the respondent position and profile. For instance, the participants were asked about their experience in testing, their position, etc. This information is useful to analyze the representativeness of the answers.

Finally, the last part of the questionnaire are related to the factors that may impact text execution effort. We tried to create simple questions that help us to identify these cost drivers. Basically, we created four questions asking what factors or characteristics impact test execution effort.

Each one of these four questions regards a specific point of view: the tested hardware (mobile phone), tested software (feature), test environment or tester capability. In addition, we defined a last question asking for any additional information that the respondent may think relevant to test execution effort estimation. We used a customized spreadsheet as our data collection instrument.

5.6.4 Methods for data analysis

For analyzing the responses, we first consolidate all responses into a single spreadsheet. Then, we screen the responses to ensure their correctness, standardizing the way they are written, removing ambiguities, etc. After that, we analyze the reviewed answers and group them using keywords. We use these keywords and their frequency of occurrence to help the identification of the cost drivers. This process is called *coding* in the qualitative analysis theory [127].

5.6.5 Questionnaire evaluation

Before running the survey, we evaluated the questionnaire through a pilot study. In this pilot, we performed the same procedures defined for the survey, excepting the participant selection. We invited seven testers to answer the questionnaire. Three of them were already collaborating with this research. The others four testers were selected due to their availability.

The answers were totally relevant, suggesting a good understanding of the questions. Also, the pilot was useful to identify missing questions related to the respondent profile. In addition, we verified that respondents took between 9 to 30 minutes to answer the questionnaire, as shown in Table 5.15.

In the beginning, we planned to collect the answers and, after three days, give the opportu-

Table 5.15 Effort to answer the questionnaire.

Time in minutes				
Mean	Median	Std Dev	Max	Min
20.43	20	8.04	30	9

nity for each respondent to read his answers and improve them with other information that he may have remembered. However, we verified in the pilot that the answers were not improved as we expected. Probably, no different situation happened during those days. For this reason, the respondent were not asked to review their answers in the survey as we did in the pilot.

5.6.6 Data analysis

In this section, we describe the results found during the analysis of the questionnaire answers. As shown in Table 5.16, we received 37 responses from the total of 50 participants, achieving a response rate of 74%. The response rate is presented according to each participant test site. Although this response rate is close to the one we were expecting, we did not expected the low response rate of test site B.

Table 5.16 Participants and response rate per test site.

	Test Site		Total
	A	B	
Participants	25	25	50
Responses	25	12	37
Response rate	100%	48%	74%

We tried to understand why some participants did not respond the survey. We asked them to justify why they did not answer the questionnaire. We also explained that the justifications are anonymous in order to have more reliable information. Some of the participants presented more than one justification, as presented in Table 5.17.

After analyzing the justifications, we did not find any pattern that could invalidate the survey. Most of the participants that did not answered the questionnaire due to lack of time. We received answers from at least 75% of the test teams, suggesting that the sample still representative. For these reasons, although the achieved response rate is lower than the one expected, we believe that the survey results were not significantly impacted.

We identified several cost drivers during the analysis of the responses. Table 5.18 lists the identified cost drivers and presents how they impact test execution effort. The other comments that did not fit a cost driver were presented to the test managers and quality engineers as a list of possible improvement opportunities.

Table 5.17 Justifications for not responding the survey.

Justification	Frequency	Percentage
I did not read the mail in the first day. The next day I did not have time.	6	46.15%
I did not have time.	4	30.78%
I felt tired answering the questionnaire and I postpone it. At the end, I did not have time.	1	7.69%
I am on vacation.	1	7.69%
I had to do an unexpected task. For this reason, I did not have time.	1	7.69%
TOTAL	13	100%

5.6.7 Threats to validity

In this section, we describe the threats to our survey validity and how we controlled them. Some of the questions of our questionnaire were answered in a too generic way, giving none or few details about what they mean. These generic answers could be misinterpreted during our analysis. To avoid this threat, these answers were discussed between real testers. In the worst case, the respondent was asked to give more detail about his answer.

Also, the way the questions were formulated led the respondents to answer what they think about specific subjects of our interest. However, we were not able to identify what reported situations or comments were really based on their own experiences. For instance, a respondent may have reported an impact on test execution effort just because he suppose it is true. The impact of this threat may be an increase in the number of answers to analyze. During the analysis, we were able to discard some answers that were identified as unreal or unlikely situations.

The quality of the answers for open questions depend on how well formulated is the question. We used a pilot study to ensure that the questions were understandable and the answers were useful. Besides, the respondents could ask for help during the survey.

If our sample do not properly represent the knowledge of the target population, the representativeness of our results may be compromised. To avoid this problem, we selected participants from two very representative Motorola test sites with respect to functional testing. Also, we chosen a sample size of 50 participants, a significant size that fitted our available resources. Our selection procedure ensured participants of all positions and all teams of the two test sites. In addition, our response rate was close to the expected value and we did follow-up procedures to ensure the survey validity.

5.7 Investigating Cost Drivers Using Designed Experiments (ES6)

The accuracy of estimation models depends on the correct choice of their input variables. Most of the software cost and effort estimation models use cost drivers, which are variables that

Table 5.18 Cost drivers identified by the survey.

Category	Cost Driver	Impact on Test Effort
Tested Product	Software stability	When the software is not stable, tests are more common to fail. When tests fail, an additional effort is required. First, the tester should ensure that he really found a software defect. Then, he must report it.
	Phone status	During the test activities, the phones may be damaged. This kind of problem occur mainly when testing very initial phone prototypes, since they may be fragile. The effort to execute tests in a damaged phone tend to increase.
	Phone performance	In general, newer phones are faster than older ones, reducing the time for executing the tests.
Test Project	Build documentation	Before the tests, it is necessary to install the correct build (software package) in the phone. This process can take longer if it is not documented properly.
	Quality of the test cases	Test cases are written in natural language. Tests written in a more standardized way are easer to be understood. For instance, the use of abbreviations, the lack of the expected results for each test step and the inconsistency with requirements make difficult to understand the tests.
Personnel	Phone familiarity	Testers must prepare and operate the phone during the tests. The familiarity of the testers with the phone functionalities (such as voice recognition and soft keys) may reduce the test execution time.
	Feature experience	Testers exercise feature functionalities during the tests. The familiarity of the testers with the feature may reduce the test execution time.
	Test experience	Testers are faster when they know the procedures and tools they should use, how to detect bugs, how to document them, etc.
	English skills	Tests are written in english. In the beginning of the tests, non english native speakers may have difficulties with specific english vocabulary.
Test Environment	Use of tools	Some of the test activities can be supported by tools. The support of tools may significantly reduce the test execution effort.
	Mobile network signal	The quality of the mobile network signal may influence the time for executing some mobile application functionalities.
	Performance to access Intranet resources	During the test activities, some Intranet resources are used (server, databases, etc.). The performance of the local network may impact the time to access these resources during the execution of tests.

describe dynamic conditions that impact team productivity, such as the team experience and the project environment. COCOMO [28] is a well-known model that addresses up to 17 cost drivers related to software development. Nevertheless, most of these cost drivers have no impact on the estimation of test execution effort because they are related to software development instead of software testing. Also, additional cost drivers that are specific to some industrial settings (tester profile, tested product, etc.) may improve the precision of test effort estimates.

In this study, we designed and run controlled experiments in the industrial setting to investigate the effect of some of the cost drivers identified in ES5. Although the identification of cost drivers presented in ES5 was based on test experts, it is important to have evidence about the significance of these cost drivers. Otherwise, test managers may waste effort to control cost drivers not really relevant and to build complex and inefficient estimation models with too many variables.

5.7.1 Research Objectives

Following the goal template of the Goal/Question/Metric (GQM) method [24], we define the research objective of this study as

> Analyze *cost drivers* for the purpose of *evaluation* with respect to *their relevance for estimating manual test execution effort* from the point of view of *test managers* in the context of *functional tests on the mobile application domain.*

5.7.2 Context

The empirical studies presented here were run using tests, testers, phone prototypes, software applications, test environments and tools available in two test sites located in Brazil, established as independent and dedicated test teams. Both test sites, called from now on as test sites A and B, provide testing services for a major mobile phone manufacturer. They have more than 300 people working on testing activities, such as test design, test execution and test automation.

In this work, we considered only teams executing component, regression and integration tests, since these were the teams more impacted by the problem we investigated.

5.7.3 Goal

The major goal of this experiment is the

- Gathering of evidence about the real impact of cost drivers (identified by the survey) on test execution effort.

The assessment of this goal is performed by answering the following research question:

Q1: For which cost drivers identified by the survey do we have statistical evidence about the significance of their effects on test execution effort?

We answer question Q1 by verifying the probability (p-value) of the main effects and low-order interaction effects of investigated factors (cost drivers) being different from 0 (zero).

Main effects of a factor are contrasts (impact on the response variable, in our case manual test execution effort) between levels (possible values, like damage and not damaged) of the factor averaged over all levels of the other factors. In addition, a factor interaction effect measures how the main effects of a factor are impacted by changing the level of other factors.

5.7.4 Hypotheses, Parameters, and Variables

In this study, we investigate the impact on test execution effort of some cost drivers identified by the survey. We defined two possible values (levels) for each investigated cost driver (factor), which are represented by the symbols – and +. Regarding the main effect of the investigated factors, we defined the following statistical hypothesis:

$$H_{0i} : \overline{\mu}_{i^-} = \overline{\mu}_{i^+}$$
$$H_{1i} : \overline{\mu}_{i^-} \neq \overline{\mu}_{i^+}$$

The null hypothesis (H_{0i}) of equality among the mean of the response variable when factor i set to "−" ($\overline{\mu}_{i^-}$) and the mean when factor i set to "+" ($\overline{\mu}_{i^+}$) says that the main effect of factor i is not significative ($\overline{\mu}_{i^+} - \overline{\mu}_{i^-} = 0$). Regarding the two-factor interactions, our null hypothesis (H_{0ij}) says that there is no interaction effect between i and j, that is, no impact in the main effect of i when changing the level of j and vice-versa:

$$H_{0ij} : (\overline{\mu}_{i^+} - \overline{\mu}_{i^-})_{j^-} = (\overline{\mu}_{i^+} - \overline{\mu}_{i^-})_{j^+} \text{ and } \quad (\overline{\mu}_{j^+} - \overline{\mu}_{j^-})_{i^-} = (\overline{\mu}_{j^+} - \overline{\mu}_{j^-})_{i^+}$$

$$H_{1ij} : (\overline{\mu}_{i^+} - \overline{\mu}_{i^-})_{j^-} \neq (\overline{\mu}_{i^+} - \overline{\mu}_{i^-})_{j^+} \text{ or } \quad (\overline{\mu}_{j^+} - \overline{\mu}_{j^-})_{i^-} \neq (\overline{\mu}_{j^+} - \overline{\mu}_{j^-})_{i^+}$$

High-order factor interactions (three or more) are not being considered in this study, because they usually have low probability to occur in practice [75].

Regarding the 13 cost drivers identified in the survey, we decided to analyze 7 of them in this study (see Table 5.19). The main reasons for not investigating all cost drivers were the following: the difficulty to control some cost drivers, such as network availability; the low frequency of some problems, such as the lack of well described tests; the already known effect of some factors from previous studies or historical database analysis, such as the phone familiarity. Test size and complexity are an exception. Although we already know they have a significant impact on the test execution effort [15], we want to investigate their interaction effect with other factors related to the tester profile.

All cost drivers identified by the survey and not investigated by our experiments are fixed to values that minimize test execution effort, reducing in this way the effort required for running our experiments.

5.7.5 Investigating cost drivers related to the tested product

A 2^7 full-factorial experiment requires at least 128 runs (test executions) to investigate the main effect and interaction effects of factors listed in Table 5.19, being costly, inefficient and impracticable to run in our context. To reduce this number of runs, we can analyze these

Table 5.19 Factors and levels investigated by the experiments.

Factor	Description	Level	
		–	+
A	SW stability	Old build	More recent build
B	HW performance	Old hardware	More recent hardware
C	HW status	Partially damaged	Not damaged
D	Testing experience	Scholarship holder	Employee
E	Language skills	English course level < 7	English course level $>= 7$
F	Feature experience	Tester not tested the feature in the last six months	Tester tested the feature in the last six months
G	Test size and complexity	Test with less than five test steps, common test actions and number of execution points* close to 200.	Test with more than 10 test steps, specific test actions and number of execution points close to 600.

* Execution point is a measure of test size and execution complexity previously proposed in [12].

factors using smaller experiments. First, we investigate the effect of cost drivers related to the product under test (hereafter factors A, B and C), because these factors are easier to control when compared to the others. The statistical significance of the other cost drivers are then investigated in a second experiment, as described in Section 5.7.6.

5.7.5.1 Experiment Design

To investigate the main effects and interaction effects of factors A, B and C in a 2^3 full-factorial design, eight runs are required, one for each possible combination of factors levels (treatment). However, we decided to run only a $\frac{1}{2}$ fraction of the 2^3 factorial structure. This fractional factorial design [75] allows us to reduce the number of runs by confounding (aliasing) some effects considered dismissible with others considered significant. If some aliased effect is considered significative, additional runs can be used to identify which one of the aliased effects is significative.

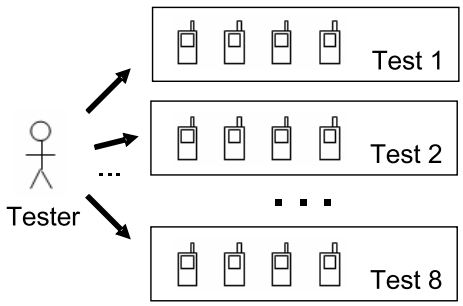
We use the principal $\frac{1}{2}$ fraction of a 2^3 factorial structure, which confounds the interactions effects with the main effects of factors A, B and C, as presented in Table 5.20. For instance, the effect of treatment c (only factor C set to "+") is confounded with the effect of treatment ab (interaction effect between factors A and B). If the effect of C is significant in the experiment, few more factors levels combinations can be run to verify if the significant effect is due to the main effect of C or due to the interaction effect between A and B. The details about how to generate the treatment design matrix can be found in [75] or in most statistical packages that support design of experiments.

In this experiment, each treatment defines the characteristics of the phone to be tested (software and hardware). Hence, we need four phone prototypes to run this experiment. Considering a single product family selected by availability of resources (different models, software releases

Table 5.20 Treatment design matrix and aliasing structure for the principal half fraction of a 2^3.

Treatment	Factor A (stability)	B (perf.)	C (status)	Alias
c	–	–	+	ab
a	+	–	–	bc
b	–	+	–	ac
abc	+	+	+	

Figure 5.1 Randomized complete block design on tested products.

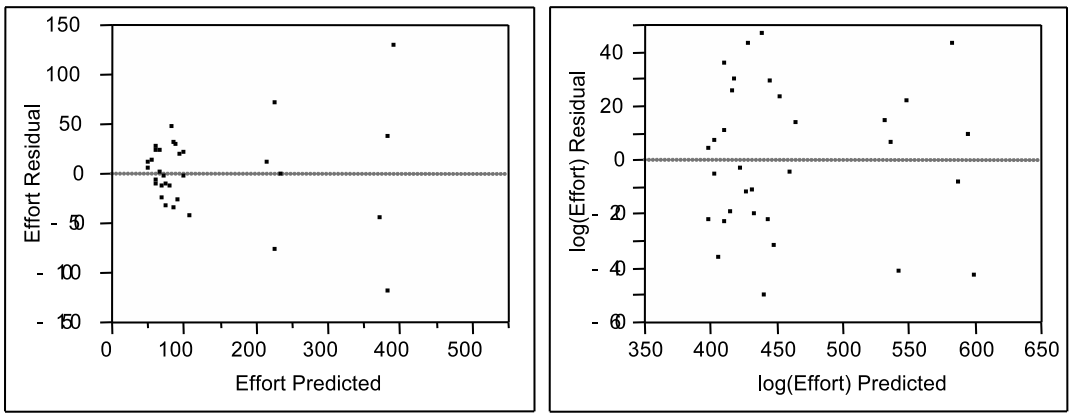


and phone prototypes, experienced testers, etc.), a pair of phone prototypes is selected from an old model and other pair is selected from a more recent model with a newer hardware (factor B). Each pair must have one damaged phone (factor C), which is a phone with some problem in the keyboard or display caused by its intense use during the test phases or during physical tests (drop tests, etc.). Finally, a more recent or old software build is installed in each phone (factor A), according to the treatments defined in Table 5.20.

To control the effect of the cost drivers related to the tester profile, a single tester manually executes a set of tests four times, each time using a different mobile phone (treatment). We estimate the number of tests to be executed based on the time available to run the experiment. As we know that the size of the tests impact the test execution time [15], each test is considered as a block in this experiment, as presented in Figure 5.1.

The run order of the tests within each block is randomly defined, as well as the definition of which test (block) is executed first. This randomization process reduces the learning effect that may occur between test executions. All other cost drivers are set in a way to minimize the test execution time. For instance, the selected tester must have experience with the tests and the tested feature.

Figure 5.2 Variance of residuals versus predicted before and after log transformation.



5.7.5.2 Execution and analysis

The experiment was run according to the planning and the results are presented in Table 5.21. The average effort to execute a test was 131.8 seconds. The table also shows the average effort per treatment and per block, as well as the observed treatments and block effects. Although the block effect on test execution effort is clearly significant when compared to the grand average or the residuals, it is not clear the significance of the treatments effects or the significance of the factors effects.

We used analysis of variance (ANOVA) [75] to test our hypotheses that the main effects of the investigated factors are dismissible. The residuals of the model built by the ANOVA using the raw data are presented in Table 5.19. During the validation of the model assumptions, we detected a heterogenic variance of the residuals against the predicted values. As presented in Figure 5.2, a transformation of the response variable [53] based on the log stabilized the variance of the residuals against the predicted values. After the data transformation, we also verified the normality of the residuals using the Shapiro-Wilk W test (p-value=0.6211) and the additivity of treatments and block effects using the Tukey's test of nonadditivity [131] (p-value=0.1154) at α=0.05.

The results of the ANOVA using the transformed data are presented in Table 5.22. The last column of the table (p-values) can be respectively interpreted as the probability of rejecting H_{0_A}, H_{0_B} and H_{0_C} when they are true. Hence, it is not possible to reject these null hypotheses at α=0.05, that is, the main effect of factors A (Software stability), B (Hardware performance) and C (Hardware status) cannot be considered significative. We also cannot reject the null hypotheses $H_{0_{AB}}$, $H_{0_{AC}}$ and $H_{0_{BC}}$, which say that the effects of two-factor interactions are not significative, since at least one factor must be significative to occur an interaction effect (hierarchy principle [75]).

Finally, comments about possible reasons of not rejecting the null hypotheses are presented later, in Section 5.7.7.

Table 5.21 Results in seconds from the randomized block design on tested products*.

	treatment (phone)				block	block	residuals for treatment			
block	c	a	b	abc	average	effect	c	a	b	abc
test 7	$53^{(4)}$	$56^{(2)}$	$82^{(1)}$	$48^{(3)}$	59.8	-72.1	$3.2^{(4)}$	$-13.0^{(2)}$	$22.0^{(1)}$	$-12.2^{(3)}$
test 2	$66^{(3)}$	$63^{(4)}$	$67^{(2)}$	$127^{(1)}$	80.8	-51.1	$-4.8^{(3)}$	$-27.0^{(4)}$	$-14.0^{(2)}$	$45.8^{(1)}$
test 5	$326^{(3)}$	$519^{(1)}$	$261^{(2)}$	$418^{(4)}$	381.0	249.2	$-45.0^{(3)}$	$128.7^{(1)}$	$-120.3^{(2)}$	$36.6^{(4)}$
test 3	$116^{(1)}$	$64^{(2)}$	$119^{(3)}$	$94^{(4)}$	98.3	-33.6	$27.7^{(1)}$	$-43.5^{(2)}$	$20.5^{(3)}$	$-4.7^{(4)}$
test 4	$60^{(2)}$	$43^{(4)}$	$52^{(3)}$	$86^{(1)}$	60.3	-71.6	$9.7^{(2)}$	$-26.5^{(4)}$	$-8.5^{(3)}$	$25.3^{(1)}$
test 8	$225^{(2)}$	$233^{(4)}$	$296^{(1)}$	$149^{(3)}$	225.8	93.9	$9.2^{(2)}$	$-2.0^{(4)}$	$70.0^{(1)}$	$-77.2^{(3)}$
test 6	$62^{(3)}$	$111^{(1)}$	$114^{(2)}$	$49^{(4)}$	84.0	-47.8	$-12.0^{(3)}$	$17.7^{(1)}$	$29.7^{(2)}$	$-35.4^{(4)}$
test 1	$67^{(1)}$	$40^{(2)}$	$66^{(4)}$	$87^{(3)}$	65.0	-66.8	$12.0^{(1)}$	$-34.3^{(2)}$	$0.7^{(4)}$	$21.6^{(3)}$
treatment average	121.9	141.1	132.1	132.3	131.8 = grand average					
treatment effect	-10.0	9.3	0.3	0.4						

* The superscripts in parentheses associated with the observations indicate the random order in which the experiment were run within each block. Each block was executed according to the order of presentation (up to down).

Table 5.22 Effect tests for the factors SW stability, HW performance and HW status.

Effect Tests

Source	DF	Sum of Squares	Mean Square	F Ratio	Prob > F
SW stability	1	1.11	1.11	0.0010	0.9746
HW performance	1	1128.20	1128.20	1.0597	0.3150
HW status	1	182.93	182.93	0.1718	0.6827
Block	7	129743.08	18534.72		
Error	21	22358.09	1064.70		
C. Total	31	153413.41			

5.7.6 Investigating cost drivers related to the tester profile

Our second experiment has the objective of investigating the cost drivers related to the testers profile (factors D to F) and their interaction with the test size and complexity (factor G). The details of the experiment are presented next.

5.7.6.1 Experiment design

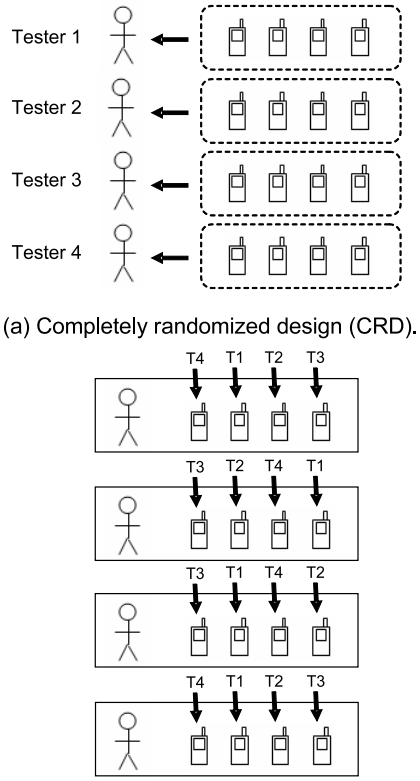
In a 2^4 full-factorial design, we have 16 different treatments. Each of these treatments requires the use of a different tester, since some of his or her characteristics are under investigation. We can reduce to 8 treatments by running only a $\frac{1}{2}$ fraction. In such design, however, each two-factor interaction is confounded with some other two-factor interaction, which would probably require additional runs to resolve ambiguities of aliased effects [141]. In addition, we cannot select any tester to participate in the experiment, but only those that match some specific profiles. We verified that we would take longer to have 8 testers available with such specific profiles.

Since the run of a $\frac{1}{4}$ fraction of a 2^4 is not used in practice (too many confounded effects), we use a more elaborated experiment design, called split-plot , to reduce the number of required testers to four. A split-plot design can be described as the superimposing of one experiment design on top of another [52] [99]. Figure 5.3 shows the split-plot design for our experiment.

In our split-plot design, the first experiment design is a complete randomized design (CRD) [75]. In this CRD, factors D and E are combined to define the levels of a factor tester profile, as we can see in the treatment design matrix for this split-plot design presented in Table 5.23. These tester profiles are the treatments used to select four testers (*whole-plots*), who are assigned to test a group of phones. All considered phones are similar and assigned to each tester at random. The execution order inside each group of phones is also defined at random.

The second experiment design is a randomized complete block design (RCBD) [75]. In this RCBD, factors F and G are combined to define the levels of a factor test characteristics, which is used to select four tests to be executed by each tester (block). For each block, we randomly allocate a test to a different phone (subplot). In this way, our split-plot design can be described as a completely randomized design on top of a randomized complete block design. We also replicate our experiment by changing the tests and using the same testers to execute the new

Figure 5.3 Superimposing a CRD on top of a RDBD (split-plot design).



tests.

With respect to the experimental units, all phone prototypes are from the same model, with the same or at least similar hardware and software versions. The testers that match the investigated profiles (factors D and E) are selected by availability. Then, for each replicate, an pair of tests is selected from a feature that all testers have recently tested and other pair of tests is selected from a feature that all testers have not tested it in the last six months (factor F). From each pair of tests, one test must be small and simple and the other should be more complex (see definition of factor G in Table 5.19).

5.7.6.2 Execution and analysis

The experiment was run according to the planning and the results are presented in Table 5.24. Based on the analysis of variance presented in Table 5.25 we can only support the hypotheses that the effect of factors testing experience and test size and complexity are significant (H_{0_F} and H_{0_G}). No statistically significant effect was detected for all other individual factors and factor interactions investigated by the experiment.

In this way, we cannot reject the remaining null hypotheses (H_{0_D}, H_{0_E}, $H_{0_{DE}}$, $H_{0_{FG}}$, ...), which say that the effects of factors testing experience (D), english skills (E) and all two-factor

Table 5.23 Treatment design matrix of the split-plot design.

Factor tester profile used as in a CRD		
	Factorial structure	
	D	E
Treatment	(Testing exp.)	(Language skills)
(1)	−	−
d	+	−
e	−	+
de	+	+
Factor test characteristics used as in a RCBD		
	Factorial structure	
	F	G
Treatment	(Feature exp.)	(Test size)
(1)	−	−
f	+	−
g	−	+
fg	+	+

interactions between factors D, E, F and G are not significative. Our interpretation of this result is presented next, in Section 5.7.7. Also, details about the construction of this specific ANOVA table, which has two error terms, can be found in [52].

5.7.7 Discussion about the experiments results

In this work, we designed experiments using 2^k factorial structures to investigate the significance of cost drivers identified by the survey using the limited resouces we had available. In the first experiment, we could not reject the null hypotheses. Afte analyzing the results, we reach the conclusions described next.

The factor software stability affected the number of tests that had failed or passed, but the difference in the effort required to execute a test when it fail or pass was not statistically significant. This factor is probably significant to estimate the effort of some testing activities not considered in this study, such as reporting and monitoring detected defects.

By observing test executions, we verified that the average time spent by the tester to read and to manipulate the phone is higher than the average phone response time for processing the tester commands. This is probably the main reason for not rejecting the null hypothesis for factor hardware performance. Hence, this factor may be significant for tests executed automatically or for tests that require higher cpu processing time.

Although we did not detect significant effect of damaged phone prototypes on test execution effort, we know that a too damaged prototype can impact or even block the test execution.

The factor testing experience was not considered significant. We considered employees and scholarship holders as the levels of testing experience. We observed that one can get experience

Table 5.24 Results from the replicated split-plot experiment*.

tester	test (1)	test f	test g	test fg	tester average
(1)	$221^{(4)}$	$268^{(1)}$	$467^{(2)}$	$409^{(3)}$	341.3
	$241^{(3)}$	$626^{(2)}$	$97^{(1)}$	$471^{(4)}$	358.8
d	$288^{(3)}$	$196^{(2)}$	$337^{(4)}$	$311^{(1)}$	283.0
	$183^{(2)}$	$589^{(4)}$	$126^{(3)}$	$731^{(1)}$	407.3
e	$180^{(3)}$	$103^{(1)}$	$528^{(4)}$	$239^{(2)}$	262.5
	$240^{(3)}$	$179^{(1)}$	$120^{(4)}$	$436^{(2)}$	243.8
de	$181^{(4)}$	$114^{(1)}$	$553^{(2)}$	$301^{(3)}$	287.3
	$142^{(2)}$	$299^{(3)}$	$294^{(1)}$	$616^{(4)}$	337.8
test average	217.5	170.3	471.3	315.0	293.5
	201.5	423.3	159.3	563.5	336.9

* The superscripts in parentheses associated with the data indicate the random order in which tests were executed by each tester. For each tester, there is one row for each replicate.

in executing tests manually in few months. As the scholarship holders have approximately 6 months of work in testing, they may already have enough experience in executing tests. This result cannot be generalized for other testing activities.

During the execution of the experiments, no one tester reported problems to understand the test specification written in a foreing language (english), which is in conformance with the results: factor language skills not significative. The language used in test specifications are usually simple, diminishing the effect of this factor. This may not be true for other testing activities, mainly those involving communication with foreigners.

When we cannot reject a null hypothesis, it is interesting to know the power of the test, that is, the probability to reject the null hypothesis when the null hypothesis is false. To calculate the power of a test, we need to define the level of significance (α), the detectable effect size (δ) and the standard error (σ) of the residual error in the model. Although we can estimate the value of σ based on the observed data (σ=32.6293), it is not correct to make retrospective power analysis to support our results [54], but only for planning future experiments. For instance, as presented in Figure 5.4, the use of 32 experimental units would be appropriate for δ=20 or 30 to aim at a power of 0.9.

5.7.7.1 Threats to Validity

In this section, we describe threats to the validity of the experiments results, the impact of each threat and how we controlled them.

Construct validity: For the factors *language skills* and *test size and complexity*, their categorization into two levels ("-" and "+") is not intuitive. Also, the significance of the factor can depend on the correct factor categorization. We used expert opinion and historical data to categorize these factors according to its expected impact on test execution productivity (low or

Table 5.25 Effect tests for the factors D, E, F, G and their interactions.

Effect Tests

Source	DF	Sum of Squares	Mean Square	F Rati	Prob > F
Rep	1	15051.1250	15051.1250		
Testing experience (D)	1	5940.5000	5940.5000	0.80	0.4367
Language skills (E)	1	33540.5000	33540.5000	4.52	0.1234
D x E	1	8256.1250	8256.1250	1.11	0.3688
Error (tester profile)	3	22241.1250	7413.7083		
Feature experience (F)	1	81204.5000	81204.5000	6.72	0.0291*
Test size and comp. (G)	1	556512.5000	556512.5000	46.08	<.0001*
F x G	1	12720.1250	12720.1250	1.05	0.3315
F x Rep	1	8.0000	8.0000	0.00	0.9800
G x Rep	1	33282.0000	33282.0000	2.76	0.1313
F x G x Rep	1	1711.1250	1711.1250	0.14	0.7153
D x F	1	4278.1250	4278.1250	0.35	0.5664
D x G	1	1326.1250	1326.1250	0.11	0.7479
D x F x G	1	3362.0000	3362.0000	0.28	0.6105
E x F	1	45451.1250	45451.1250	3.76	0.0843
E x G	1	5565.1250	5565.1250	0.46	0.5143
E x F x G	1	18050.0000	18050.0000	1.49	0.2525
D x E x F	1	40.5000	40.5000	0.00	0.9551
D X E x G	1	684.5000	684.5000	0.06	0.8172
D x E x F x G	1	3321.1250	3321.1250	0.28	0.6127
Error (test charact.)	9	108686.6250	12076.2917		
C. Total	31	961232.8750			

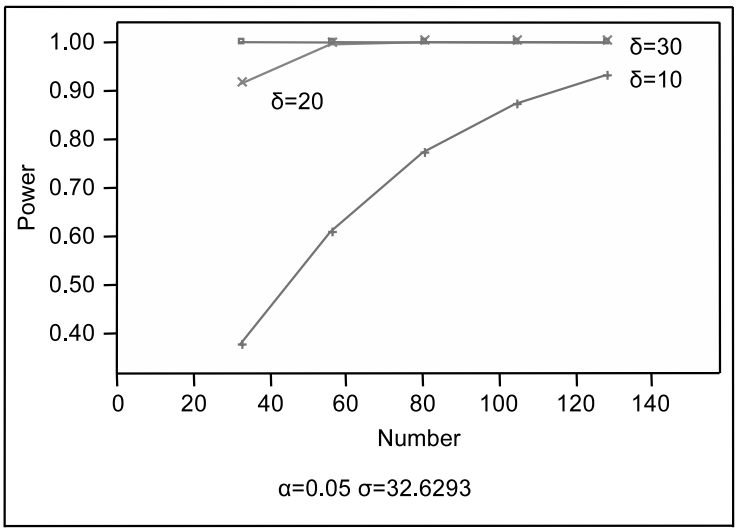
high).

Internal validity: In the second experiment, we involved more than one tester. The effect of each tester was controlled by the investigated factors D, E and F. However, many other factors (culture, education, etc.) may affect the behavior of a tester. These factors may have significant impact on test execution effort, obscuring the effects of the investigated factors. Also, we noticed differences in the test execution times between the replicates. Some learning effect and other related problem may be increasing the variance of the residuals, reducing our capacity to detect significant effects. We control this threat by considering the effect of the replicates in the analysis model.

Conclusion validity: We used an elaborated experiment design (strip-plot) with a replicate that require a special statistical analysis not directly supported by statistical packages. The participation of an statistician improved the reliability of the analysis of the results.

External validity: The participants and material used in the experiments limit the generalization of our findings. All testers work at the test sites and were selected by availability. The use of more testers and different tests may make significant factors not detected as significant in our experiment.

Figure 5.4 Statistical power for different effect sizes and number of observations.



5.7.8 Final Considerations

In this work, we used designed experiments and 2^k factorial structures to investigate the significance of cost drivers identified by a survey in an industrial setting. The survey identified 13 cost drivers and we investigated those related to the tested products and testers profile. Different from experts opinions, only two cost drivers had significative effect.

We see that not all cost drivers suggested by experts have significant effect on test execution effort. With our results, test managers can focus their attention on the cost drivers that had significant impact on manual test execution effort. Also, this reduced number of cost drivers to consider implies in the creation of simpler estimation models.

We used 2^k factorial structures and design of experiments to overcome the lack of resources found in the industrial setting to run experiments. We could reduced the number of required resources by $\frac{1}{2}$ in the first experiment and by $\frac{1}{4}$ in the second experiment.

5.8 Summary of Empirical Results

In this section, we analyze how our empirical studies answered our research questions and supported or rejected our research hypotheses.

RQ1.1: What are the relevant characteristics that should be considered when sizing a test specification?

This question was answered in ES2, where experts identified the list of characteristics that should be considered when sizing test specifications. Some of

these characteristics are specific to particular features. Only four of them were applicable to the feature used in ES4. In addition, we used statistical methods in ES4 to verify the significance of these four characteristics. One of them was high correlated to the others and was discarded.

RQ1.2: What is the weight of each relevant characteristic with respect to test execution effort?

This question was also answered in ES2, where experts identified the weights of the characteristics considered relevant for sizing test specifications. In addition, we used statistical methods in ES4 to automatically calculate the weight of these characteristics based on historical data.

RQ1.3: Can we define a soundness measure for representing test size and execution complexity?

According to our empirical results in ES2, Execution Points is a measure of test size and execution complexity that is soundness and that can be used in practice.

RQ1.4: Can the measurement method be automated?

For running ES2, we developed a tool prototype [20] (see Appendix C) that implemented the execution points measurement method, showing that is possible to automate all steps of the measurement method.

RQ1.5: What are the relevant cost drivers for estimating test execution effort?

To answer this research question, we first use expert opinion in ES1, identifying 12 possible cost drivers related to manual test execution. To confirm the relevance of these cost drivers and to identify others, we applied a survey (ES5), resulting in 13 possible cost drivers that include most of the ones found in ES1. In ES6, we tryied to confirm the relevance of seven of these cost drivers, but only the significance of test size and tester experience were statistically confimed.

RQ1.6: What is the impact on test execution effort (weight) of each each cost driver?

In ES1, experts assigned weights to 12 possible cost drivers related to manual test execution. In ES6, the weight of the cost drivers considered statistically significant are determined based on their effects observed in the experiment.

RQ2: Is the estimation accuracy of test execution effort improved by regarding the size and execution complexity of test specifications?

Studies ES2, ES3 and ES4 presented accuracy improvements (with statistical significance) by considering the size and execution complexity.

RQ3: Is the estimation accuracy of test execution effort improved by regarding cost drivers related to manual test execution?

In ES6, some of the investigated cost drivers presented statistically significant effect on test execution effort. In addition, the factor representing the tester experience was considered statistically significant in ES4. In this way, we can conclude that we can improve estimation accuracy by considering these relevant cost drivers.

RQ4: What is the cost for estimating manual test execution effort based on the size of test specifications and cost drivers related to manual test execution?

As we showed that the measurement method can be automated (RQ1.4), the cost for using the model is basically the cost for configure the measurement method, that is, the cost for identifying the characteristics and cost drivers to consider and the cost for determining their weights. In ES1, we had the cost of four hours (two Delphi sessions) to configure the model using six experts from the mobile application domain. We believe that similar cost will be necessary for other application domains.

Based on the answers we presented for research questions RQ1.1 to RQ1.6, we can conclude that it is possible to determine the size and execution complexity of a test specification by observing characteristics of each test action in the specification. This result answers our more general research question RQ1 (Can we measure the size and execution complexity of a test specification?) and supports our research hypothesis **RH1**.

In addition, our answer for research question RQ2 supports our research hypothesis **RH2**, which says that the use of a measure for test size and execution complexity will significantly improve the accuracy of test execution effort estimates. Similarly, our answer for research question RQ3 supports our research hypothesis **RH3**, which says that the use of cost drivers related to test execution will significantly improve the accuracy of test execution effort estimates.

Finally, the answers for research questions RQ1.4 and R4 support our research hypothesis **RH4**, which says that it is possible to automate test execution effort estimation based on test specifications, leading to a low estimation cost.

5.9 Final Considerations

In this chapter, we presented the details of planning, execution, analysis and results of six empirical studies. Each one of these studies was run to answer one or more research questions. The whole sequence of studies produced enough information to answer all of our research questions and to support all of our research hyphoteses.

Since we presented the details of our research in the previous chapter and the results from our empirical studies in this chapter, the next chapter relates this work to others found in literature and in industry.

CHAPTER 6

Related Work

In this research, we proposed and evaluated a measure, called Execution Points, and its measurement method. We also proposed estimation models for test execution effort based on this measure. In addition, we run several empirical studies to configure the Execution Points (EP) measurement method, evaluated and compared its adequacy for estimating test execution effort. In this chapter, we relate all these works with others found in literature.

6.1 Size Measures and Measurement Methods

In Chapter 2, we presented the characteristics of several size measures and measurement methods. Table 6.1 recalls the main characteristics of these existing size measurement methods and summarizes the main characteristics of the EP measurement method. EP, number of tests and number of test steps are the only measures we know that can be used to size test suites and test specifications. This is an important characteristic for our context, where we have to estimate the effort to execute a subset of the tests designed for a feature that are selected according to what we need to test (parts that were created, modified or possibly impacted by changes done in the code of a feature).

Some of the other size measurement methods can be calibrated using local data, but this procedure is optional and difficulty in some cases. The EP measurement method must be configured according to each application domain. We defined and evaluated procedures for that based on expert judgement and regression analysis. The EP measurement method and its configuration procedures are automated by supporting tools, reducing the cost to use this approach.

With respect to the validity of the EP measurement method, we verified that some of the critiques to Function Points Analysis (FPA) are valid to our approach, but they are not valid when we configure the EP measurement method without using regression analysis. For instance, the weights used in most versions of FPA were defined using expert judgement. Hence, subjectivity is considered a weaknesses of the FPA weights system. The weights used in the EP measurement method can be defined using expert judgement through Delphi panel (a subjective method) or using historical data through regression analysis, which is not a subjective method. In fact, some researchers already proposed the use of non subjective methods to determine the weights system of FPA. For example, the authors of [6] established a new FP weights system by using Artificial Neural Networks. They reported accurate results when the new FP weight system was used to estimate development effort based on real data sets of software projects.

There are some works in literature that proposed estimation models for specific application

domains, requiring the definition of new size measures. Regarding the development of Web applications, there are proposals of how to better measure the size of this kind of software projects [86]. Our research and these others are all based on the use of surveys, expert judgment and statistical techniques.

Finally, Test Point Analysis [105] is a method for measuring the volume of testing of a project. This includes all functional test activities, such as the definition, automation and execution of all tests, being not appropriated in situations such as:

- Companies having different test teams, some of them responsible to design and others to execute tests. In this situation, each different test team needs to estimate the effort required to perform their activities (for instance, only test design or only test execution).

- When it is not possible or required to execute all test cases due to time constraints or optmization techniques, it is important to estimate the effort to execute only a selected subset of tests.

For example, it is not possible to use the Test Point measure for estimating only the effort to execute test cases that were automatically generated by a Model-based testing tool [46], since this measure is related to requirements instead of test specifications. Using EP, we can estimate the effort required only to execute tests, which is apropriated to the context of this research: dedicated and independent test execution teams working with the possibility to automatically generate tests.

6.2 Adequacy of Size Measures for Estimating Effort

In ES4 (see Section 5.5), we compared the adequacy of four size measures to estimate test execution effort. We observed statistically significant differences between the estimation accuracies achieved by models based on these measures and created by linear regression analysis. In literature, we can find several studies comparing size measures using similar approach. In [96], the authors compared groups of Web-based size measures (lenght, complexity and functionality metrics) used to estimate Web design and authoring effort. For that, the authors used linear and stepwise regression analyses to create estimation models and compare their estimation accuracy. Their empirical results revealed that “none of the obtained models produced reasonable accurate estimates of the effort", and “the models did not produce significantly different residual values”.

In [65], the authors used regression analysis to verify the correlation between two different function points counting techniques and to compare the accuracy achieved when using each technique. Also, Ruhe et al. used regression analysis to compare Web Objects and Function Points (FPs) measures with respect to their adequacy to estimate the effort to develop Web applications [122]. The results of the empirical analysis revealed that the model based on Web Objects presented significantly better prediction accuracy, which also support our idea that we can improve estimation accuracy by considering the characteristics particular to each different application domain.

Table 6.1 Summary of EP and other software size measurement methods.

	SLOC	FPA	COSMIC	TPA	No. of Tests	No. of steps	EP
Artifact	Source code	Requirements	Requirements	Requirements	Test suite	Test specification	Test specification
Attribute	Application size and complexity	Application size and complexity	Application size and complexity	Volume of test	Test suite size	Test size	Test size and execution complexity
Restriction	Language specific			Requirements written as use cases			
Availability	After implementation	After requirement specification	After requirement specification	After requirement specification	After test specification	After test specification	After test specification
Standardized	There is some work in progress	Yes	Yes	Yes	Yes	Yes	Yes
Deterministic	Yes	No, result depends on the expertise in the measurement method	No, result depends on the expertise in the measurement method	No, result depends on the expertise in the measurement method	Yes	Yes	Yes
Measurement cost	None for languages supported by tools	High, since there is a lot of manual effort	Very high, since there is a lot of manual effort and it is more abstract method	High, since there is a lot of manual effort	None when supported by tools	None when supported by tools	Low, since we need to manually rate each new test action
Calibration cost	Not required	Significant, but not required	Significant, but not required	Significant, but not required	Not required	Not required	Low, but required
Required backgound	None when supported by tools	Knowledge in the measurement method and measured application	Knowledge in the measurement method and measured application	Knowledge in the measurement method and measured application	None	None	Knowledge to rate new test actions
Automation	Yes	No	No	No	Yes	Yes	Yes
Validity	Same application can be implemented with significant different SLOC, depending on the language, developer, etc.	There are several critiques (see Section 2.3.1)	Result depends on who is measuring	Problems similar to FPA	Tests can have significantly different sizes and complexity	Steps can have significantly different execution complexities. Same test can have different measures depending on the level of detail used by the test designer	Some critiques of FPA are valid for EP measurement calibrated by expert judgement, but not when calibrated by regression analysis

In [86], the authors compared four sets of size measures (Length, Web Objects, Functional and Tukutuku measures) using not only regression analysis (Forward Stepwise Regression), but also Case-Based Reasoning. All the measures provided good predictions in terms of MMRE, MdMRE, and Pred(0.25) statistics, for both FSR and CBR. Moreover, when using SWR, Length measures and Web Objects gave significant better results than Functional measures, however presented similar results to the Tukutuku measures [3]. As for CBR, results did not show any significant differences amongst the four sets of size measures.

6.3 Estimation Techniques

The existing algorithmic estimation models, such as COCOMO [28] and FPA-based approaches [49], are usually based on the size of the software under development and their effort estimates are for the whole development cycle (with some breakdown). For this reason, it is difficult to estimate the effort required to execute tests. Our approach is based on EP, a measure of test size and execution complexity, which allows us to estimate the effort required to execute tests based on their specifications. One additional advantage of this characteristics is that sometimes the requirements are not available (not written or outdated). However, the test specifications are more likely to be documented and updated.

Since it is not appropriate to use the existing models in our context, we presented the use of three different approaches to estimate test execution effort: productivity-based, regression-based and COCOMO-like approaches. The use of these approaches can be found in several works, such as [49], [28], [30], [105], [98], [91] and others. In addition to these approaches, several other techniques can be used to estimate test execution effort based on EP, such as robust regression [53], Bayesian networks [106] and models based on machine learning, such as CBR [139], regression trees [119] and neural networks [51]. Next, we highlight some works that applied these other techniques.

Costagliola et al. compared Web-based Length and Functional measures using both Stepwise Linear Regression, Regression Trees and Case-Based Reasoning [43]. Their empirical results suggest that Length measures provide best accuracy for estimating software development effort. Briand et al. also applied Stepwise Linear Regression, Regression Trees and Case-Based Reasoning in [33] and [34]. Their results point out that Linear Regression and Regression Trees are better than Case-Based Reasoning in predicting effort, and in general RT provides the best results in terms of MMRE and Pred(25).

Mendes has several publications ([93], [95], etc.) and a book [91] about creating models for estimating Web development effort. She investigated the use of well-known techniques, such as Case-Based Reasoning (CBR), Stepwise Regression (SWR) and Classification and Regression Trees (CART), obtaining interesting accuracy results [98]. She used several measures instead a single size measure in order to avoid scale and transformation problems. In ES4, we also compared the use of EP with the use of multiple measures. However, we observed a slightly better result when using EP.

6.4 Identification of Cost Drivers and Model Calibration

In our research, we used a Delphi panel and a survey to identify possible cost drivers for manual test execution. Then, we designed experiments to investigate the effect of some of these cost drivers on the effort to manually execute tests. In literature, we can find many related works that identified cost drivers for software development. To identify the cost drivers used in COCOMO II [28], the authors used expert judgement through Wideband Delphi panels. To define the effect of the identified cost drivers, they used a Bayesian network to combine effect estimates based on expert judgment (Wideband Delphi panel) and historical data (regression analysis). The combination of these effect estimates produced a more accurate and generalizable model calibration.

Another related work was developed by Mendes et al. [97]. They identified several size metrics and cost drivers for early Web cost estimation based on current practices of several Web Companies worldwide. For that, they applied two surveys and a case study using a mature Web company. The results of these activities were used to gather data on Web projects. Finally, regression and other statistical analyses were performed to estimate the effect of cost drivers and size measures. They detected statistically significant effect for few size measures, but not for the cost drivers. During our work, we also investigated the effect of cost drivers using historical data (see B), but the effect of confounding factors could not be controlled.

6.5 Measurement Method Automation

Many researchers highlighted the need to automate the measurement of size measures. In [29], Boehm and Malik highlighted the need for automating the counting processes. In [96], the authors confirmed the need for objective size measures collected automatically whenever possible. In this research, we developed an automated EP measurement method that process test specifications written in natural language. For using our measurement method, it is only necessary to configure the method and to rate the first occurrence of each possible test action (verb), reducing the costs of our approach.

There are several other works that automate measurement methods. In [44], the authors propose a formalization of the IFPUG Function Point (FP) definition for automated measurement of the software requirements specified in a formal language called B [77]. In addition to the reduction of measurement costs, they identified specific holes in the IFPUG FP definition and proposed modifications to ensure completeness. In [142], Zivkovic et al. proposed a mapping of UML models into function points, enabling an automation of the counting procedure. Also, Levesque et al. proposed a method for automating the measurement of COSMIC-FFP from UML [79].

CHAPTER 7

Conclusions

This work focus in a relevant problem for industry and academia. In industry, good planning is essential and it depends on how good estimates are. In general, the consequences of poor estimates are scope reduction, schedule overrun and overtime, which increase costs and might reduce quality.

For academia, even after several decades of research in estimation models, there are several studies being performed to achieve accurate results. Some examples are the creation of domain-specific models [98] and the analysis of cross-company and within-company effort estimation models [94]. In addition, testing is a fundamental activity for the industry involving several research groups [136].

This chapter summarizes the contributions of our work, presenting their impacts and limitations, showing lessons learned from this research, and presenting future works.

7.1 Summary of Contributions

The main contribution of this research is the proposal and evaluation of a measure for test size and execution complexity. This measure, called Execution Points, is based on test specifications written in controlled or standardized natural language. This characteristic better supports the estimation of the effort to execute any given set or subset of functional tests. The use of specifications written in natural language also makes easier further adoption of the proposed approach by industry, a usual challenge for an academic research.

Also, our test size measure supports a better capacity specification and testers productivity comparison [13]. For instance, it is difficulty to evaluate team capacity or tester productivity only based on the number of tests they can execute per day, since tests may vary a lot in size and complexity. However, we can better state manual test execution capacity and compare manual test execution productivity by using execution points, since it is a generic unit of work.

We measure execution points by analyzing each sentence (step or test action) found in test specifications according to a set of characteristics. In this measurement method, the evaluations of similar test actions are reused, since we observed that their execution complexity can be determined by analyzing only the main verb of each sentence. Also, it is possible to automate all steps of this measurement method, except the evaluation of the first occurrence of each different test action (verb). With this automation support, the number of necessary manual evaluations of test actions tends to decrease over the time. All these optimizations significantly reduce the costs for using our model.

Another contribution of this work is the proposal and evaluation of estimation approaches

for manual test execution effort based on test specifications. After being configured and calibrated, our estimation approaches (regression-based, productivity-based, etc.) do not require previous execution times of the tests to be executed, since they are based on the characteristics of test specifications. This characteristic is extremely important in several situations, such as when tests are new (never executed before), when they are very different from the existing ones, or when we do not have reliable historical data about these tests to be executed.

We planned our research in terms of main goals, questions and hypotheses. This planning guided not only the research development, but also its evaluation through a sequence of six empirical studies on the mobile application domain. These studies were based on the most used techniques in the experimental software engineering field, such as experiments, surveys, case studies, expert judgments and statistics. These studies helped us to generate more reliable results, which answered our research questions and supported our research hypotheses.

In summary, the empirical study ES1 used expert judgment through a Delphi panel to create a first version of the test execution effort estimation model. The empirical study ES2 was a case study where we observed a statistical significant accuracy improvement when using execution points. We also showed empirically the soundness of execution points and the automation of its measurement method. The empirical study ES3 observed an accuracy improvement when using execution points to analyze a large historical database. The empirical study ES4 was a Montecarlo Experiment that collected test execution data from different testers in a controlled environment, verifying that execution points was the more adequated test size measure for estimating test execution effort. The survey ES5 identified 13 possible cost drivers related to manual test execution through the application of a questionnaire to several testers with different profiles and positions. Finally, ES6 was structured as two experiments that investigated the effect on test execution effort caused by cost drivers identified in ES5, suggesting that only few of them are really relevant for estimating test execution effort.

As detailed in Section 5.8, all these results supported our research hypotheses in the following way:

- We found evidences of the validity of execution points, a measure for test size and execution complexity based on the characteristics of each test action found in the test specifications.

- The use of a measure for test size and execution complexity (execution points) significantly improved the estimation accuracy of manual test execution effort.

- Some cost drivers related to manual test execution (tester experience, test size, etc.) significantly improved the estimation accuracy of manual test execution effort.

- We automated the measurement of execution points, reducing the costs of its use.

An additional contribution of our empirical studies is the presentation of how to configure and calibrate measurement methods and estimation models by using expert opinion or data analysis. These studies also show how to compare size measures and how to identify and investigate the effect of cost drivers in industrial settings.

Some other additional contributions of this work are the review of existing software effort estimation models and software size measurements methods, as well as the development of a test execution effort estimation tool and the ManualTEST, a tool that resulted in significant quality improvements in collecting data during experiments and case studies related to manual test execution.

7.2 Impact and Limitations

This section describes the most important findings of this research with respect to their potential impact in industry, emphasizing potential impacts on cost, time, and quality, as well as the principal limitations of this work.

The cost to introduce the use of execution points for estimating test execution effort in an software organization is small, since we can automatically process test specifications written in an controlled or standardized natural language. Basically, we have to configure the execution points measurement method according to the characteristics relevant to the considered application domain. According to the results of ES1, the method configuration can cost up to four hours of five to seven experienced testers in a Delphi panel. We can also calibrate the measurement method (define weights) using historical data, which can takes few more hours of an experienced data analyst. This cost is very small when considering possible overtime reductions of large test execution teams.

The results of this work also have impact on the time of planning test activities. These activities can be performed several times per month, since test projects are usually shorter in time than software development projects. In practice, experienced testers have to analyze test specifications or to consider historical test execution times to estimate test execution effort. This activity can takes few hours of more than one tester. In our approach, estimates of test execution effort are automatically performed by the tool we developed.

As our estimation approach improves the accuracy of test execution effort estimates, it also has impact on software quality. By avoiding the problems of having poor estimates, such as scope reduction, we decrease the probability of having tests not executed due to time restrictions.

Despite of the benefits of our estimation approach that were observed in our empirical studies, they depend on the appropriate configuration of the execution points measurement method, that is, the proper selection of characteristics and weights. In addition, it is necessary to properly evaluate all test actions (verbs) according to the set of characteristics considered relevant. Although the verbs appear to be good test action identifiers in most cases, we did not investigated in which cases the verb arguments should also be used to improve accuracy. In addition, we have to deal with several different situations when processing test specifications written in a standardized natural language: when verbs are used in different sentences with different meanings, we have to consider the verb arguments or the use of an additional verb to eliminate the ambiguity; when different verbs are used with the same meaning (synonyms), we should select only one verb and eliminate the others or consider all of them as the same to keep consistency; general sentences that represent more than one test action should be rewritten in a high level of detail (more sentences); etc.

Also, we did not consider in our studies the variation that can occur when changing the group of experts used to define test characteristics, guidelines, weights, etc. As a consequence, we do not know if this variation can significantly change the observed results. Finally, if we consider different application domains, we have to run again: empirical studies E1 and E5 (see Table 5.1), to identify the relevant test characteristics and cost drivers related to test execution, as well as their impact (weight) based on expert opinion; empirical studies similar to E3 and E6, to evaluate the effect of each of these test characteristics and cost drivers with statistical significance.

7.3 Lessons Learned

During this research, we had several lessons learned. In particular, we learned how to do research in industrial settings, which includes the negotiation of research goals, schedules and resources in order to have minor impact on the organizational performance. With respect to research techniques and methodologies, we learned how to propose and evaluate size measures. To verify the conformity of Execution Points with the measurement theory, we had to better understand and better formulate the concepts of our proposal.

During the planning and execution of our empirical studies, we learned how to control confounding factors and other threats to the validity of our studies. For that, we usually took much more time planning the studies than running them. The use of Design of Experiment techniques helped us to minimize the resources (participants and experimental material) required to run our experiments, making them feasible to industrial settings without impacting the teams productivity.

7.4 Future Work

During our research, we discussed some ideas of new models and possible extensions of our work. Some of these ideas were discussed in [16] and are presented next.

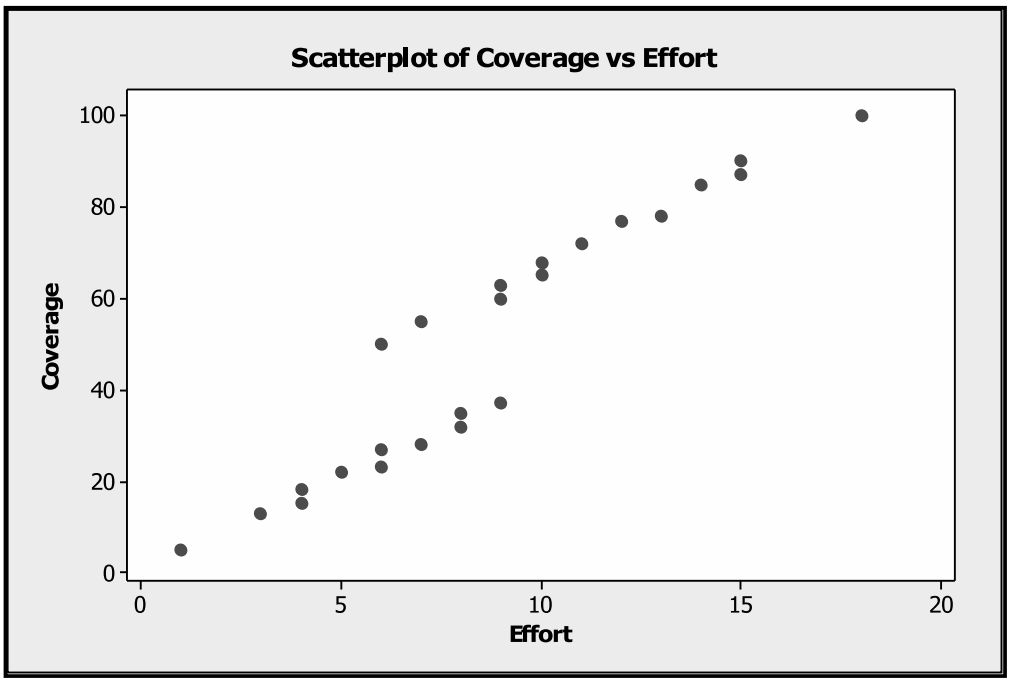
7.4.1 Estimating Test Execution Effort Based on Other Techniques

This research did not investigate the development and evaluation of models using techniques different from regression analysis, such as Bayesian Networks [123], decision trees [123], Case-Based Reasoning [139], Neural networks [123] and other machine learning approaches. However, these models may have interesting results with test execution effort estimation. To verify that, more empirical studies have to be run.

7.4.2 More Empirical Studies

In order to evaluate the our work, we run the empirical studies presented in Chapter 5. These experiments are restricted to the mobile and desktop application domains. We can try to extend this work for different application domains. In addition, a sequence of empirical studies should

Figure 7.1 Sample graph for test coverage vs. execution effort analysis.



be run in order to evaluate the other models proposed here as future work.

We also want to verify the performance of our method to measure the size and execution complexity of other types of tests (performance, stress, etc.).

7.4.3 Test Coverage vs. Test Execution Effort Analysis

When executing tests, it is important to achieve the highest possible level of test coverage [140]. Nevertheless, in practice, the resources are limited and it may not be possible to execute all test cases as desired. Then, the tester has to select a subset of tests to execute.

Some times it is possible to know the contribution of each test case for the test coverage before executing them, such as when using some Model-based testing (MBT) approaches [116]. Using our proposed test effort estimation model, it is also possible to know the effort to execute each test case individually.

In this context, we are investigating the ways to select the best set of tests to execute. For instance, the test selection can be supported by a tool that generates graphs such as the one shown in Figure 7.1. In this sample graph, each dot represents a test case. In general, tests with high coverage and low effort to execute are good candidates for selection. In this way, we can help to achieve maximum coverage within a limited cost.

7.4.4 Test Automation Effort Estimation

Another important and time-consuming activity is test automation. In general, automated tests are cheaper to execute, since they usually require few human interaction. For instance, a tester can control the execution of several automated tests at the same time. In addition, automated tests can be schedule to be automatically executed at night or on the weekends.

Nevertheless, test managers must consider the cost to automate the tests. For this reason, we intend to create an estimation model for test automation effort. This other proposed estimation model is also based on the test specifications written in CNL.

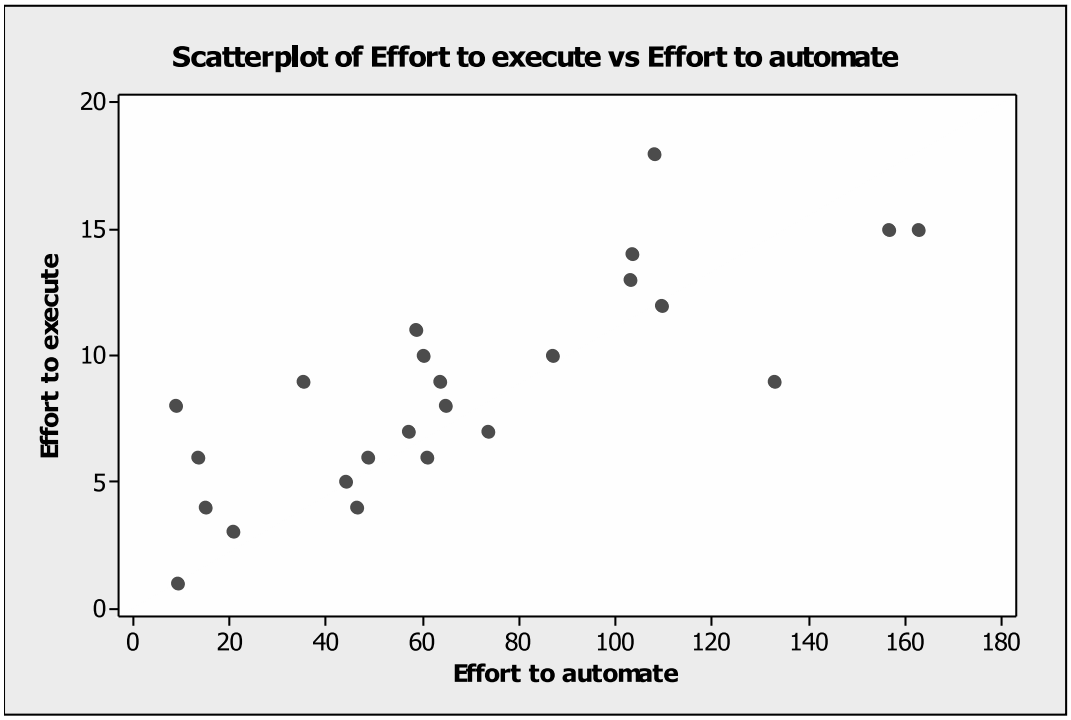
In this way, we need to identify the list of characteristics and risk factors that are relevant to estimate test automation effort.

7.4.5 Test Automation vs. Manual Execution Analysis

Although test automation has a lot of benefits, it may not be possible to automate all test cases. One of the main reasons is the cost of automation. The automation effort must be paid by its resulting benefits. In this research, we intend to create tools that combine the outputs of both test execution and automation effort estimation models and other relevant information in order to support test selection for automation.

For instance, the tester can select tests to automate by analyzing the effort to execute and to automate each test using graphs, as the sample shown in Figure 7.2. In addition, the model will regard other variables such as the expected test execution frequency and the available time to automate the tests.

Figure 7.2 Sample graph for test automation vs. execution effort analysis.



APPENDIX A

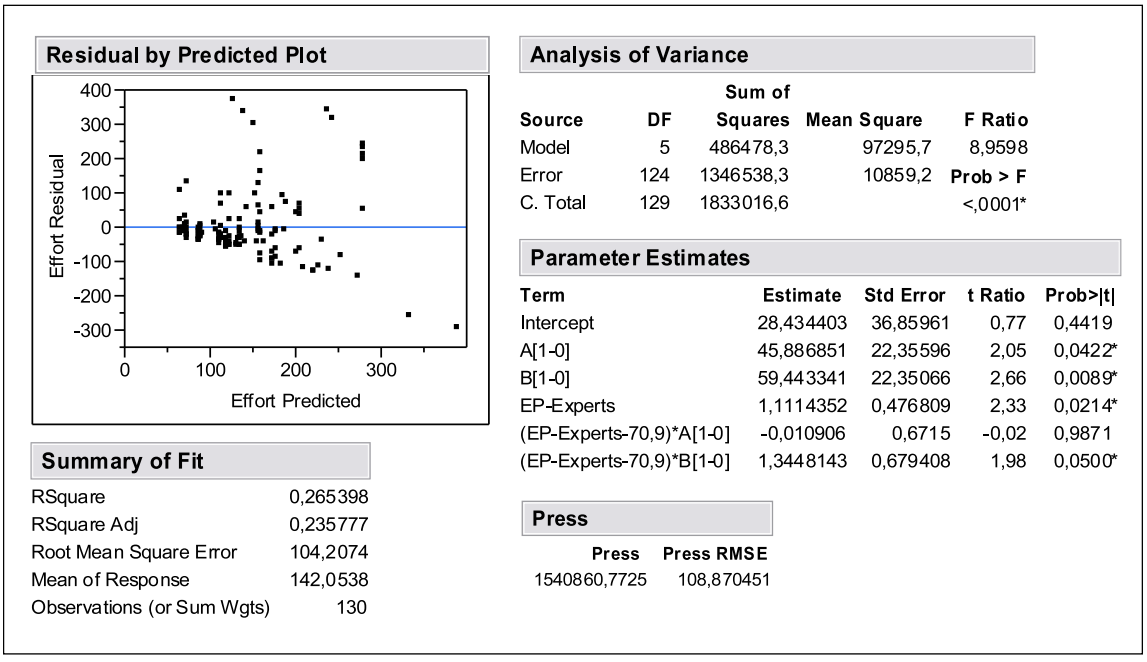
Statistical Analysis of Empirical Study ES3

This appendix shows several statistical analyses performed during the execution of empirical study ES3.

A.1 Regression and Other Statistical Analyses for Creating Estimation Models.

A.1.1 Regression Analyses for Size Measure M1.

This section presents the output of the regression analyses performed using measure M1.



Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio
Model	5	486478,3	97295,7	8,9598
Error	124	1346538,3	10859,2	Prob > F
C. Total	129	1833016,6		<,0001*

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	28,434403	36,85961	0,77	0,4419
A[1-0]	45,886851	22,35596	2,05	0,0422*
B[1-0]	59,443341	22,35066	2,66	0,0089*
EP-Experts	1,1114352	0,476809	2,33	0,0214*
(EP-Experts-70,9)*A[1-0]	-0,010906	0,6715	-0,02	0,9871
(EP-Experts-70,9)*B[1-0]	1,3448143	0,679408	1,98	0,0500*

Summary of Fit

RSquare	0,265398
RSquare Adj	0,235777
Root Mean Square Error	104,2074
Mean of Response	142,0538
Observations (or Sum Wgts)	130

Press

Press	Press RMSE
1540860,7725	108,870451

Figure A.1 Regression analysis for initial model and raw data using M1.

A.1.2 Regression Analyses for Size Measure M2.

This section presents the output of the regression analyses performed using measure M2.

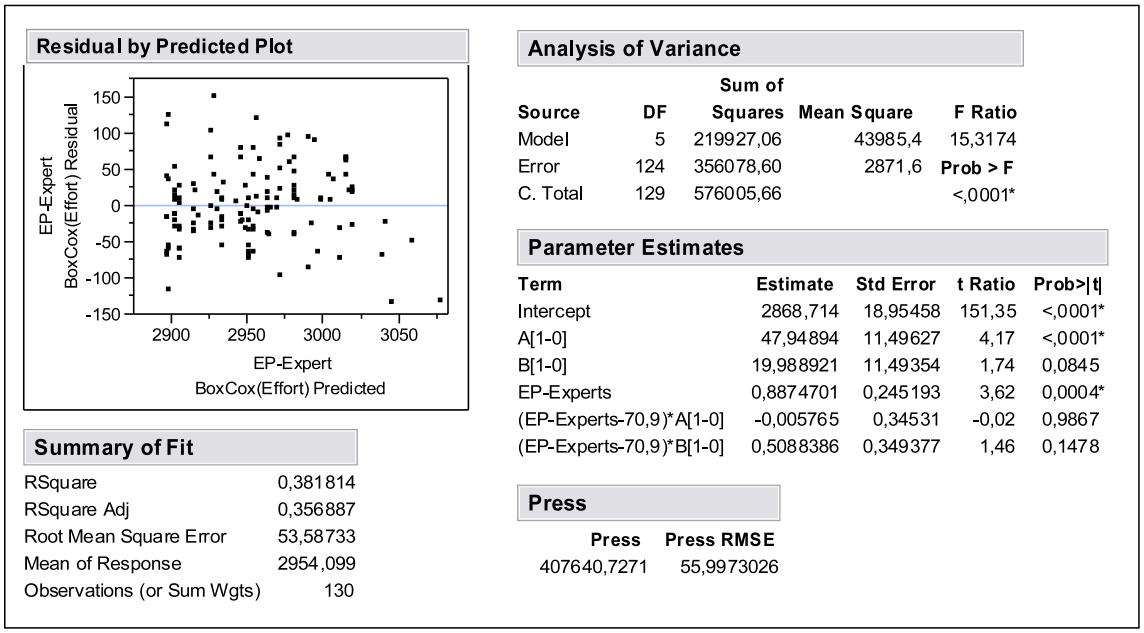


Figure A.2 Regression analysis for initial model and transformed data using M1.

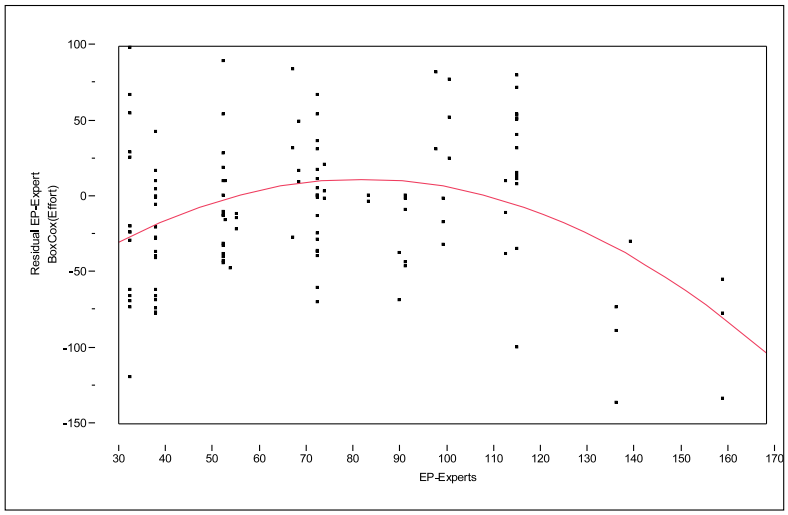


Figure A.3 Quadratic relationship between execution points (calibrated by experts) and the transformed effort.

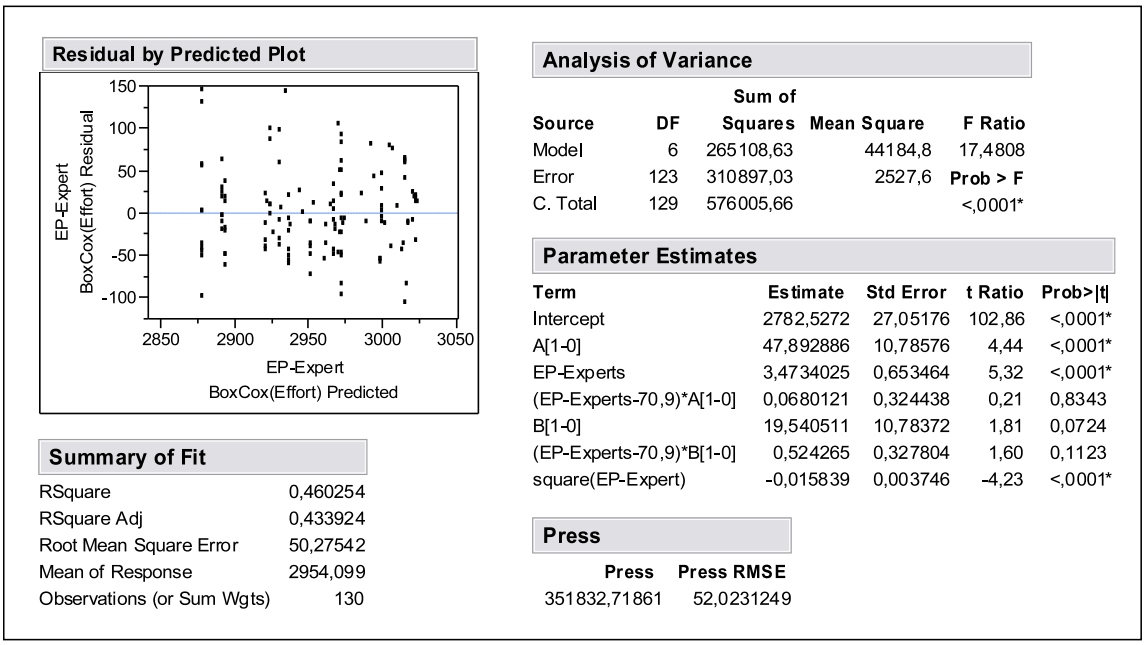


Figure A.4 Regression analysis for model with quadratic effect and transformed data using M1.

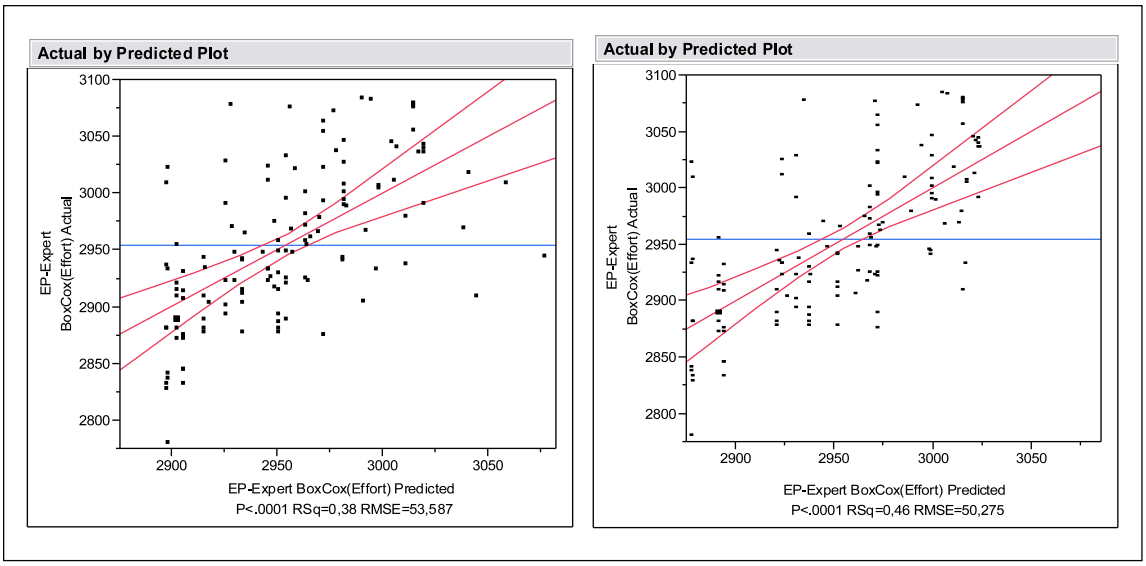


Figure A.5 Some improvement in the linear relationship between the actual and the predicted effort after inclusion of the quadratic effect using M1.

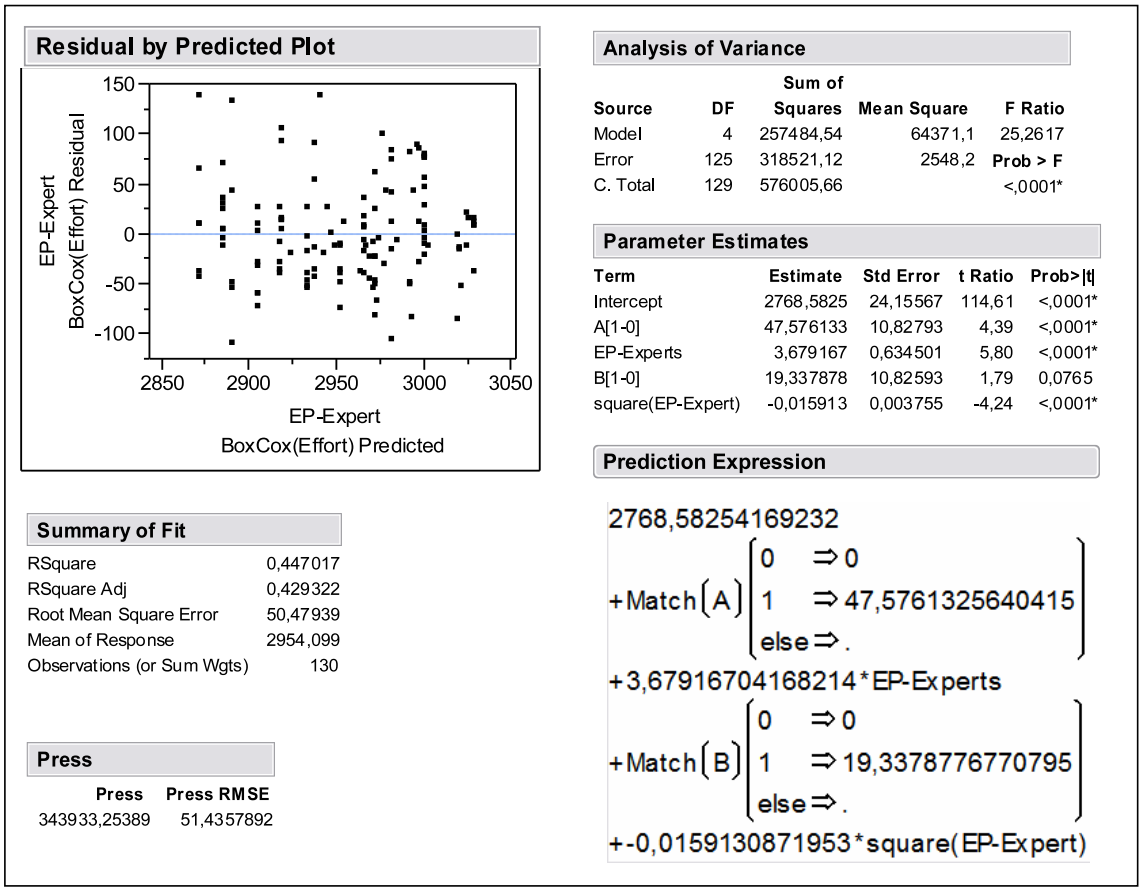
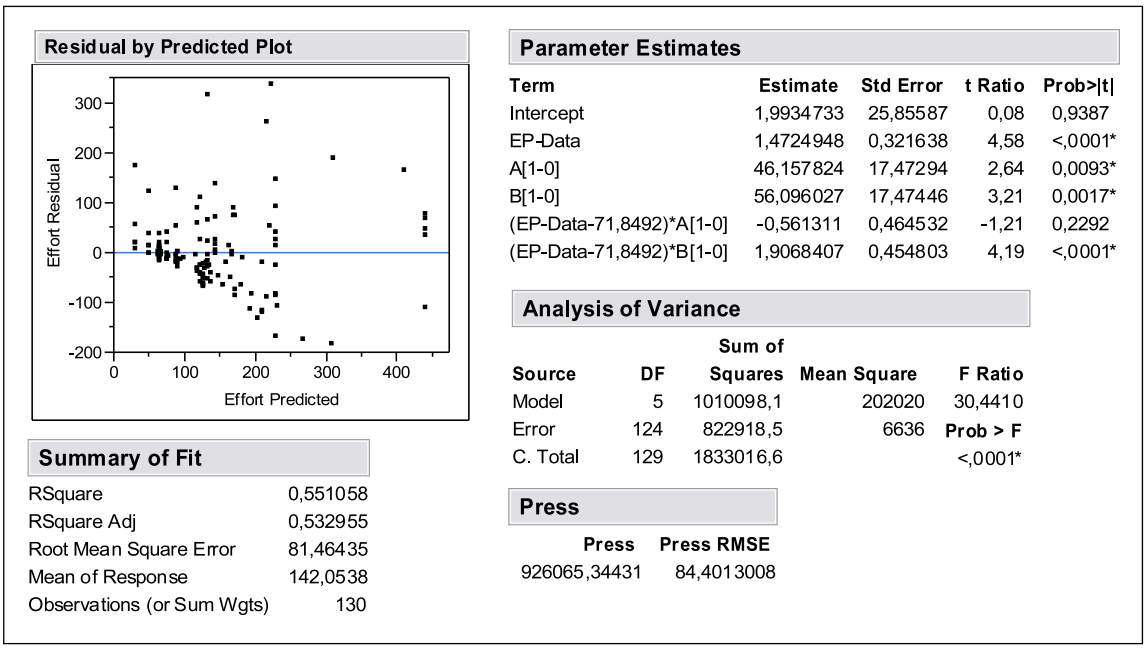


Figure A.6 Final regression analysis for model with measure M1 (EP-Experts).



Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	1,9934733	25,85587	0,08	0,9387
EP-Data	1,4724948	0,321638	4,58	<,0001*
A[1-0]	46,157824	17,47294	2,64	0,0093*
B[1-0]	56,096027	17,47446	3,21	0,0017*
(EP-Data-71,8492)*A[1-0]	-0,561311	0,464532	-1,21	0,2292
(EP-Data-71,8492)*B[1-0]	1,9068407	0,454803	4,19	<,0001*

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio
Model	5	1010098,1	202020	30,4410
Error	124	822918,5	6636	Prob > F
C. Total	129	1833016,6		<,0001*

Summary of Fit

RSquare	0,551058
RSquare Adj	0,532955
Root Mean Square Error	81,46435
Mean of Response	142,0538
Observations (or Sum Wgts)	130

Press

Press	Press RMSE
926065,34431	84,4013008

Figure A.7 Regression analysis for initial model and raw data using M2 (EP-Data).

A.2 Regression Analyses for Size Measure M3.

This section presents the output of the regression analyses performed using measure M3.

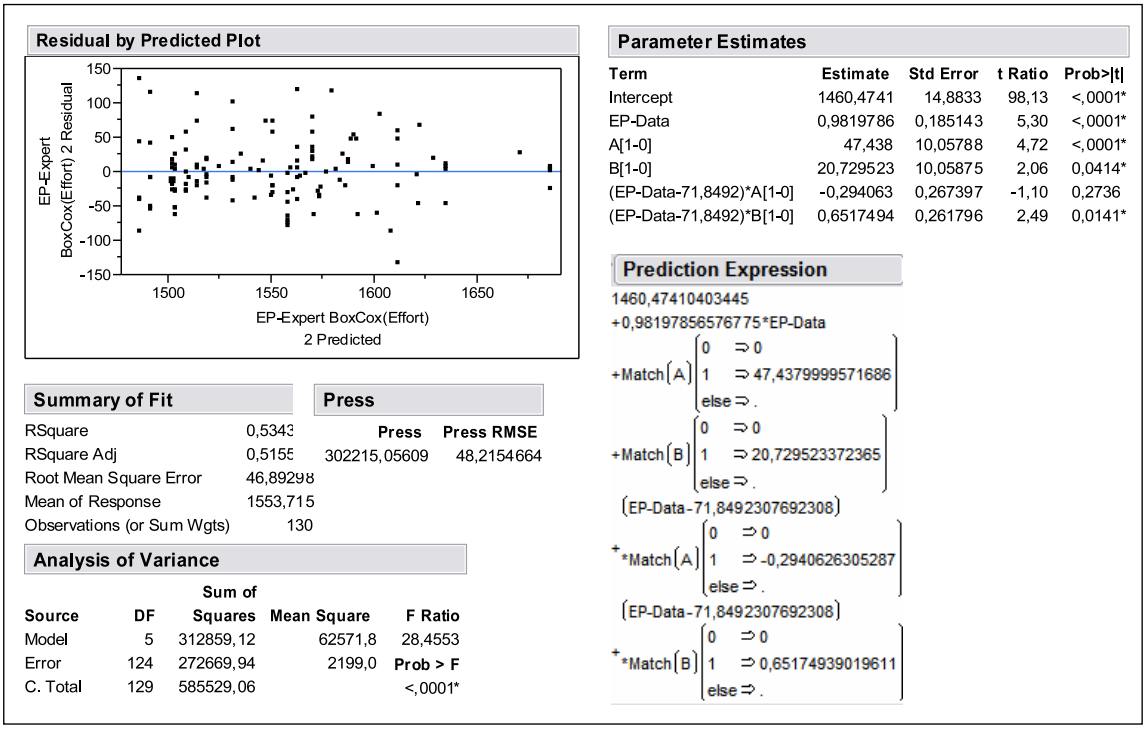
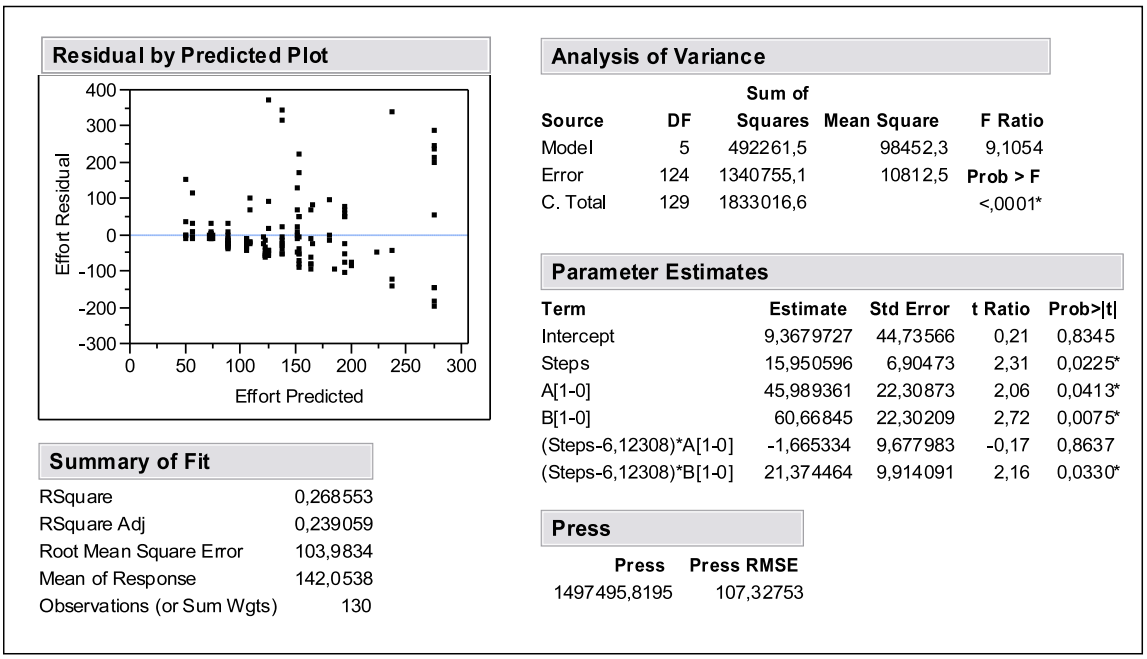


Figure A.8 Final regression analysis for model with measure M2 (EP-Data).

A.3 Regression Analyses for Size Measure M4.

This section presents the output of the regression and other statistical analyses performed using measure M4.



Summary of Fit

RSquare	0,268553
RSquare Adj	0,239059
Root Mean Square Error	103,9834
Mean of Response	142,0538
Observations (or Sum Wgts)	130

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio
Model	5	492261,5	98452,3	9,1054
Error	124	1340755,1	10812,5	Prob > F
C. Total	129	1833016,6		<,0001*

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	9,3679727	44,73566	0,21	0,8345
Steps	15,950596	6,90473	2,31	0,0225*
A[1-0]	45,989361	22,30873	2,06	0,0413*
B[1-0]	60,66845	22,30209	2,72	0,0075*
(Steps-6,12308)*A[1-0]	-1,665334	9,677983	-0,17	0,8637
(Steps-6,12308)*B[1-0]	21,374464	9,914091	2,16	0,0330*

Press

Press	Press RMSE
1497495,8195	107,32753

Figure A.9 Regression analysis for initial model and raw data using M3 (Steps).

A.4 Estimation Accuracy Achieved During the Montecarlo Experiment.

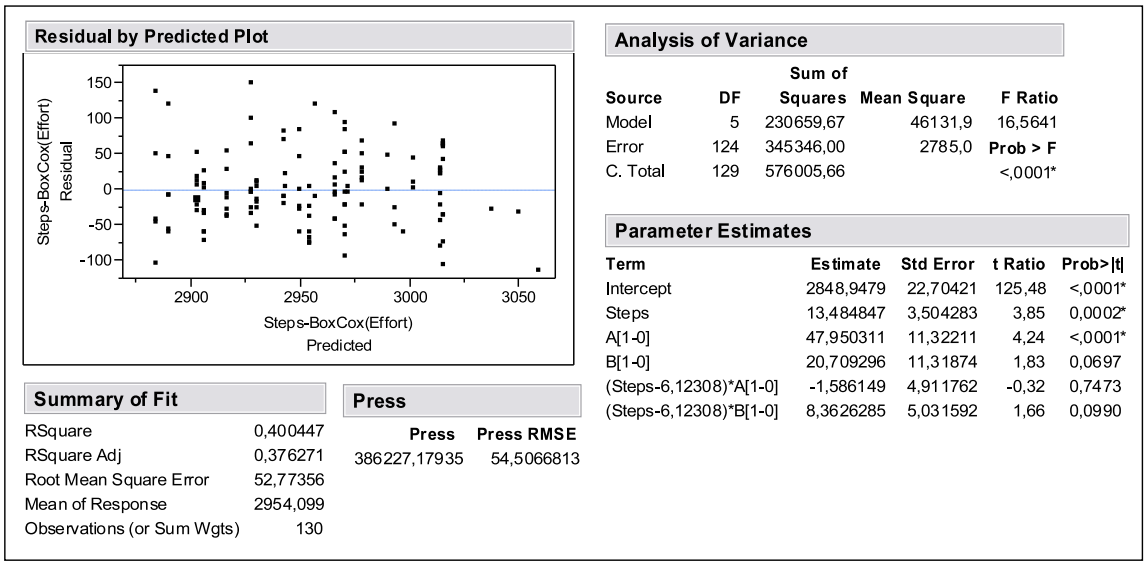


Figure A.10 Regression analysis for transformed data using M3 (Steps).

Table A.1 Person correlations between variables Effort, Keys, Screen, Delay and ListManip.

	Effort	Keys	Screen	Delay
Keys	0,338			
	0,000			
Screen	0,347	0,676		
	0,000	0,000		
Delay	0,639	0,259	0,406	
	0,000	0,003	0,000	
ListManip	0,300	0,778	0,137	0,256
	0,001	0,000	0,119	0,003

Cell Contents: Pearson correlation
P-Value

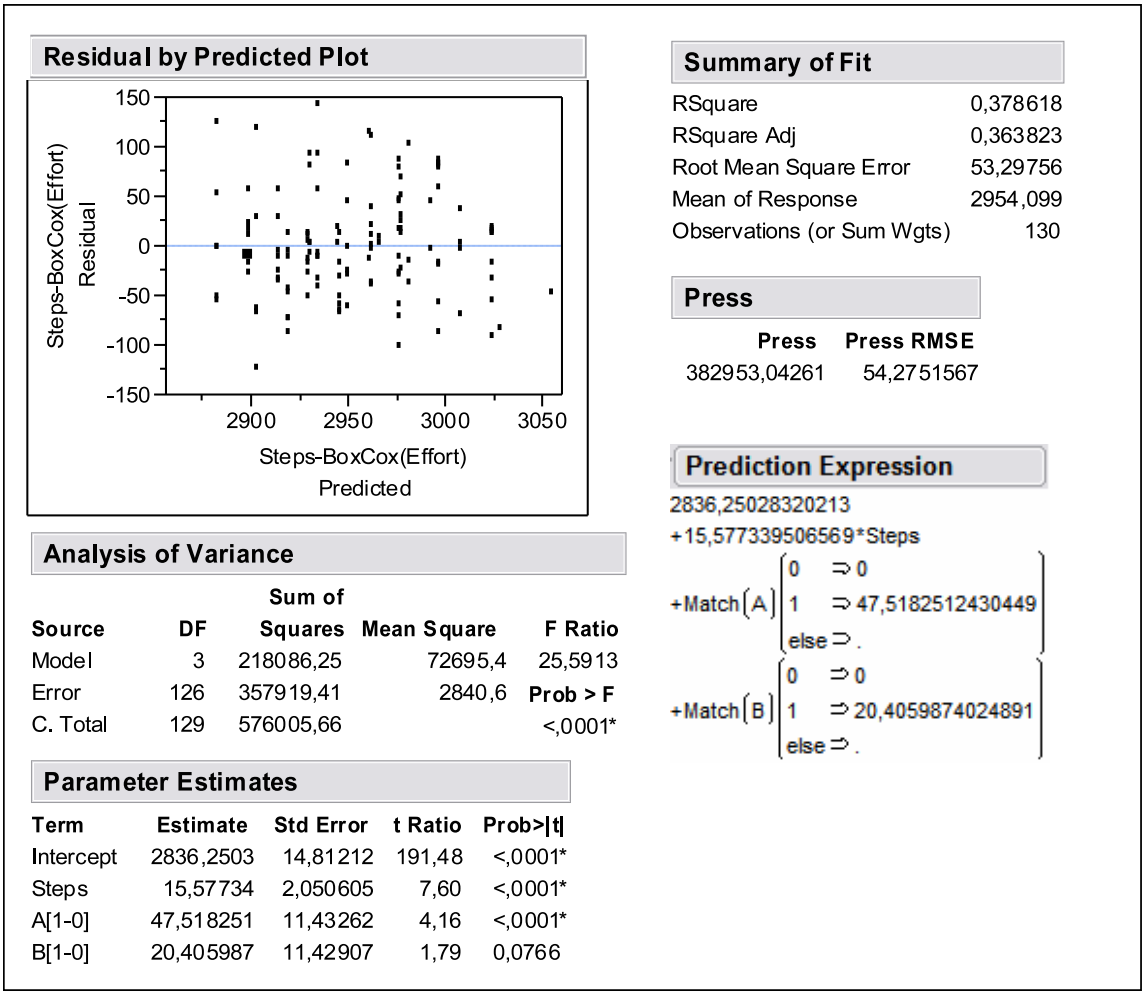


Figure A.11 Final regression analysis for model with measure M3 (Steps).

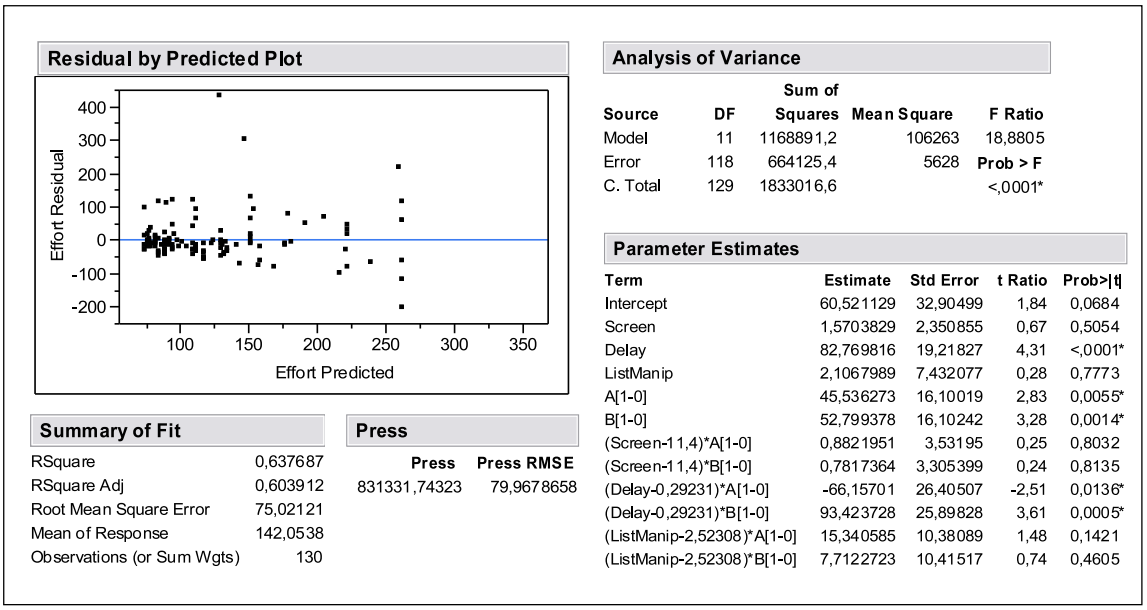


Figure A.12 Regression analysis for initial model and raw data using M4 (Screen, Delay and ListMap).

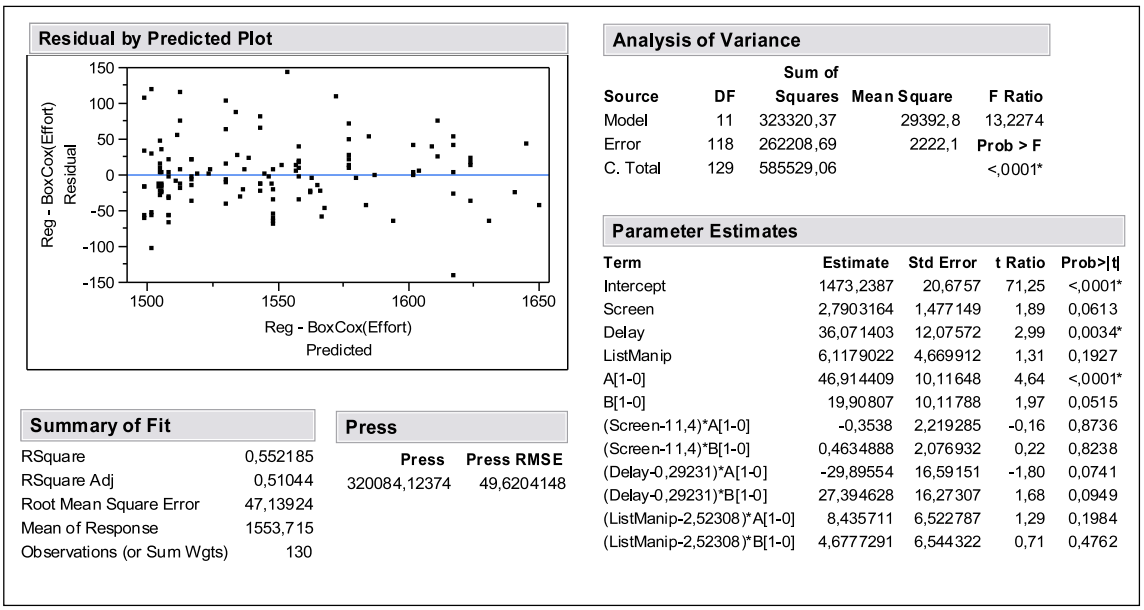


Figure A.13 Regression analysis with transformed data and using M4 (Screen, Delay and ListMap).

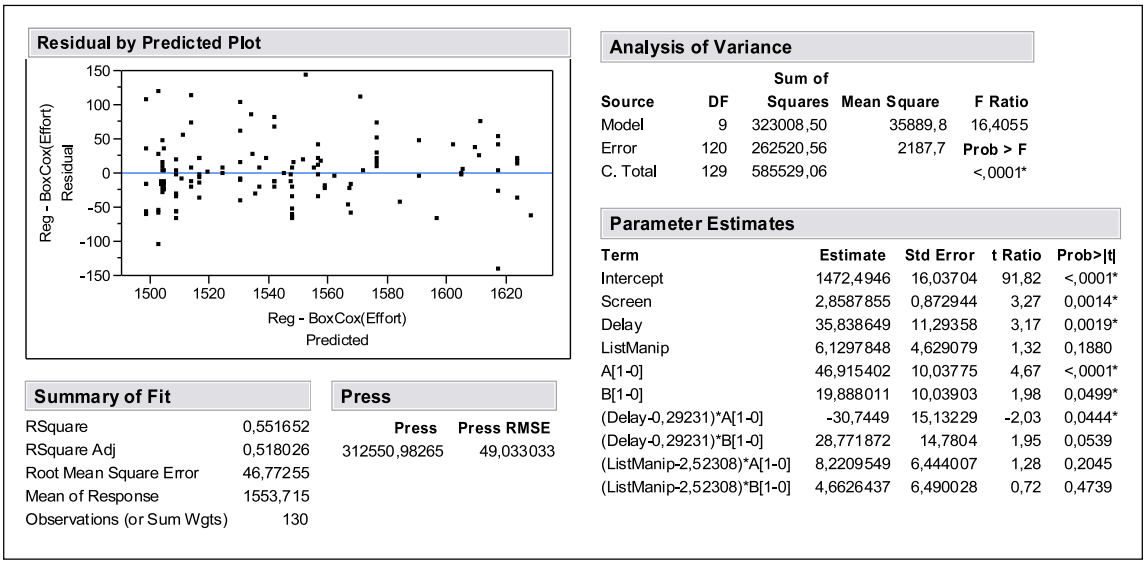
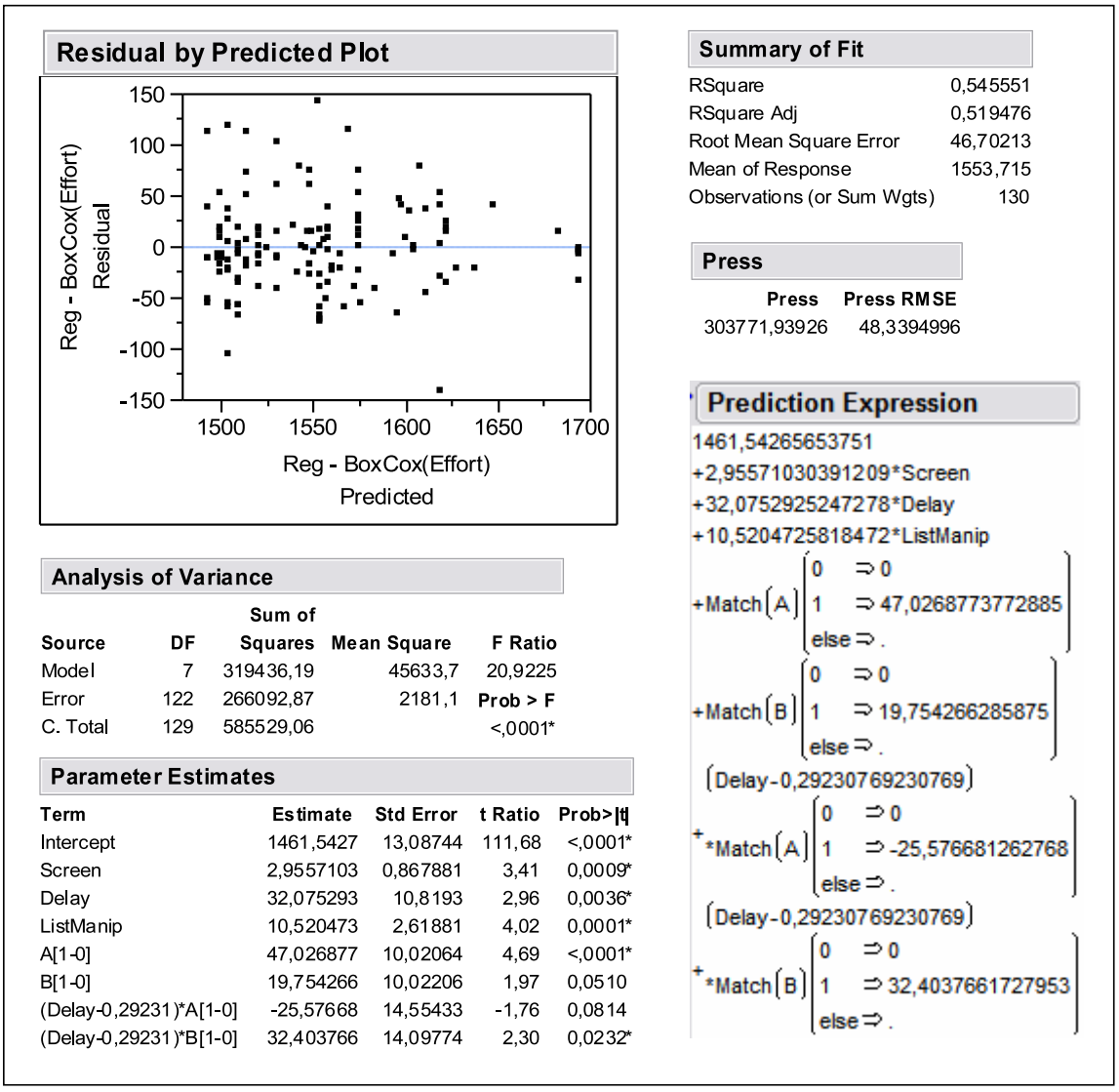


Figure A.14 Regression analysis after removing Screen and Tester interaction.



Summary of Fit

RSquare	0,545551
RSquare Adj	0,519476
Root Mean Square Error	46,70213
Mean of Response	1553,715
Observations (or Sum Wgts)	130

Press

Press	Press RMSE
303771,93926	48,3394996

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio
Model	7	319436,19	45633,7	20,9225
Error	122	266092,87	2181,1	Prob > F
C. Total	129	585529,06		<,0001*

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	1461,5427	13,08744	111,68	<,0001*
Screen	2,9557103	0,867881	3,41	0,0009*
Delay	32,075293	10,8193	2,96	0,0036*
ListManip	10,520473	2,61881	4,02	0,0001*
A[1-0]	47,026877	10,02064	4,69	<,0001*
B[1-0]	19,754266	10,02206	1,97	0,0510
(Delay-0,29231)*A[1-0]	-25,57668	14,55433	-1,76	0,0814
(Delay-0,29231)*B[1-0]	32,403766	14,09774	2,30	0,0232*

Prediction Expression

$$
\begin{aligned}
&1461{,}54265653751 \\
&+2{,}95571030391209 * \text{Screen} \\
&+32{,}0752925247278 * \text{Delay} \\
&+10{,}5204725818472 * \text{ListManip} \\
&+\text{Match}(A)\begin{cases} 0 \Rightarrow 0 \\ 1 \Rightarrow 47{,}0268773772885 \\ \text{else} \Rightarrow . \end{cases} \\
&+\text{Match}(B)\begin{cases} 0 \Rightarrow 0 \\ 1 \Rightarrow 19{,}754266285875 \\ \text{else} \Rightarrow . \end{cases} \\
&(\text{Delay} - 0{,}29230769230769) \\
&+ *\text{Match}(A)\begin{cases} 0 \Rightarrow 0 \\ 1 \Rightarrow -25{,}576681262768 \\ \text{else} \Rightarrow . \end{cases} \\
&(\text{Delay} - 0{,}29230769230769) \\
&+ *\text{Match}(B)\begin{cases} 0 \Rightarrow 0 \\ 1 \Rightarrow 32{,}4037661727953 \\ \text{else} \Rightarrow . \end{cases}
\end{aligned}
$$

Figure A.15 Final regression analysis for model with the multiple measure M4 (Screen, Delay and ListMap).

Table A.2 Achieved estimation accuracy during the Montecarlo experiment.

Run	Fold	$MMRE_{M1}$	$MMRE_{M2}$	$MMRE_{M3}$	$MMRE_{M4}$	$MdMRE_{M1}$	$MdMRE_{M2}$	$MdMRE_{M3}$	$MdMRE_{M4}$	$PRED(25)_{M1}$	$PRED(25)_{M2}$	$PRED(25)_{M3}$	$PRED(25)_{M4}$
1	1	0.3231531	0.3222408	0.3879816	0.3302809	0.2200194	0.1979572	0.3521780	0.2263055	53.1914900	55.3191500	46.8085100	55.3191500
	2	0.3687061	0.2981505	0.3658667	0.2889180	0.3465122	0.2048523	0.3292313	0.2017730	33.3333300	54.7619000	35.7142900	52.3809500
	3	0.4055633	0.3379296	0.3699111	0.3416559	0.2116998	0.2276806	0.2326314	0.1611436	56.0975600	58.5365900	51.2195100	63.4146300
2	1	0.8082085	0.4755509	0.5601816	0.5659554	0.5032686	0.3914574	0.5579014	0.4838891	31.1111100	31.1111100	28.8888900	31.1111100
	2	0.3199700	0.2371512	0.2826592	0.2432853	0.2583015	0.1717762	0.2700247	0.1867208	46.6666700	64.4444400	48.8888900	64.4444400
	3	0.3356722	0.3321568	0.3161393	0.3310696	0.2510364	0.2045042	0.2303129	0.1914859	50.0000000	55.0000000	52.5000000	57.5000000
3	1	0.3831275	0.3329893	0.3941076	0.3067447	0.3269603	0.2508045	0.3459371	0.2489294	34.8837200	48.8372100	39.5348800	51.1627900
	2	0.4583230	0.3012411	0.3698476	0.4388989	0.2437702	0.2442298	0.2515116	0.2886696	50.0000000	50.0000000	50.0000000	45.6521700
	3	0.3795359	0.3676712	0.3772175	0.4019098	0.2693037	0.2354573	0.3393405	0.1941013	48.7804900	56.0975600	43.9024400	56.0975600
4	1	0.4031048	0.3676349	0.5026704	0.3501078	0.3893295	0.3319917	0.4402304	0.3077768	34.8837200	44.1860500	32.5581400	41.8604700
	2	0.2995103	0.3192169	0.3293302	0.4511147	0.2101507	0.1616278	0.2727008	0.1965387	56.8181800	61.3636400	45.4545500	61.3636400
	3	0.3827815	0.2694322	0.3518752	0.3119563	0.2814537	0.2255025	0.2350464	0.2067006	44.1860500	51.1627900	51.1627900	53.4883700
5	1	0.3424800	0.3139264	0.3552702	0.3135339	0.3292214	0.2657454	0.3435031	0.2706394	36.9565200	45.6521700	43.4782600	47.8260900
	2	1.2735840	0.3852773	0.5311836	0.5503391	0.2307896	0.2488098	0.3001957	0.2486581	51.2195100	51.2195100	41.4634100	51.2195100
	3	0.3532152	0.2818783	0.3545528	0.2671516	0.3106591	0.2212512	0.2848413	0.1788499	41.8604700	53.4883700	44.1860500	62.7907000
6	1	0.3447737	0.4228141	0.4718681	0.4021450	0.2750306	0.2550669	0.2977026	0.2075058	45.0000000	47.5000000	37.5000000	55.0000000
	2	0.3647691	0.3230157	0.3711690	0.3372393	0.3053829	0.2860536	0.2663031	0.2501254	44.4444400	48.8888900	44.4444400	48.8888900
	3	0.3878517	0.3227557	0.3900215	0.3434537	0.3464986	0.2398275	0.3363814	0.2487052	37.7777800	51.1111100	40.0000000	51.1111100
7	1	0.4608072	0.3619990	0.4216753	0.3633618	0.3921776	0.2508791	0.3764264	0.2126513	22.2222200	48.8888900	31.1111100	57.7777800
	2	0.6385491	0.3091850	0.4230127	0.3925942	0.2606970	0.2398228	0.3359569	0.2050851	44.1860500	51.1627900	44.1860500	51.1627900
	3	0.3411599	0.3127663	0.3504575	0.3353045	0.2417284	0.2376973	0.3268247	0.3280628	52.3809500	52.3809500	40.4761900	45.2381000
8	1	0.3695658	0.2587222	0.3295230	0.2906511	0.2771141	0.2011871	0.2310858	0.1933677	45.6521700	56.5217400	50.0000000	56.5217400
	2	0.3857831	0.3501629	0.4563051	0.3436118	0.2823556	0.2575570	0.3877016	0.2876184	40.9090900	47.7272700	36.3636400	47.7272700
	3	0.3267374	0.3459486	0.3473023	0.3361639	0.3431156	0.2687971	0.2817649	0.2494405	45.0000000	42.5000000	42.5000000	52.5000000
9	1	0.4301671	0.3028943	0.3894596	0.3312418	0.3558362	0.1837366	0.3083495	0.1903318	43.4782600	60.8695700	43.4782600	60.8695700
	2	0.3226473	0.3311261	0.3390044	0.3077798	0.2312563	0.2619608	0.2673120	0.2151440	55.8139500	48.8372100	46.5116300	53.4883700
	3	0.3144915	0.3095553	0.3894399	0.3119565	0.2561495	0.2484808	0.3495004	0.2929986	46.3414600	51.2195100	41.4634100	41.4634100
10	1	0.3782253	0.3664297	0.3812491	0.3597830	0.3231096	0.2229840	0.2447773	0.2167557	38.6363600	54.5454500	52.2727300	56.8181800
	2	0.3127602	0.2565278	0.4169600	0.2630931	0.3004206	0.1401137	0.3449308	0.1804112	44.1860500	58.1395300	41.8604700	58.1395300
	3	0.4101570	0.3468671	0.4223514	0.4198185	0.2979135	0.2759639	0.3597334	0.3319871	44.1860500	44.1860500	37.2093000	37.2093000
11	1	0.3160376	0.2793473	0.3033297	0.2610053	0.2313745	0.2377623	0.2185913	0.1983536	55.3191500	53.1914900	51.0638300	63.8297900
	2	0.3389720	0.4102436	0.4980558	0.4674766	0.2851647	0.2782390	0.3478130	0.3072740	48.7804900	46.3414600	43.9024400	43.9024400
	3	0.3936770	0.3627783	0.4049295	0.3330514	0.3146490	0.2562198	0.3129560	0.2243989	45.2381000	47.6190500	40.4761900	52.3809500
12	1	0.3816348	0.3171687	0.3983212	0.3064865	0.2831230	0.2482651	0.3224443	0.2284614	47.6190500	50.0000000	42.8571400	54.7619000
	2	0.2983577	0.2963749	0.2972389	0.2926312	0.2438710	0.2061782	0.2751800	0.2455689	52.1739100	56.5217400	43.4782600	52.1739100
	3	0.8484145	0.4034218	0.5098526	0.5961846	0.4151261	0.1941184	0.3734938	0.3124658	40.4761900	54.7619000	40.4761900	45.2381000
13	1	0.7571444	0.4241337	0.5973104	0.6566125	0.4910010	0.3104128	0.5666175	0.3904472	31.8181800	43.1818200	29.5454500	36.3636400
	2	0.3678053	0.2644300	0.3406024	0.2659470	0.3150985	0.2120615	0.3216466	0.2010791	35.5555600	57.7777800	37.7777800	62.2222200
	3	0.2801965	0.3021304	0.2680231	0.3132029	0.1874391	0.1762225	0.1981086	0.2047955	63.4146300	58.5365900	58.5365900	56.0975600
14	1	0.3924630	0.3583864	0.5338854	0.3678526	0.2589884	0.2561219	0.3697831	0.2356765	47.5000000	50.0000000	37.5000000	52.5000000
	2	0.3548780	0.3645307	0.3597169	0.3579502	0.2952041	0.3177460	0.3359592	0.3107713	44.4444400	37.7777800	42.2222200	44.4444400
	3	0.3304880	0.2630977	0.3303476	0.2619533	0.3003557	0.1950516	0.2379633	0.1610393	46.6666700	64.4444400	55.5555600	62.2222200

15	1	0.2904830	0.2710063	0.5005408	0.2726495	0.2135658	0.1400226	0.2869511	0.1578536	59.5238100	64.2857100	45.2381000	61.9047600	
	2	0.4488122	0.3861752	0.4501162	0.4212459	0.3441601	0.2998069	0.4246570	0.2803793	31.9148900	42.5531900	31.9148900	46.8085100	
	3	0.3759861	0.3410088	0.3687985	0.3158255	0.3190396	0.2304074	0.3407867	0.2108731	31.7073200	53.6585400	43.9024400	58.5365900	
	1	0.4171941	0.2782483	0.4411465	0.2641473	0.3083049	0.1846931	0.3473898	0.1697331	44.1860500	55.8139500	44.1860500	60.4651200	
16	2	0.4193076	0.4086050	0.4668451	0.4715510	0.3136406	0.2756592	0.3387610	0.3171086	34.0909100	45.4545500	40.9090900	43.1818200	
	3	0.3740837	0.3850148	0.4059280	0.3796023	0.3225798	0.3383976	0.3372809	0.2882041	39.5348800	41.8604700	37.2093000	41.8604700	
	1	0.4033106	0.4042740	0.4196828	0.4092889	0.3050726	0.3298018	0.3670797	0.3641608	40.9090900	38.6363600	29.5454500	40.9090900	
17	2	0.4042971	0.2817891	0.3549501	0.2915481	0.3527203	0.1968417	0.2485510	0.1957725	43.1818200	56.8181800	50.0000000	54.5454500	
	3	0.2720138	0.2507760	0.4042051	0.2282906	0.2235242	0.1522805	0.2639264	0.1581760	54.7619000	61.9047600	47.6190500	66.6666700	
	1	0.3740371	0.3521402	0.3761406	0.4383943	0.3241792	0.2455112	0.2903017	0.2509793	42.2222200	51.1111100	40.0000000	48.8888900	
18	2	0.3718238	0.3710872	0.3695392	0.3559424	0.3864646	0.2969627	0.3327533	0.2650990	43.1818200	43.1818200	43.1818200	47.7272700	
	3	0.3015659	0.2864805	0.4564183	0.2952485	0.2292078	0.2195339	0.3297123	0.2334276	53.6585400	51.2195100	41.4634100	53.6585400	
	1	0.5374282	0.2989806	0.4302508	0.4579440	0.3352734	0.2237530	0.3344979	0.2678835	37.5000000	56.2500000	39.5833300	45.8333300	
19	2	0.3680821	0.3415138	0.4016134	0.3182518	0.2928886	0.2844749	0.2994710	0.1957172	45.6521700	47.8260900	41.3043500	54.3478300	
	3	0.3795666	0.4095046	0.4342311	0.4025095	0.2548819	0.2586800	0.3387307	0.2724784	50.0000000	50.0000000	47.2222200	50.0000000	
	1	0.3689718	0.3083550	0.3653394	0.3087586	0.3209625	0.2453803	0.3267531	0.2279828	39.1304300	50.0000000	43.4782600	54.3478300	
20	2	0.6234981	0.3996726	0.4742701	0.4823132	0.4657378	0.3510290	0.4612770	0.4098534	36.1702100	42.5531900	36.1702100	38.2978700	
	3	0.3005221	0.2604501	0.2931436	0.2350128	0.2192784	0.1759679	0.2123798	0.1682139	54.0540500	64.8648600	54.0540500	64.8648600	
	1	0.2512398	0.2579160	0.2765426	0.2697048	0.1711904	0.2195263	0.1744131	0.1764700	65.1162800	58.1395300	60.4651200	58.1395300	
21	2	0.3818387	0.4030685	0.3904774	0.5397877	0.3647787	0.2463772	0.3479498	0.2954381	40.9090900	50.0000000	40.9090900	45.4545500	
	3	0.7020670	0.3423741	0.5197481	0.6846622	0.3415775	0.2783690	0.3670599	0.3978615	32.5581400	44.1860500	25.5814000	39.5348800	
	1	0.9147913	0.3443745	0.4297557	0.4273250	0.2816322	0.2872796	0.3077031	0.3060680	38.6363600	40.9090900	45.4545500	45.4545500	
22	2	0.4247661	0.3865697	0.4301702	0.3958360	0.3728182	0.2685263	0.3714420	0.2403840	36.3636400	47.7272700	36.3636400	52.2727300	
	3	0.3095269	0.2660507	0.3072073	0.2692346	0.3098379	0.2302718	0.2796263	0.2307205	47.6190500	54.7619000	47.6190500	54.7619000	
	1	0.3087700	0.2855006	0.2928306	0.2806263	0.2168805	0.1917723	0.1912730	0.1930079	53.3333300	55.5555600	55.5555600	57.7777800	
23	2	0.3896682	0.3739602	0.4276404	0.3960767	0.3456354	0.2613371	0.3174709	0.2687629	41.3043500	50.0000000	41.3043500	47.8260900	
	3	0.3799771	0.4246405	0.5135088	0.3606956	0.3396388	0.3408349	0.3613756	0.3091225	41.0256400	38.4615400	33.3333300	46.1538500	
	1	0.3054624	0.2665792	0.3127117	0.2723393	0.2258112	0.1948328	0.2442640	0.1866395	54.7619000	57.1428600	57.1428600	57.1428600	
24	2	1.3330650	0.3485674	0.6396040	0.5030265	0.3873722	0.3045933	0.4262326	0.3565426	31.1111100	44.4444400	35.5555600	40.0000000	
	3	0.3583825	0.3403032	0.3323760	0.3452216	0.2374548	0.2457892	0.2337940	0.1609647	51.1627900	53.4883700	51.1627900	58.1395300	
	1	0.4923840	0.3962998	0.5084618	0.6437915	0.3911838	0.2830203	0.3577838	0.2650202	36.3636400	45.4545500	38.6363600	43.1818200	
25	2	0.3430444	0.3316746	0.3968287	0.3609079	0.2399020	0.2178125	0.2921784	0.2549675	50.0000000	52.1739100	39.1304300	50.0000000	
	3	0.3186181	0.2631613	0.2897880	0.2682154	0.2684474	0.2261512	0.2617005	0.1943262	47.5000000	55.0000000	47.5000000	60.0000000	
	1	0.3441864	0.2963088	0.3687368	0.2755318	0.2839783	0.2252772	0.3056884	0.2171104	41.3043500	52.1739100	43.4782600	56.5217400	
26	2	0.3801342	0.3600816	0.3960429	0.3517967	0.2965076	0.2766621	0.2722739	0.2544760	37.7777800	44.4444400	46.6666700	48.8888900	
	3	0.3786089	0.4934357	0.4116495	0.5697831	0.2578853	0.3171148	0.3096966	0.3139776	46.1538500	46.1538500	46.1538500	46.1538500	
	1	0.3585842	0.3038143	0.4223971	0.3170740	0.2908019	0.2166593	0.3673678	0.3025103	43.4782600	54.3478300	39.1304300	43.4782600	
27	2	0.3936629	0.2985101	0.3556246	0.3280930	0.3255773	0.1815890	0.2870713	0.1819533	43.4782600	63.0434800	41.3043500	60.8695700	
	3	0.3295887	0.3680235	0.3598696	0.3507570	0.2153556	0.2845943	0.2360606	0.2600249	57.8947400	47.3684200	50.0000000	50.0000000	
	1	0.3362577	0.3058933	0.3649527	0.2682416	0.2682499	0.1832973	0.2392332	0.1633367	47.8260900	58.6956500	52.1739100	60.8695700	
28	2	0.3727300	0.2892695	0.3726712	0.2974676	0.3019123	0.2434624	0.3440066	0.2355123	45.2381000	50.0000000	42.8571400	50.0000000	
	3	0.3913811	0.3417571	0.4048732	0.3463225	0.3458093	0.2266235	0.3463727	0.2464200	38.0952400	52.3809500	38.0952400	52.3809500	
	1	0.5000447	0.3425452	0.4802927	0.4932172	0.3999501	0.2558706	0.4429714	0.3587928	38.6363600	45.4545500	38.6363600	43.1818200	
29	2	0.3435460	0.3060727	0.3137854	0.3234526	0.2361788	0.1748290	0.2158636	0.1553292	52.2727300	65.9090900	52.2727300	65.9090900	
	3	0.3096423	0.2722647	0.3054111	0.2520151	0.2502888	0.2095254	0.2975098	0.1870984	50.0000000	57.1428600	42.8571400	64.2857100	

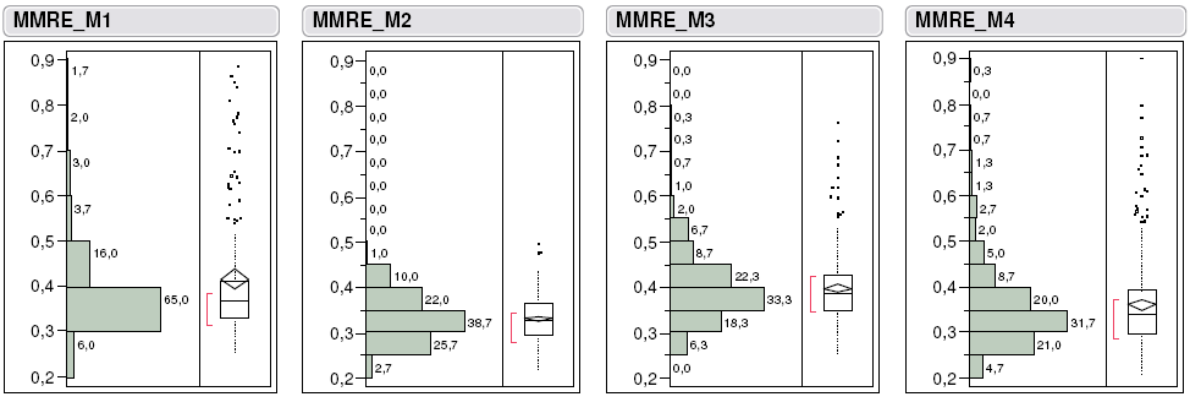
30	1	0.2833690	0.2832454	0.2812119	0.2765234	0.1875372	0.1657786	0.2235453	0.1989869	55.5555600	60.0000000	53.3333300	64.4444400
	2	0.6930475	0.4409406	0.6833792	0.5560549	0.4715069	0.3635104	0.5213627	0.3538105	30.2325600	39.5348800	32.5581400	39.5348800
	3	0.3031468	0.2801734	0.3238490	0.2850277	0.2302785	0.2050848	0.2132957	0.1649786	52.3809500	59.5238100	52.3809500	61.9047600
31	1	0.3134348	0.3692380	0.3344984	0.3598269	0.2613027	0.3110798	0.2705707	0.2827104	48.8888900	35.5555600	44.4444400	40.0000000
	2	0.3734707	0.3064326	0.3939990	0.3671665	0.2777410	0.2053310	0.3665319	0.2342816	47.6190500	59.5238100	40.4761900	52.3809500
	3	0.3780099	0.2936234	0.4477804	0.2952292	0.3582080	0.2317931	0.3775016	0.2224544	34.8837200	53.4883700	32.5581400	53.4883700
32	1	0.3206003	0.3365997	0.3589137	0.3617232	0.2839640	0.2745082	0.2596079	0.2686170	45.6521700	45.6521700	45.6521700	47.8260900
	2	0.4866637	0.3061124	0.4813852	0.4675444	0.2802344	0.2082908	0.3315499	0.2398751	46.6666700	60.0000000	40.0000000	51.1111100
	3	0.4083581	0.3430425	0.4044617	0.3360041	0.3389951	0.2386452	0.3343528	0.2160797	43.5897400	53.8461500	46.1538500	53.8461500
33	1	0.3639443	0.2906365	0.3185624	0.3233284	0.2893718	0.1709287	0.2819027	0.1881776	40.9090900	59.0909100	47.7272700	68.1818200
	2	0.5092185	0.3216261	0.4577393	0.4012983	0.3457513	0.2214786	0.3649441	0.3085567	41.3043500	54.3478300	36.9565200	45.6521700
	3	0.3531944	0.4062154	0.3948630	0.3907978	0.2787786	0.2866121	0.3230154	0.2707934	47.5000000	47.5000000	47.5000000	50.0000000
34	1	0.3872245	0.2927672	0.3828613	0.2961784	0.3588772	0.2736898	0.3249266	0.2312848	35.5555600	48.8888900	40.0000000	55.5555600
	2	0.2980375	0.2901695	0.3773043	0.2673053	0.2283433	0.2298492	0.2572913	0.2490905	55.5555600	55.5555600	48.8888900	51.1111100
	3	0.3994787	0.4178776	0.4123786	0.4237660	0.2538480	0.3458772	0.3783578	0.3378922	47.5000000	37.5000000	37.5000000	40.0000000
35	1	0.3485101	0.3198515	0.3830275	0.3147973	0.2635711	0.2682711	0.3274967	0.1963317	48.7804900	48.7804900	43.9024400	58.5365900
	2	0.3585062	0.3018215	0.3386403	0.3009672	0.2382187	0.2155205	0.2818950	0.2248350	52.1739100	54.3478300	45.6521700	52.1739100
	3	0.3512021	0.3664276	0.4323005	0.4082586	0.2856521	0.2333269	0.3144980	0.2743557	44.1860500	53.4883700	34.8837200	46.5116300
36	1	0.3494392	0.3175181	0.4165073	0.3043716	0.2679499	0.2180013	0.2952186	0.2394557	48.8372100	53.4883700	44.1860500	51.1627900
	2	0.4289240	0.3737027	0.4044823	0.3868890	0.3848649	0.2898872	0.3039190	0.2501850	36.9565200	47.8260900	30.4347800	50.0000000
	3	0.3277001	0.2820631	0.3539027	0.2806643	0.2432227	0.2324233	0.2979814	0.2124138	51.2195100	53.6585400	43.9024400	53.6585400
37	1	0.3423939	0.3182573	0.3861947	0.3440409	0.3122389	0.2400186	0.2883325	0.3292488	37.7777800	53.3333300	40.0000000	44.4444400
	2	0.3361501	0.3854553	0.3791369	0.3754219	0.2670313	0.2878582	0.3431034	0.3189152	48.8372100	37.2093000	37.2093000	37.2093000
	3	0.4080500	0.2977754	0.3667399	0.3315998	0.2444853	0.1909441	0.2495883	0.1597293	50.0000000	71.4285700	50.0000000	66.6666700
38	1	0.4094632	0.3564929	0.3953017	0.3450629	0.3345304	0.2864089	0.3764993	0.3050956	33.3333300	42.2222200	37.7777800	48.8888900
	2	0.2519506	0.3446561	0.3294185	0.4006743	0.1838864	0.1953442	0.1890097	0.2089161	65.2173900	60.8695700	58.6956500	56.5217400
	3	0.3662047	0.2977964	0.3921851	0.2947717	0.3288188	0.2132341	0.3142106	0.2126861	38.4615400	51.2820500	38.4615400	58.9743600
39	1	0.4177722	0.3446110	0.4239742	0.3364388	0.3378390	0.2712184	0.3832618	0.2749812	35.5555600	46.6666700	35.5555600	44.4444400
	2	0.2985433	0.2785681	0.2870263	0.3034691	0.2581886	0.2155529	0.2230703	0.2712917	48.8888900	57.7777800	51.1111100	48.8888900
	3	0.3428025	0.3986164	0.4891319	0.3968313	0.2559416	0.2448331	0.3497236	0.2516973	47.5000000	52.5000000	35.0000000	50.0000000
40	1	0.3495400	0.3858762	0.4241322	0.3788236	0.2564210	0.2744419	0.3581022	0.2316018	48.8888900	44.4444400	44.4444400	57.7777800
	2	0.2820709	0.2795667	0.2508831	0.2492312	0.2328352	0.2113983	0.1921624	0.1914922	52.3809500	54.7619000	57.1428600	59.5238100
	3	0.4409121	0.3167561	0.4375662	0.3390313	0.3910239	0.3005511	0.3840969	0.3039090	32.5581400	46.5116300	25.5814000	41.8604700
41	1	0.8370299	0.4238575	0.5546240	0.5210420	0.3744060	0.4036022	0.3535961	0.3708013	36.3636400	36.3636400	38.6363600	43.1818200
	2	0.3196623	0.2421846	0.3076349	0.2339196	0.2548419	0.1675777	0.2152227	0.1759846	48.8372100	60.4651200	51.1627900	60.4651200
	3	0.3422110	0.2837287	0.3511695	0.3001181	0.2817927	0.1818347	0.3023532	0.2161937	48.8372100	58.1395300	46.5116300	58.1395300
42	1	0.3461391	0.2701948	0.3105463	0.2485185	0.2770662	0.2057036	0.3002912	0.1751378	43.4782600	60.8695700	43.4782600	65.2173900
	2	0.4232825	0.3968237	0.4302298	0.3921070	0.3025430	0.3269384	0.3834751	0.2982242	38.6363600	43.1818200	40.9090900	47.7272700
	3	0.3283335	0.3004384	0.3987532	0.3160236	0.3035561	0.2011893	0.3553197	0.2800800	45.0000000	60.0000000	45.0000000	47.5000000
43	1	0.3744972	0.3474159	0.4052111	0.3458709	0.2991157	0.2875880	0.3510954	0.2388446	44.4444400	46.6666700	44.4444400	51.1111100
	2	0.3046342	0.2929095	0.2938761	0.2841256	0.2643946	0.2101754	0.2857507	0.2110668	45.4545500	56.8181800	43.1818200	54.5454500
	3	0.8835826	0.3303985	0.5127742	0.8981970	0.4034160	0.2487695	0.3290634	0.3825366	26.8292700	51.2195100	39.0243900	29.2682900
44	1	0.3180588	0.2546016	0.3041562	0.2606906	0.2329550	0.1623470	0.2042192	0.1969433	52.0833300	56.2500000	58.3333300	60.4166700
	2	0.3736585	0.3958080	0.4709125	0.3909272	0.3249426	0.2553936	0.4439730	0.2347462	41.8604700	46.5116300	32.5581400	53.4883700
	3	0.3924350	0.3141357	0.3467153	0.2849552	0.2721457	0.2698391	0.3002115	0.2497884	46.1538500	48.7179500	43.5897400	51.2820500

45	1	0.3646964	0.3466661	0.3502764	0.3047249	0.2803691	0.2092272	0.2373079	0.1654402	45.2381000	57.1428600	54.7619000	59.5238100	
	2	0.2954406	0.2438265	0.2849636	0.2387744	0.2525128	0.1800699	0.2012093	0.1937823	48.9361700	63.8297900	51.0638300	65.9574500	
	3	1.2138920	0.4043676	0.6674945	0.5415898	0.3557955	0.3760702	0.4572442	0.3835527	34.1463400	41.4634100	36.5853700	36.5853700	
46	1	0.3573765	0.3667912	0.4844711	0.3497609	0.2519710	0.2205379	0.4164761	0.2753149	48.8372100	53.4883700	37.2093000	48.8372100	
	2	0.3870369	0.2980134	0.3764762	0.2948557	0.3163812	0.2445686	0.2864887	0.2453954	40.9090900	52.2727300	40.9090900	50.0000000	
	3	0.3431302	0.3307701	0.3403515	0.3706898	0.3004839	0.1988265	0.2918909	0.2168029	46.5116300	51.1627900	44.1860500	51.1627900	
47	1	0.3480073	0.3239897	0.3494652	0.3367563	0.3393435	0.2751529	0.2723313	0.2670792	45.6521700	47.8260900	47.8260900	45.6521700	
	2	0.2818555	0.2853387	0.3502404	0.2575568	0.2145042	0.2144474	0.2307708	0.1932026	60.0000000	57.7777800	51.1111100	62.2222200	
	3	0.4168519	0.3211166	0.4355479	0.3703156	0.3806636	0.2181770	0.3752339	0.2108869	41.0256400	56.4102600	33.3333300	53.8461500	
48	1	0.4776570	0.4262623	0.5000144	0.4115953	0.3971374	0.3531788	0.3626891	0.3251758	38.0952400	35.7142900	38.0952400	42.8571400	
	2	0.3493451	0.4040998	0.3753392	0.3380636	0.2766208	0.3032733	0.2864593	0.2520937	45.4545500	43.1818200	43.1818200	50.0000000	
	3	0.2841693	0.2792160	0.3308060	0.2956085	0.1795146	0.1437683	0.2125015	0.1867457	59.0909100	63.6363600	54.5454500	63.6363600	
49	1	0.3823250	0.3783556	0.4012924	0.3698800	0.2955928	0.2652477	0.3169985	0.2846088	39.0243900	46.3414600	41.4634100	46.3414600	
	2	0.3897459	0.3222902	0.4018754	0.3250870	0.3944305	0.2705539	0.2946955	0.2513523	31.2500000	47.9166700	37.5000000	47.9166700	
	3	0.6273910	0.3115725	0.4903514	0.7670410	0.2045839	0.2209000	0.2807858	0.3523317	60.9756100	53.6585400	43.9024400	41.4634100	
50	1	0.3011898	0.2671293	0.2911238	0.2577343	0.2035402	0.2115738	0.2284272	0.1764882	58.1395300	60.4651200	53.4883700	62.7907000	
	2	0.7629380	0.3787962	0.5253933	0.4831045	0.4764328	0.2531188	0.3469107	0.2758163	34.7826100	50.0000000	34.7826100	45.6521700	
	3	0.3307482	0.3103517	0.3561616	0.3015692	0.2972877	0.2691661	0.3466939	0.3043237	46.3414600	46.3414600	36.5853700	43.9024400	
51	1	0.3325419	0.3182673	0.3413560	0.3109597	0.2522373	0.2108389	0.2490843	0.1870237	48.8888900	53.3333300	51.1111100	62.2222200	
	2	0.8623596	0.4363446	0.6167958	0.4884441	0.4723434	0.3853933	0.5013774	0.3767512	32.6087000	30.4347800	23.9130400	34.7826100	
	3	0.3777739	0.2578787	0.3195976	0.2346020	0.2967699	0.1794138	0.2424723	0.1829162	43.5897400	56.4102600	51.2820500	58.9743600	
52	1	0.3379470	0.2684425	0.3044442	0.2741645	0.3146326	0.1521768	0.2553727	0.1852021	43.1818200	56.8181800	50.0000000	59.0909100	
	2	2.2952330	0.3280128	0.4522834	0.5751354	0.3997054	0.2721249	0.3465319	0.2824563	33.3333300	46.6666700	37.7777800	44.4444400	
	3	0.3968939	0.3815512	0.4022771	0.3855837	0.3114373	0.2564582	0.2939588	0.2385948	41.4634100	48.7804900	48.7804900	51.2195100	
53	1	0.6168577	0.3052824	0.4069238	0.3115733	0.3057302	0.2513273	0.3001500	0.2508241	41.3043500	50.0000000	36.9565200	50.0000000	
	2	0.4156757	0.4057802	0.3970704	0.3699827	0.3819782	0.2193539	0.3279420	0.2396545	44.1860500	51.1627900	44.1860500	51.1627900	
	3	0.3371090	0.3311884	0.3732708	0.3318758	0.2501578	0.2065605	0.2278501	0.1947848	48.7804900	53.6585400	53.6585400	53.6585400	
54	1	0.4177208	0.3173788	0.3647455	0.3246763	0.2833268	0.2866402	0.2884213	0.2767458	45.4545500	47.7272700	45.4545500	47.7272700	
	2	0.3004346	0.3808072	0.3721697	0.3674183	0.1956311	0.2272725	0.2525537	0.2559421	60.8695700	52.1739100	47.8260900	50.0000000	
	3	0.3780430	0.2921516	0.3891560	0.2747169	0.3572323	0.2226886	0.3307580	0.1943553	35.0000000	52.5000000	32.5000000	62.5000000	
55	1	0.3617110	0.3025318	0.3587986	0.3204869	0.3396498	0.2292748	0.3106527	0.2032021	33.3333300	57.7777800	40.0000000	60.0000000	
	2	0.5485970	0.3685521	0.5942407	0.5630981	0.2782342	0.3063868	0.3218867	0.2872325	45.6521700	45.6521700	41.3043500	45.6521700	
	3	0.3532838	0.3385601	0.3615898	0.3320067	0.2702162	0.2180326	0.3023577	0.2628983	46.1538500	51.2820500	46.1538500	48.7179500	
56	1	0.4831650	0.4083559	0.4721307	0.4179023	0.3843412	0.3508630	0.4014507	0.3476362	37.2093000	41.8604700	30.2325600	34.8837200	
	2	0.3213175	0.2911832	0.3628796	0.3161809	0.2559081	0.2119930	0.2646545	0.2108083	48.8888900	53.3333300	44.4444400	55.5555600	
	3	0.2838518	0.2604604	0.3623545	0.2145703	0.2349044	0.1985978	0.2542373	0.1477393	52.3809500	61.9047600	50.0000000	69.0476200	
57	1	0.4005793	0.2861437	0.3987691	0.2930363	0.3592903	0.2599099	0.3600205	0.2421992	37.7777800	48.8888900	40.0000000	51.1111100	
	2	0.2916518	0.3617607	0.4765699	0.3499719	0.2334459	0.2710598	0.3929733	0.3032751	50.0000000	47.6190500	42.8571400	42.8571400	
	3	0.3491349	0.3187589	0.3300827	0.3310285	0.3085647	0.2218095	0.2660726	0.1977956	44.1860500	58.1395300	48.8372100	60.4651200	
58	1	0.4018602	0.3139289	0.3623776	0.3573823	0.3662520	0.1965252	0.2607690	0.2164089	42.5531900	55.3191500	46.8085100	55.3191500	
	2	0.3337689	0.3041560	0.4020999	0.2981016	0.2783665	0.2097132	0.3493247	0.2479333	46.5116300	58.1395300	41.8604700	51.1627900	
	3	0.3391564	0.3402499	0.3663555	0.3486315	0.2590295	0.2949167	0.3070869	0.3135866	47.5000000	42.5000000	42.5000000	40.0000000	
59	1	0.4498700	0.3317841	0.3917154	0.3325495	0.3480761	0.2650102	0.2687261	0.2243865	35.7142900	47.6190500	45.2381000	52.3809500	
	2	0.3831006	0.2964351	0.3740240	0.3226434	0.3687319	0.2556685	0.3166982	0.2765100	35.4166700	50.0000000	37.5000000	39.5833300	
	3	0.3038043	0.3459165	0.4026401	0.3456502	0.1904594	0.1597200	0.3027627	0.2169758	52.5000000	55.0000000	45.0000000	55.0000000	

60	1	0.3589263	0.2871739	0.3517363	0.2947723	0.3028370	0.2291126	0.2973947	0.2333354	41.3043500	52.1739100	45.6521700	52.1739100		
	2	0.2991415	0.3431870	0.3293367	0.3236304	0.2343984	0.2459972	0.2996721	0.2545594	53.4883700	51.1627900	46.5116300	46.5116300		
	3	0.6134123	0.4395802	0.5226206	0.7944776	0.3404141	0.2485714	0.3638122	0.2645668	39.0243900	51.2195100	36.5853700	48.7804900		
61	1	0.3396187	0.3620345	0.4138367	0.3815478	0.3415026	0.2553550	0.3204835	0.2348563	40.0000000	48.8888900	35.5555600	53.3333300		
	2	0.3753746	0.3268695	0.3908199	0.3102815	0.2618977	0.2265235	0.3445931	0.2576401	50.0000000	52.1739100	43.4782600	47.8260900		
	3	0.3783412	0.3068600	0.3275114	0.2852289	0.3103977	0.2631821	0.2612191	0.2096248	35.8974400	48.7179500	48.7179500	51.2820500		
62	1	0.2975033	0.2445695	0.3351699	0.2516936	0.2647822	0.1520297	0.2411121	0.1504936	47.8260900	67.3913000	52.1739100	63.0434800		
	2	0.3702619	0.3261278	0.3597973	0.3182472	0.2878155	0.2729122	0.2942035	0.2462102	44.4444400	46.6666700	48.8888900	51.1111100		
	3	0.3814497	0.4396652	0.4509899	0.4451516	0.2933439	0.3573586	0.3782803	0.3175787	46.1538500	38.4615400	33.3333300	43.5897400		
63	1	0.2781554	0.2802427	0.3087087	0.2794998	0.2254738	0.2253318	0.2382803	0.1745132	52.3809500	57.1428600	52.3809500	54.7619000		
	2	0.4493243	0.3667502	0.4605924	0.5323613	0.3908908	0.3087118	0.3422935	0.3549524	28.2608700	43.4782600	34.7826100	32.6087000		
	3	0.3607106	0.4133715	0.3935899	0.4461057	0.2620063	0.2156865	0.2710175	0.2186045	47.6190500	57.1428600	45.2381000	59.5238100		
64	1	0.3208683	0.3263453	0.4072629	0.3060768	0.2396316	0.2167882	0.3528603	0.2445015	52.2727300	56.8181800	38.6363600	50.0000000		
	2	0.2992704	0.2825270	0.3065466	0.2830754	0.2349761	0.2535385	0.2366836	0.2303322	51.1627900	48.8372100	51.1627900	55.8139500		
	3	0.4101745	0.3532221	0.4040225	0.4389836	0.2897701	0.2217101	0.2831424	0.2365824	41.8604700	53.4883700	44.1860500	51.1627900		
65	1	0.6409660	0.3287646	0.4062270	0.4685041	0.3083567	0.2336524	0.3451475	0.2225224	41.3043500	54.3478300	43.4782600	52.1739100		
	2	0.3528045	0.3301349	0.3747853	0.3380706	0.2685175	0.2391098	0.2945604	0.2328995	47.8260900	52.1739100	43.4782600	54.3478300		
	3	0.3795107	0.2647317	0.3387536	0.2498443	0.3063557	0.2244121	0.3139166	0.1643284	34.2105300	60.5263200	47.3684200	55.2631600		
66	1	0.4292615	0.3329837	0.3754952	0.3017607	0.4004586	0.2525470	0.3166909	0.2453448	30.9523800	50.0000000	35.7142900	52.3809500		
	2	0.3693333	0.3081351	0.4034973	0.2889771	0.2703260	0.2616951	0.2903629	0.2041599	48.8888900	46.6666700	40.0000000	57.7777800		
	3	0.3372249	0.3525165	0.3803914	0.4554702	0.2847138	0.2256834	0.2967768	0.2948771	44.1860500	51.1627900	37.2093000	44.1860500		
67	1	0.7796990	0.4361010	0.5634064	0.6639696	0.4025065	0.3062011	0.4801849	0.2965684	41.8604700	44.1860500	37.2093000	41.8604700		
	2	0.3681397	0.3032667	0.3231127	0.2916153	0.2737978	0.2494775	0.2501962	0.2304561	47.7272700	50.0000000	50.0000000	54.5454500		
	3	0.3111440	0.2789157	0.3442807	0.2748028	0.2501114	0.1796696	0.2455565	0.1624417	48.8372100	65.1162800	51.1627900	62.7907000		
68	1	0.4472970	0.3848026	0.4234863	0.3814446	0.4232903	0.2241720	0.3440248	0.2149737	25.5814000	51.1627900	34.8837200	55.8139500		
	2	0.2778869	0.2860347	0.3235853	0.2632478	0.2203392	0.2063015	0.2583811	0.1665342	54.3478300	54.3478300	47.8260900	63.0434800		
	3	0.3615526	0.2975240	0.3803815	0.3308861	0.2236040	0.2444038	0.2692001	0.2231530	53.6585400	53.6585400	43.9024400	53.6585400		
69	1	0.3722997	0.2573874	0.3388939	0.2581483	0.2739082	0.1789773	0.2837586	0.1946706	38.6363600	63.6363600	45.4545500	56.8181800		
	2	0.3595004	0.4045980	0.3904144	0.4428626	0.3225070	0.3287225	0.3234238	0.3472849	40.0000000	40.0000000	35.5555600	42.2222200		
	3	1.3797940	0.3144600	0.5026606	0.4625831	0.2815145	0.2322296	0.2826371	0.2674876	41.4634100	51.2195100	48.7804900	48.7804900		
70	1	0.3748779	0.3254920	0.4638268	0.3160359	0.3262693	0.2135878	0.3306665	0.1590473	39.1304300	54.3478300	39.1304300	54.3478300		
	2	0.4103353	0.3403302	0.4278954	0.3636980	0.2983943	0.2847131	0.3542297	0.3201655	47.6190500	45.2381000	42.8571400	47.6190500		
	3	0.3231707	0.3139042	0.3321167	0.3098944	0.2575099	0.2288988	0.2796486	0.2406760	47.6190500	52.3809500	45.2381000	52.3809500		
71	1	0.3859624	0.3670697	0.4070047	0.4449743	0.3273867	0.2739938	0.3222382	0.2913048	34.8837200	48.8372100	30.2325600	46.5116300		
	2	0.5765127	0.2668863	0.4157799	0.6114376	0.2304649	0.2027045	0.2359255	0.2637420	55.5555600	60.0000000	51.1111100	48.8888900		
	3	0.3483999	0.3586299	0.3530073	0.3422202	0.2337044	0.2499269	0.2581167	0.2034260	52.3809500	50.0000000	50.0000000	52.3809500		
72	1	0.3901118	0.2473078	0.3716073	0.2529911	0.2676198	0.1806375	0.2427873	0.1361679	44.4444400	60.0000000	51.1111100	64.4444400		
	2	0.4451124	0.4211556	0.4292926	0.3984997	0.3545732	0.2754781	0.3082434	0.2157195	32.5000000	45.0000000	42.5000000	57.5000000		
	3	0.3180987	0.3395709	0.3782553	0.3467012	0.2324962	0.2315946	0.2898166	0.2549304	55.5555600	51.1111100	48.8888900	48.8888900		
73	1	0.3893728	0.3596497	0.3856463	0.3915228	0.3222361	0.2620475	0.3118680	0.2283765	37.7777800	48.8888900	37.7777800	51.1111100		
	2	0.3150695	0.3628259	0.4395888	0.3550312	0.2287800	0.2430316	0.3806946	0.2951521	53.4883700	51.1627900	44.1860500	46.5116300		
	3	0.3758090	0.2758224	0.3262157	0.2760445	0.3210766	0.2125765	0.2511349	0.1814537	42.8571400	59.5238100	50.0000000	54.7619000		
74	1	0.4408192	0.4275484	0.5080685	0.4353627	0.4675687	0.3718559	0.4787676	0.3543726	37.7777800	40.0000000	26.6666700	40.0000000		
	2	0.3276963	0.2527155	0.3103334	0.2644848	0.2437359	0.2086773	0.1990446	0.1834132	53.3333300	57.7777800	55.5555600	60.0000000		
	3	0.3789825	0.3669943	0.4334715	0.3781782	0.3123766	0.1869044	0.2717526	0.2140066	37.5000000	55.0000000	47.5000000	55.0000000		

75	1	0.3518443	0.3267643	0.3700717	0.3384420	0.2128322	0.2703933	0.2717296	0.2715327	52.1739100	45.6521700	50.0000000	47.8260900
	2	0.3441817	0.3595755	0.4672426	0.3621040	0.2946620	0.2782576	0.3766045	0.2497635	38.0952400	47.6190500	33.3333300	50.0000000
	3	0.3451528	0.2678763	0.3196665	0.2578086	0.2723640	0.1829943	0.2115323	0.1640147	47.6190500	61.9047600	54.7619000	64.2857100
76	1	0.3168950	0.2719494	0.3003691	0.2621978	0.2544241	0.2386344	0.2576079	0.2381368	50.0000000	52.1739100	50.0000000	56.5217400
	2	0.5098264	0.4139428	0.4996717	0.4109360	0.4442790	0.3193911	0.4143888	0.2635226	26.8292700	43.9024400	29.2682900	46.3414600
	3	0.2786612	0.3001802	0.3662923	0.3195228	0.2425002	0.2157418	0.2353085	0.2355185	53.4883700	53.4883700	51.1627900	58.1395300
77	1	0.3705594	0.3768175	0.4328096	0.3544802	0.3098273	0.2944988	0.3369948	0.2312887	48.8888900	48.8888900	42.2222200	51.1111100
	2	0.2982211	0.2792717	0.3049353	0.2771756	0.1848281	0.1784835	0.2203430	0.1654296	56.8181800	61.3636400	54.5454500	61.3636400
	3	0.5489652	0.3200751	0.4804607	0.3309194	0.4206593	0.2319440	0.4025407	0.2391441	26.8292700	51.2195100	29.2682900	51.2195100
78	1	0.4103070	0.2961961	0.3734318	0.3078541	0.3649047	0.2846688	0.3514701	0.2617636	32.6087000	47.8260900	36.9565200	43.4782600
	2	0.2969537	0.3152579	0.3070875	0.3073901	0.1976189	0.2417746	0.2364946	0.2545917	56.0975600	51.2195100	53.6585400	48.7804900
	3	0.6507405	0.3583787	0.5069428	0.4990380	0.3110879	0.1967235	0.3743607	0.2609798	46.5116300	51.1627900	41.8604700	48.8372100
79	1	0.3652011	0.3218576	0.3808787	0.4080897	0.2644668	0.2295662	0.3199696	0.3256644	46.6666700	57.7777800	37.7777800	42.2222200
	2	0.3656974	0.2838384	0.4022473	0.2741104	0.2741083	0.1919828	0.3134740	0.1732542	50.0000000	56.8181800	47.7272700	56.8181800
	3	0.3489550	0.3495649	0.3658232	0.3270024	0.3112977	0.2400223	0.2862741	0.2315561	36.5853700	51.2195100	46.3414600	56.0975600
80	1	0.3350087	0.3540168	0.3243074	0.4271601	0.2125236	0.1901688	0.2469283	0.2132167	54.7619000	59.5238100	50.0000000	52.3809500
	2	0.4162316	0.3225048	0.4414844	0.3418157	0.3620774	0.2408391	0.3625847	0.2362723	36.9565200	54.3478300	41.3043500	52.1739100
	3	0.3238401	0.3518221	0.4047847	0.3201529	0.2773167	0.2796759	0.2879156	0.2474613	47.6190500	47.6190500	45.2381000	50.0000000
81	1	0.3132461	0.3219281	0.4072988	0.3068295	0.2572532	0.2590808	0.3131026	0.2477491	48.8372100	44.1860500	39.5348800	51.1627900
	2	0.3338442	0.3416417	0.3601557	0.3251565	0.2622921	0.2688364	0.2979329	0.2195781	45.6521700	47.8260900	43.4782600	54.3478300
	3	0.4416975	0.3351405	0.4088000	0.3574613	0.2779434	0.1972192	0.2322468	0.1879061	43.9024400	58.5365900	53.6585400	58.5365900
82	1	0.5156373	0.3177855	0.4837145	0.3574559	0.4287993	0.2506738	0.3887741	0.2559183	21.4285700	50.0000000	33.3333300	50.0000000
	2	0.2760978	0.2775517	0.3133152	0.2602771	0.2013365	0.1958442	0.2555416	0.2028712	55.5555600	53.3333300	48.8888900	62.2222200
	3	0.3449104	0.3755705	0.3914767	0.4291842	0.2552617	0.2693277	0.2966026	0.2385236	48.8372100	48.8372100	44.1860500	53.4883700
83	1	0.3833680	0.3332808	0.3504952	0.3099113	0.3305595	0.2765275	0.2892102	0.2231634	35.7142900	47.6190500	47.6190500	57.1428600
	2	0.3299765	0.3179602	0.3877605	0.3059258	0.3040851	0.2403707	0.3673302	0.2290136	40.4255300	53.1914900	40.4255300	53.1914900
	3	0.3436131	0.3712486	0.3639196	0.5347240	0.1707582	0.1747488	0.2418403	0.2320838	58.5365900	58.5365900	51.2195100	51.2195100
84	1	0.7386648	0.3889671	0.5580965	0.6041432	0.4130162	0.2890679	0.5065037	0.2980317	36.9565200	43.4782600	30.4347800	41.3043500
	2	0.3312184	0.2291037	0.2956699	0.2438644	0.2652925	0.1696607	0.1843328	0.1775113	45.0000000	70.0000000	60.0000000	62.5000000
	3	0.3503513	0.3525747	0.3741310	0.3928504	0.3131674	0.2487421	0.3465060	0.2414775	40.9090900	50.0000000	36.3636400	52.2727300
85	1	0.3768612	0.3616867	0.3325100	0.3847385	0.3111535	0.2505156	0.2833472	0.2370604	43.4782600	50.0000000	43.4782600	54.3478300
	2	0.3256865	0.3278298	0.4303042	0.3087094	0.2327500	0.1849687	0.3252923	0.1857343	54.3478300	56.5217400	45.6521700	63.0434800
	3	0.3940904	0.3299992	0.4083663	0.3426210	0.3108716	0.2861796	0.3739267	0.2837205	39.4736800	42.1052600	39.4736800	42.1052600
86	1	0.3167974	0.2928361	0.2975346	0.3026221	0.2636116	0.2110499	0.2146039	0.1830404	46.6666700	60.0000000	51.1111100	60.0000000
	2	0.4104827	0.3322111	0.4076245	0.3113143	0.4092717	0.3030238	0.3469314	0.2534439	37.7777800	44.4444400	35.5555600	48.8888900
	3	0.3099783	0.3280262	0.4508528	0.3374917	0.2126703	0.2544846	0.3560249	0.2780511	52.5000000	47.5000000	42.5000000	45.0000000
87	1	0.4183568	0.3411251	0.4204802	0.3411233	0.4089016	0.2541263	0.4030705	0.2411782	32.6087000	47.8260900	34.7826100	50.0000000
	2	0.3028325	0.3149739	0.3159096	0.3506093	0.1688538	0.2151799	0.2249206	0.2155911	60.0000000	57.7777800	53.3333300	62.2222200
	3	0.3166065	0.2866915	0.5132950	0.2943289	0.2439448	0.2252775	0.3380711	0.2425137	51.2820500	56.4102600	43.5897400	51.2820500
88	1	0.7718368	0.4235806	0.6187970	0.6861628	0.3708225	0.2793652	0.4249438	0.2687733	35.5555600	37.7777800	35.5555600	46.6666700
	2	0.3506788	0.3211887	0.3709625	0.3019963	0.3132216	0.2386384	0.2913132	0.2082914	43.1818200	52.2727300	43.1818200	54.5454500
	3	0.3115945	0.3308167	0.2834091	0.2955990	0.2425241	0.2077583	0.2108555	0.1746010	51.2195100	56.0975600	58.5365900	56.0975600
89	1	0.3410640	0.2432120	0.3390348	0.2548133	0.2358209	0.1711340	0.2315940	0.1601417	52.1739100	65.2173900	52.1739100	67.3913000
	2	0.3142235	0.3310879	0.4020273	0.3274192	0.2602367	0.2577539	0.3317733	0.2703573	47.7272700	47.7272700	40.9090900	45.4545500
	3	0.4323974	0.3871653	0.3934658	0.3717334	0.3816186	0.3328582	0.3439345	0.2744407	37.5000000	42.5000000	37.5000000	47.5000000

90	1	0.3489064	0.2614509	0.3389931	0.2824408	0.3145974	0.2027398	0.2924216	0.2032923	38.2978700	61.7021300	42.5531900	61.7021300
	2	0.3587134	0.3950344	0.5049008	0.4656816	0.2478583	0.2953421	0.2704891	0.2994968	51.1627900	48.8372100	48.8372100	39.5348800
	3	0.3472310	0.3739562	0.3639682	0.3510876	0.2502139	0.2366591	0.2344232	0.2334771	50.0000000	52.5000000	52.5000000	52.5000000
91	1	0.5459279	0.2770252	0.5053487	0.5523080	0.3605594	0.2174157	0.3633588	0.2616532	42.2222200	60.0000000	44.4444400	48.8888900
	2	0.3256194	0.3507663	0.3796206	0.4032296	0.2779118	0.2547676	0.2549110	0.3854722	48.9361700	46.8085100	48.9361700	40.4255300
	3	0.3746793	0.4169853	0.4003693	0.3556700	0.2407592	0.2658369	0.3353677	0.1792421	50.0000000	47.3684200	39.4736800	52.6315800
92	1	0.3599760	0.3360941	0.3751943	0.3523008	0.3295788	0.3245900	0.3715006	0.2887550	36.9565200	43.4782600	39.1304300	43.4782600
	2	0.3184263	0.3120139	0.4067715	0.2934151	0.2419332	0.1885242	0.2399954	0.1795101	51.1627900	58.1395300	51.1627900	60.4651200
	3	0.4246668	0.3559542	0.4267487	0.3964185	0.3492239	0.2547337	0.3560764	0.2734448	31.7073200	48.7804900	36.5853700	46.3414600
93	1	0.5871281	0.2787247	0.4045150	0.3706311	0.2258212	0.1934794	0.2969777	0.1890692	52.1739100	60.8695700	45.6521700	58.6956500
	2	0.3277498	0.3389597	0.3255038	0.3216296	0.2556537	0.2443670	0.2683391	0.2416849	48.8888900	53.3333300	48.8888900	51.1111100
	3	0.4227246	0.3809746	0.4203892	0.3625769	0.4002227	0.2617689	0.3917751	0.2561860	35.8974400	48.7179500	35.8974400	48.7179500
94	1	0.3083997	0.2168323	0.2754460	0.2029147	0.2337619	0.1875625	0.1935236	0.1443843	55.5555600	64.4444400	55.5555600	68.8888900
	2	1.6183180	0.4741985	0.7197203	0.7235415	0.4834169	0.3583806	0.5375341	0.4061069	28.5714300	45.2381000	26.1904800	38.0952400
	3	0.3177171	0.3237089	0.3462156	0.3381957	0.2986992	0.2493960	0.2923990	0.2525051	44.1860500	51.1627900	46.5116300	48.8372100
95	1	0.3786423	0.3987485	0.4467908	0.3612250	0.3237319	0.3355805	0.3251528	0.2310552	42.2222200	37.7777800	28.8888900	53.3333300
	2	0.2982706	0.2584363	0.2913406	0.2315553	0.2014879	0.1850492	0.1876637	0.1549019	57.1428600	64.2857100	59.5238100	71.4285700
	3	0.4385959	0.4319274	0.5063706	0.4673583	0.3233762	0.3805828	0.4608949	0.4109027	41.8604700	39.5348800	37.2093000	39.5348800
96	1	0.3124963	0.2799691	0.4074673	0.2251768	0.2211005	0.1913427	0.3528121	0.1747627	53.3333300	62.2222200	46.6666700	68.8888900
	2	0.3733452	0.3038699	0.3844355	0.3514599	0.2593387	0.2076535	0.2923772	0.2088785	48.8372100	55.8139500	41.8604700	55.8139500
	3	0.3764826	0.3848851	0.3905337	0.3734660	0.3466340	0.3545853	0.3851914	0.3192598	38.0952400	35.7142900	30.9523800	33.3333300
97	1	0.3867470	0.3992236	0.3858684	0.3804548	0.3486876	0.3196042	0.3146273	0.2534807	30.2325600	41.8604700	39.5348800	48.8372100
	2	0.2998694	0.3427810	0.3741658	0.3579449	0.2176048	0.2496498	0.2726929	0.2838181	53.1914900	51.0638300	44.6808500	48.9361700
	3	0.6951454	0.2959008	0.7605825	0.7025348	0.3983250	0.2672535	0.3922007	0.3303522	30.0000000	47.5000000	35.0000000	42.5000000
98	1	0.4334657	0.3254603	0.4207962	0.3351549	0.3429166	0.2828291	0.3567806	0.2478828	34.0425500	44.6808500	29.7872300	51.0638300
	2	0.3296130	0.3320504	0.3732084	0.3363737	0.2213248	0.1600684	0.2466010	0.1551253	51.1627900	58.1395300	51.1627900	58.1395300
	3	0.2851279	0.3423681	0.3679259	0.3744569	0.2405277	0.1998190	0.2309230	0.2716161	52.5000000	52.5000000	52.5000000	45.0000000
99	1	0.5419061	0.3829216	0.4937181	0.3785928	0.4231820	0.2516885	0.3857705	0.2384920	23.2558100	48.8372100	34.8837200	53.4883700
	2	0.3479886	0.3337721	0.3907765	0.3018272	0.2428926	0.2155485	0.3287908	0.2304648	52.2727300	54.5454500	38.6363600	50.0000000
	3	0.2955894	0.2868188	0.3207125	0.2922147	0.2091871	0.2118695	0.2001772	0.1879010	58.1395300	53.4883700	53.4883700	60.4651200
100	1	0.9832362	0.3449878	0.5071198	0.6074663	0.3754474	0.2490674	0.3652907	0.3445351	34.0909100	50.0000000	34.0909100	38.6363600
	2	0.3645273	0.3116204	0.3539551	0.3073810	0.3291429	0.2567671	0.3011672	0.2380785	32.6087000	47.8260900	39.1304300	52.1739100
	3	0.2960865	0.3526035	0.2981843	0.3785261	0.1899727	0.1718822	0.1952693	0.1570130	62.5000000	65.0000000	57.5000000	65.0000000



Quantiles		
100,0%	maximum	2,2952
99,5%		1,9534
97,5%		0,9473
90,0%		0,5738
75,0%	quartile	0,4102
50,0%	median	0,3660
25,0%	quartile	0,3277
10,0%		0,2999
2,5%		0,2795
0,5%		0,2516
0,0%	minimum	0,2512

Moments	
Mean	0,4155882
Std Dev	0,2037997
Std Err Mean	0,0117664
upper 95% Mean	0,4387436
lower 95% Mean	0,3924328
N	300

Quantiles		
100,0%	maximum	0,49344
99,5%		0,48440
97,5%		0,43622
90,0%		0,40426
75,0%	quartile	0,36595
50,0%	median	0,32802
25,0%	quartile	0,29622
10,0%		0,26854
2,5%		0,24601
0,5%		0,22303
0,0%	minimum	0,21683

Moments	
Mean	0,3321069
Std Dev	0,0499553
Std Err Mean	0,0028842
upper 95% Mean	0,3377827
lower 95% Mean	0,3264311
N	300

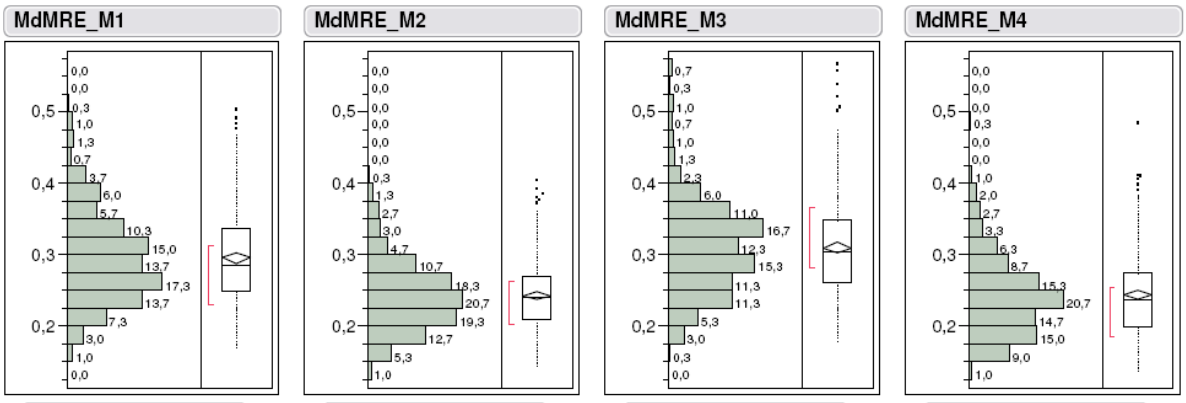
Quantiles		
100,0%	maximum	0,76058
99,5%		0,73995
97,5%		0,60657
90,0%		0,50267
75,0%	quartile	0,42964
50,0%	median	0,38857
25,0%	quartile	0,35025
10,0%		0,30887
2,5%		0,28423
0,5%		0,25954
0,0%	minimum	0,25088

Moments	
Mean	0,3977854
Std Dev	0,0774697
Std Err Mean	0,0044727
upper 95% Mean	0,4065874
lower 95% Mean	0,3889834
N	300

Quantiles		
100,0%	maximum	0,89820
99,5%		0,84582
97,5%		0,67380
90,0%		0,48791
75,0%	quartile	0,39247
50,0%	median	0,33778
25,0%	quartile	0,29650
10,0%		0,26418
2,5%		0,23482
0,5%		0,20880
0,0%	minimum	0,20291

Moments	
Mean	0,3619492
Std Dev	0,103297
Std Err Mean	0,0059639
upper 95% Mean	0,3736857
lower 95% Mean	0,3502128
N	300

Figure A.16 Analysis of MMRE distribution according to estimation models using measures M1, M2, M3 and M4.



Quantiles		
100,0%	maximum	0,50327
99,5%		0,49707
97,5%		0,46661
90,0%		0,39073
75,0%	quartile	0,33871
50,0%	median	0,28494
25,0%	quartile	0,24842
10,0%		0,22042
2,5%		0,18749
0,5%		0,16982
0,0%	minimum	0,16885

Moments	
Mean	0,2960826
Std Dev	0,0654538
Std Err Mean	0,003779
upper 95% Mean	0,3035194
lower 95% Mean	0,2886459
N	300

Quantiles		
100,0%	maximum	0,40360
99,5%		0,39747
97,5%		0,36082
90,0%		0,30848
75,0%	quartile	0,27118
50,0%	median	0,24002
25,0%	quartile	0,20957
10,0%		0,18302
2,5%		0,15990
0,5%		0,14007
0,0%	minimum	0,14002

Moments	
Mean	0,2434163
Std Dev	0,0495079
Std Err Mean	0,0028583
upper 95% Mean	0,2490413
lower 95% Mean	0,2377913
N	300

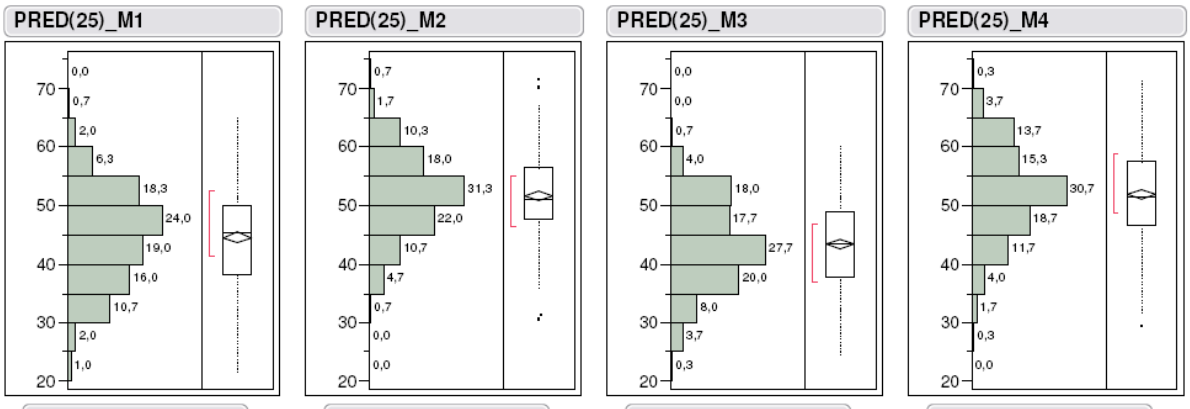
Quantiles		
100,0%	maximum	0,56662
99,5%		0,56222
97,5%		0,47944
90,0%		0,38403
75,0%	quartile	0,34946
50,0%	median	0,30639
25,0%	quartile	0,26134
10,0%		0,23079
2,5%		0,19444
0,5%		0,17942
0,0%	minimum	0,17441

Moments	
Mean	0,3106087
Std Dev	0,067516
Std Err Mean	0,003898
upper 95% Mean	0,3182798
lower 95% Mean	0,3029377
N	300

Quantiles		
100,0%	maximum	0,48389
99,5%		0,44703
97,5%		0,38446
90,0%		0,31923
75,0%	quartile	0,27442
50,0%	median	0,23851
25,0%	quartile	0,19951
10,0%		0,17480
2,5%		0,15621
0,5%		0,14032
0,0%	minimum	0,13617

Moments	
Mean	0,2437111
Std Dev	0,0578662
Std Err Mean	0,0033409
upper 95% Mean	0,2502858
lower 95% Mean	0,2371364
N	300

Figure A.17 Analysis of MdMRE distribution according to estimation models using measures M1, M2, M3 and M4.



Quantiles		
100,0%	maximum	65,217
99,5%		65,166
97,5%		60,000
90,0%		54,720
75,0%	quartile	50,000
50,0%	median	45,119
25,0%	quartile	38,095
10,0%		33,333
2,5%		27,581
0,5%		21,829
0,0%	minimum	21,429

Moments	
Mean	44,477639
Std Dev	8,1796988
Std Err Mean	0,4722551
upper 95% Mean	45,407003
lower 95% Mean	43,548274
N	300

Quantiles		
100,0%	maximum	71,429
99,5%		70,707
97,5%		64,929
90,0%		60,465
75,0%	quartile	56,494
50,0%	median	51,220
25,0%	quartile	47,500
10,0%		42,616
2,5%		37,362
0,5%		30,776
0,0%	minimum	30,435

Moments	
Mean	51,581219
Std Dev	6,9405711
Std Err Mean	0,4007141
upper 95% Mean	52,369796
lower 95% Mean	50,792641
N	300

Quantiles		
100,0%	maximum	60,465
99,5%		60,230
97,5%		57,896
90,0%		52,381
75,0%	quartile	48,876
50,0%	median	43,478
25,0%	quartile	37,778
10,0%		34,160
2,5%		29,088
0,5%		24,756
0,0%	minimum	23,913

Moments	
Mean	43,300828
Std Dev	7,2466132
Std Err Mean	0,4183834
upper 95% Mean	44,124177
lower 95% Mean	42,477479
N	300

Quantiles		
100,0%	maximum	71,429
99,5%		70,226
97,5%		66,667
90,0%		62,222
75,0%	quartile	57,411
50,0%	median	51,282
25,0%	quartile	46,550
10,0%		41,860
2,5%		36,480
0,5%		30,199
0,0%	minimum	29,268

Moments	
Mean	51,90451
Std Dev	7,7254883
Std Err Mean	0,4460313
upper 95% Mean	52,782268
lower 95% Mean	51,026752
N	300

Figure A.18 Analysis of PRED(25) distribution according to estimation models using measures M1, M2, M3 and M4.

APPENDIX B

Attempt to Validate Cost Drivers Using Historical Data

After identifying cost drivers that may significantly impact test execution effort, we run a case study to evaluate the use of the cost drivers to estimate test execution effort. The details of this study are presented next.

B.1 Planning

We planned this case study using the Goal/Question/Metric approach [24] and defining the procedures to be executed, as presented next.

B.1.1 Goals, Questions, Metrics and Hypotheses

This study has the following main goals:

G1: Verify if the identified cost drivers can improve test execution effort estimation accuracy.

G2: Compare the performance of a test execution effort estimation model when regarding and disregarding cost drivers.

The assessment of these goals are performed by answering the following research questions:

Q1: Which cost drivers significantly improve the test execution effort estimation accuracy?

Q2: How more accurate are the estimates when regarding cost drivers?

As described in the next section, we build and analyze estimation models using statistical techniques. One of these statistical techniques (stepwise regression) identifies the cost drivers that impact test execution effort with statistical significance, answering the research question Q1.

For answering the research question Q2, we analyze the estimation accuracy of the created estimation models using standard accuracy measures [129]:

MMRE: Mean magnitude of the relative error.

MdMRE: Median magnitude of the relative error.

PRED(25): Percentage of estimates that are within 25% of the actual values.

The statistical significance of the observed estimation accuracy is analyzed through the test of the following statistical hypotheses:

1) The mean magnitude of the relative error is smaller when regarding cost drivers (*reg*) than when disregarding them (*disreg*).

$$H_{01} : MMRE_{reg} \geq MMRE_{disreg}$$

$$H_{11} : MMRE_{reg} < MMRE_{disreg}$$

2) The median magnitude of the relative error is smaller when regarding cost drivers than when disregarding them.

$$H_{02} : MdMRE_{reg} \geq MdMRE_{disreg}$$

$$H_{12} : MdMRE_{reg} < MdMRE_{disreg}$$

3) The percentage of estimates that are within 25% of the actual values is larger when regarding cost drivers than when disregarding them.

$$H_{03} : PRED(25)_{reg} \leq PRED(25)_{disreg}$$

$$H_{13} : PRED(25)_{reg} > PRED(25)_{disreg}$$

B.1.2 Historical Data Analysis

In this study, we analyze the historical data of the industry on the mobile application domain that sponsored the survey described in Section 5.6. The analyzed database had approximately six-month data containing information about the execution of more than 300 test suites, which means the execution of thousands of different test cases by more than 30 testers of different test teams.

To achieve our goals, this study is organized in three parts. In the first part, we analyze and prepare the data, removing inconsistencies and some outliers, as well using expert opinion to gather missing values. Then, the second part of this study uses stepwise regression (SWR) [53] to find the best set of cost drivers (predictors) to explain the variation in the test execution effort (response).

SWR is a statistical technique that builds a regression model, which is a mathematical equation relating independent (predictors) and dependent (response) variables. We use the forward and backward method, that is, the regression model is created by adding and possibly removing independent variables one-at-a-time until some stopping rule is satisfied. In each step of this analysis, a variable can be added to the model if it is considered statistically significant.

Also, variables can be removed from the model if they become nonsignificant after the inclusion of other variables.

After that, the last part of this study performs a cross-validation analysis to evaluate and compare the accuracy to estimate test execution effort when using the set of selected cost drivers and when disregarding it. Cross-validation is a method for generalizing the results of a model evaluation. The main idea is to partition a sample of data into folds (subsets) such that you test the model on a single fold, while the other folds are used to build the model. The use of this method reduces the probability of obtaining results by chance and enable us to test our statistical hypotheses.

B.2 Execution

In the next subsections, we detail the main activities executed during this study and their results. To avoid bias during the presentation of these results, our interpretation of them are presented later, in Section B.3.1.

B.2.1 Gathering Historical Data

In the historical database, each record stored information about the executed tests, tested product, testers and related information. However, the information about most of the identified cost drivers were not stored, since they were not being used at that time. For this reason, we had to gather information about the cost drivers in the analyzed period of time.

We analyzed the possibility to obtain past information for each cost driver described in Table 5.18. After talking to test experts, managers and people that maintain the infrastructure (test environment, test tools, etc.), we verified that we were able to obtain reliable past information only for the cost drivers presented in Table B.1. Information about other cost drivers were not completely available, too costly to be obtained or only indirectly available through some of the variables presented in Table B.2. We used these variables related to some cost drivers in an attempt to capture some effect of them.

Our first task to obtain past information about cost drivers was to organize and consolidate the data to identify which tests were executed during the considered period, who executed the tests, what phones were tested and so forth. After that, we analyzed each cost driver individually, as shown next.

Since we did not have access to all phone performance specifications, we invited seven experienced testers to assess phone performance using their previous experience in testing them. We adapted the Delphi [81] and Card Sorting [47] techniques to reach a consensus about the performance of the phones tested during the analyzed period.

We asked seven participants to classify the phone performance into three levels (low, average and high). They received some slips of paper, each one containing the model of one phone. Then, we run some trials in which they had to group the phones according to the three levels we defined. At the end of each trial, we removed the slips of the phones which performance achieved a consensus. We verified that this process was very fast and not boring for the participants. After three trials, we achieved the consensus about the performance of the phones tested

Table B.1 Cost drivers obtained by analyzing past projects information and expert opinion.

Description	Abbreviation	Scale	Range or Scale points	Source of information
Phone performance	PhonePerf	ordinal	Low, Avg, High	Expert assessment.
Quality of the test cases	QualityTC	ordinal	Low, Avg, High	Expert assessment.
% of Testers with Average experience	ExpAvgPerc	ratio	[0, 100]	Historical records of training and team allocation.
% of Testers with High experience	ExpHighPerc	ratio	[0, 100]	Historical records of training and team allocation.
% of Testers with Average English skills	EngAvgPerc	ratio	[0, 100]	English class level and team allocation.
% of Testers with High English skills	EngHighPerc	ratio	[0, 100]	English class level and team allocation.

in the past.

We used the same approach to reach a consensus about the quality of the test cases. As the tests related to each family of products have similar quality, the participants classified the tests of each family of products in low, average or high. Also, the english skill of the testers were assessed based on their current level of english class. Regarding the tester experience in testing, we determined it based on their job start date.

B.2.2 Stepwise Regression

The stepwise regression analysis have some assumptions that must be satisfied. First, the relation between independent and dependent variables is expected to be linear. Also, it is assumed that the residuals (predicted minus actual values) are normally distributed. We verified that the numerical variables used in this study were not normally distributed. For instance, a simple linear regression analysis between test execution effort and execution points built a model with residuals not normally distributed. For this reason, we transformed these numerical variables into a natural logarithmic scale using the box-cox transformation technique [53] [87] in order to approximate the data to normal distributions.

For each numerical variable that required data transformation we created a new variable using the letter L and the original variable name. For instance, we created the variable $LEffort$ for storing the transformed data of the variable $Effort$. After all this process, we verified that the linear correlation between the transformed effort data ($LEffort$) and the other transformed data from independent variables were improved.

In this study, some variables considered have ordinal scales, such as the tested applica-

Table B.2 Other related variables available in historical databases.

Description	Abbreviation	Scale	Range or Scale points	Related cost drivers
Effort to execute the tests	Effort	ratio	Positive numbers	-
Number of execution points	EP	ratio	Positive numbers	-
Number of tests	NumOfTests	interval	Positive integers	-
Name of the tested product	Product	nominal	A, B, C, ..., M, N, O	Stability
Product Family of the tested product	ProductFamily	nominal	F1, F2, F3, F4, F5, F6, F7	Performance
Technology of the tested product	Tech	nominal	T1, T2	Feature Experience

tion, the model of the tested hardware and the tester's language skill. For including them in the regression model, we created $(n-1)$ dummy variables [53] for each of these categorical variables, where n is the number of different values (scale points) that the variable can take. For instance, the variable *Product* requires the creation of 14 dummy variables (*Product_A* to *Product_N*) for representing 15 different products (from letter A to O). The dummy variables can take the values 0 (not present) and 1 (present). For example, the variable *Product_A* takes value 1 only when A is the tested product. When variables *Product_A* to *Product_N* take the value 0, it means that the tested product is O.

After that, we run the stepwise regression with α = 0.15 to add and to remove variables, the most common used values in the practice. We verified that some of the selected independent variables were high correlated (multicollinearity problem). For instance, the independent variables number of tests and execution points are highly correlated. However, a high degree of multicollinearity produces large variance in the regression coefficient estimates.

To avoid multicolinearity, we calculated the Pearson's correlation coefficient for measuring the linear relationship between the independent variables and also between independent and dependent variables. Hence, when the SWR selected two independent variables with high linear correlation, we discarded the one less correlated with the dependent variable (effort) and repeated the stepwise regression analysis. In this way, we run the stepwise regression until all selected independent variables were not highly correlated between them.

The SWR selected 5 variables with α at 0.15. However, after running a multiple regression analysis (independent variables are fixed), we verified that one of these variables (*LEngSkillAvPerc*) were not significative with α at 0.05. We discarded this variable and then used the multiple regression analysis to build the following equation:

$$LEffort = -3.91 + 1.06LEP - 0.0586LExpHighPerc$$

$$-0.0989 ProductFamilyD + 0.08 ProductB \quad \text{(B.1)}$$

The output of the multiple regression analysis is presented next.

```
Predictor            Coef   SE Coef       T      P
Constant          -3,9000    0,2249  -17,34  0,000
LEP                1,05855   0,02390   44,29  0,000
LExpHighPerc    -0,057588  0,009053   -6,36  0,000
ProductFamilyD   -0,09927   0,02318   -4,28  0,000
ProductB          0,08038   0,01915    4,20  0,000


S = 0,139850   R-Sq = 86,8%   R-Sq(adj) = 86,6%


Analysis of Variance

Source           DF      SS      MS       F      P
Regression        4  40,361  10,090  515,91  0,000
Residual Error  314   6,141   0,020
Total           318  46,502


Source            DF  Seq SS
LEP                1  38,831
LExpHighPerc       1   0,622
ProductFamilyD     1   0,564
ProductB           1   0,345
```

As we can see, the adjusted R^2 achieved was 86.6%. We confirmed the regression analysis assumptions and the model stability by:

- Using the normal probability plot in Figure B.1, the residual plot to in Figure B.2 and other graphs to verify if the residuals were random and normal.
- Calculating the Cook´s distance values [87] to identify influential data points. Twelve test project with distance higher than 3 x (4/n) were removed to test the model stability. After running the regression analysis with the new data set, the coefficients remaining stable and the adjusted R^2 was improved in almost 1%.

Then, we transformed back the regression equation to the following equation:

$$Effort = EP^{1.06}(ExpHighPerc + 1)^{-0.0586} \\ e^{(-3.91 + 0.0989 ProductFamilyD + 0.08 ProductB)} \quad \text{(B.2)}$$

B.2.3 Cross-validation Analysis

The last part of this study is the cross-validation analysis. We run a 10-fold cross-validation, that is, we partition the data into 10 folds. Then, each of these folds are used to test the models (test set) built using the others folds (training set). In each iteration of the cross-validation analysis, we run the multiple regression analysis to build two models based on the training data set:

Figure B.1 Normal probability plot for the regression model.

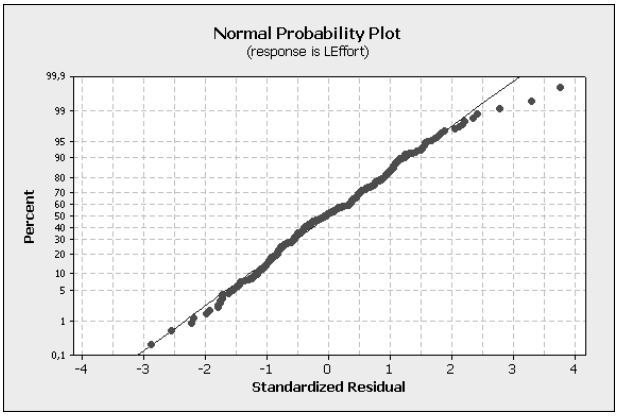
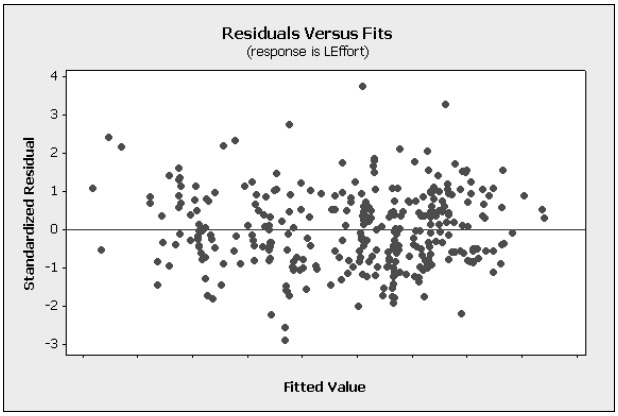


Figure B.2 Residual plot for the regression model.



Model-1: Model that regards cost drivers. This model consider all the variables selected by the stepwise regression analysis.

Model-2: Model that does not regard cost drivers. This model only consider the independent variable *LEP*, since it only represents the size and complexity of the tests to be executed.

After that, we use the test set to evaluate the accuracy of both models using the measures defined in Section B.1.1. We repeated the cross-validation analysis three times with different partitioning. Table B.3 summarizes the mean accuracy achieved by each model in the 30 iterations.

Table B.3 Mean of the MMRE, MdMRE and PRED(25) observed in the cross-validation analysis.

Estimation model	MMRE	MdMRE	PRED(25)
With cost drivers	0,1169	0,1015	92,48%
No cost drivers	0,1171	0,0906	90,59%

Finally, we tested our statistical hypotheses using the paired t-test with α at 0.05. In all cases, the test did not found enough evidence to reject the null hypotheses (all p-values were larger than 0.05).

B.3 Final Considerations

B.3.1 Interpretation of Results

During the stepwise regression, the achieved adjusted R^2 was greater than 70%, which is considered good, but yet smaller than the ideal (close to 95%). This means that there are missing cost drivers in the model. Indeed, this result was expected due the lack of historical data about them. The normal probability and the residual plots shows that the residuals are normally distributed and with constant variance, as assumed by the regression analysis.

Observing the equation generated by the regression, we can see that *LEP* is the most important variable due to its highest coefficient and range of values that it can take. This result was also expected, since the project size is usually the variable with more impact on project effort. However, the coefficient of the other variables are too small when compared to the *LEP* coefficient. Also, the range of values of the others variables are also smaller, leading to a small impact on the effort estimates. This fact means that the variables did not have strong relationship with the cost drivers that are affecting the effort.

On the other hand, the signs used in the coefficients of each variable are intuitively consistent. Basically, the effort increases with the size of the test project (*LEP*) and if the tested product is B (a newer product with low stability). Also, the effort decreases with the percentage of high experienced testers and when the tested product is from the family D (high performance).

With respect to the cross-validation, the accuracy achieved by both models were similar. This fact was confirmed by the statistical hypotheses tests, which did not found any statistical significant difference between the accuracy of the two models.

B.3.2 Threats to Validity

Construct validity: In this case study, we faced with the lack of historical data about the identified cost drivers. We used measures that can be indirectly related to the cost drivers under investigation. However, some of these variables may not have the expected relation with the cost drivers.

Internal validity: The use of the testers to create some missing values based on expert opinion may bias the results, since they may feel that their productivity are under evaluation too. To control this threat, we carefully explained the goals of the study.

Some variables in the historical database were used as indirect measures for cost drivers. These variables may not represent a causal relationship with the response variable (effort). To control this threat, we reviewed the variables selected by the stepwise regression analysis.

Conclusion validity: We were not able to support our hypothesis that the use of cost drivers related to testing improves the estimation accuracy of test execution effort. One of the possible causes is the missing of the relationships between the response (effort) and cost drivers which impact were not captured by none of the independent variables.

External validity: The findings of this study may not be generalized due to the lack of information about the cost drivers. Others studies are required to achieve more representative results.

APPENDIX C

Test Execution Effort Estimation Tool

This appendix 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. This tool is also discussed in [20].

C.1 Functionalities

This section presents the main functionalities provided by our tool, which uses the model described in [12] and [15] 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 C.1 presents an example of 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

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

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 [12].

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 [81]. Figure C.2 shows an example of the configuration of system characteristics for the mobile application domain.

C.2 Tool Architecture

We developed the tool in Java using the architecture shown in Figure C.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.

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.

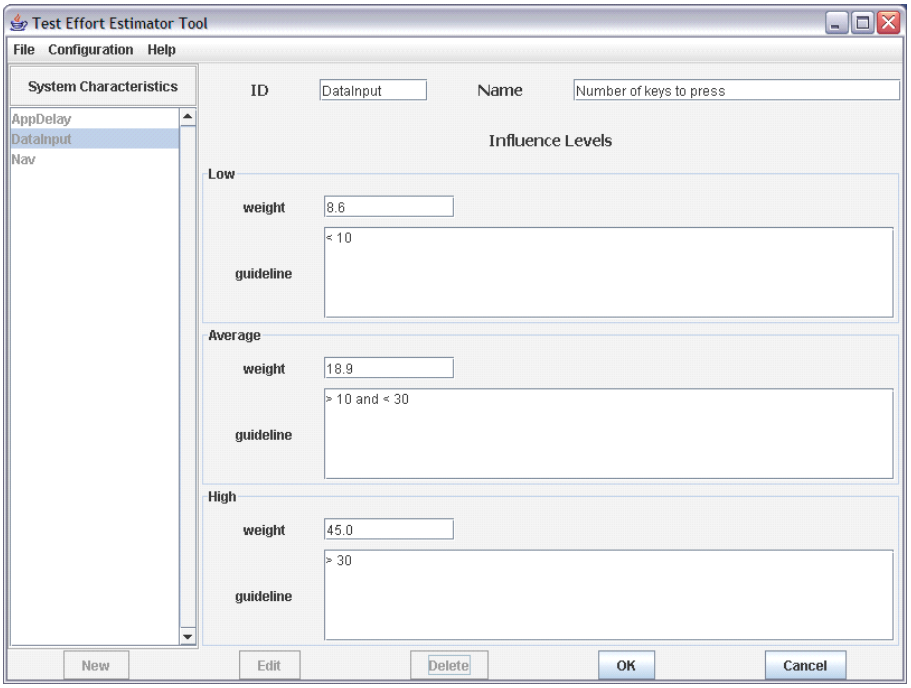


Figure C.2 Configuration of the characteristics used to measure test size and execution complexity.

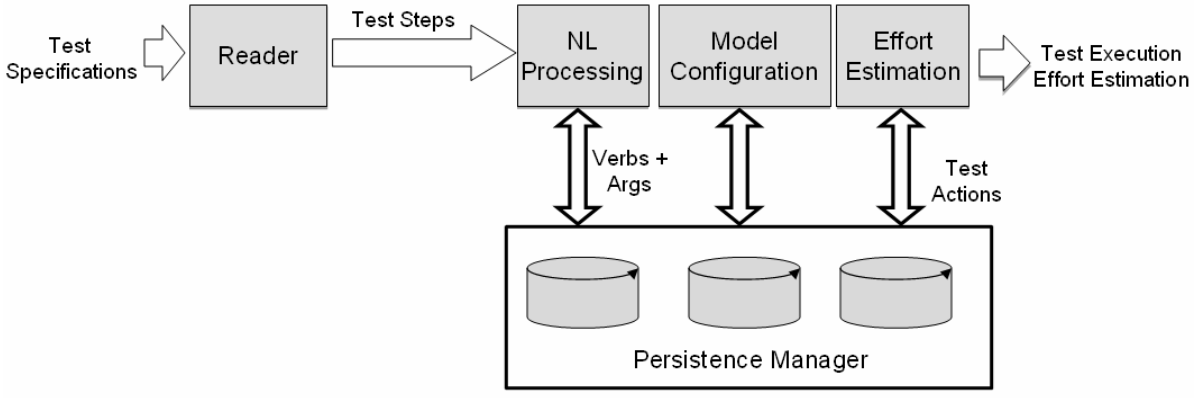


Figure C.3 Architecture of the automated test execution effort estimation based on test specifications.

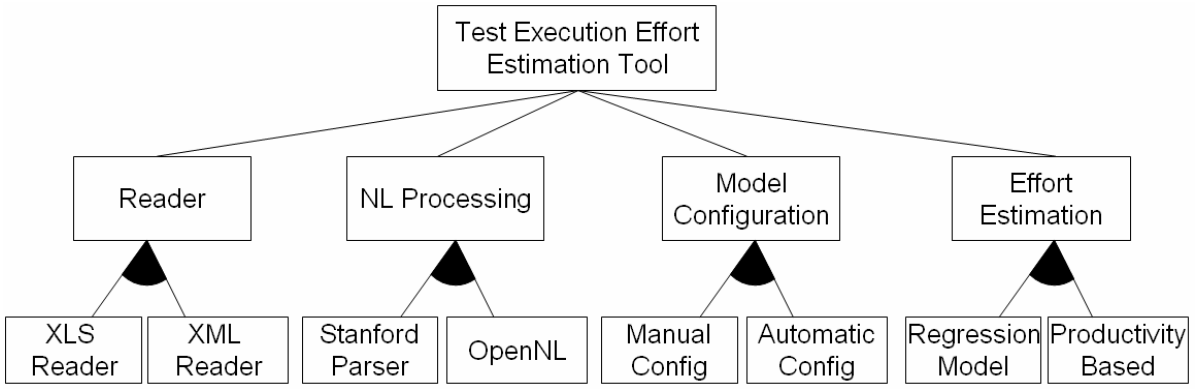


Figure C.4 Architecture of the automated test execution effort estimation based on test specifications.

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.

C.2.1 Feature Model

The presented architecture makes easy the customization of our tool, according to the customer needs. The feature model [120] presented in Fig. C.4 shows the possible extensions of each component of the tool architecture.

We implemented this as follows. The Reader component uses the Apache POI API [1] to read test specifications from Microsoft Excel files. The Natural Language Processing uses the Stanford Parser API [2] 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.

C.3 Final Considerations

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 used to support our empirical studies, 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) [15].

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.

APPENDIX D

ManualTEST: A Tool for Collecting Manual Test Execution Data

In this appendix, we present ManualTEST [17], a tool that we developed to improve the accuracy of the data collected in our empirical studies and to provide information to locate and identify sources of problems occurred during the execution of these studies.

In addition, ManualTEST can improve the accuracy of the collected data, providing information to locate and identify sources of problems occurred during the execution of these studies. ManualTEST was successfully used in several studies run in two real test sites that provide testing services for a major mobile phone manufacturer.

D.1 Manual Test Execution

In empirical studies involving manual test execution, it is necessary to have at least one or more of the following components: tester, product under tested, test specification and test environment. In a manual test execution, a tester reads and executes each step of a test specification using a product under test and a test environment.

The test specifications are usually described in terms of pre-conditions (initial conditions to start the test), procedure (list of test steps with inputs and expected outputs) and post-conditions [68]. In addition, they can contain more information, such as a test description, test objectives, the tested requirements and a software and hardware setup procedure. These specifications are commonly written in natural language, as shown in Figure D.1.

The data collected during manual test execution depend on the objective of the study, but the most common collected data are:

- Test id.
- Tester id.
- Date and time in which the test execution was started.
- Test execution time, which is the time spent to execute the test.
- Test result.

In general, test result indicates if the test was executed successfully (passed), if a bug was detected (failed) or if the tester could not execute the test (blocked) due to some constraint, such as lack of resource, test not applicable to the product under test, etc.

Test id: TaRGeT_0-1

Description: TaRGeT_TC_1

Objectives: Test the chronometer of the ManualTEST tool.

Requirements:
RQ_05

Setup:
None.

Initial conditions:
a) Tool should be started.
b) Tool is in the Test Selection perspective.
c) At least one test is shown in the Test Plans View.

Notes: Test case auto-generated by TaRGeT system.

Test procedure:
1) Double-click a test case X.
» The test specification is shown in the view "Selected TC". A new view X is created for the test case X.

2) Change perspective to Test Execution and inform your tester id.
» The tool is now in the Test Execution perspective. The view "Selected TC" is not opened. The view X is presented.

3) Click on button Play.
» The chronometer start counting time.

4) Click on button Pause.
» The chronometer stop counting time.

5) Click on button Play.
» The chronometer resume the counting of time.

6) Click on button Cancel.
» The counted time is discarded.

Final conditions:
None.

Cleanup:
None.

Figure D.1 Sample test specification written in natural language.

During the manual test execution in an empirical study, some problems can occur and threat the validity of the data analysis:

P1. While executing the tests, testers have to start counting test execution time just before the beginning of the test execution activity and to stop just after finishing it. The tester can forget to start or stop the chronometer, including between test executions, reducing the accuracy of the collected data.

P2. When running studies in industrial settings, participants may be interrupted by important phone calls or by other emergencies [21]. The time spent during these interruptions should not be taken into account.

P3. These interruptions should also be recorded to verify if they impacted the study [21]. In

our context, these interruptions can impact the total test execution time and the tester performance.

P4. From previous experiences [15], we observed that the effect of some confounding factors can be hard to detect, such as changes in network bandwidth or in other dynamic environmental conditions. These problems are usually hard to avoid when considering studies in industrial settings, even when running supervised experiments.

P5. For some studies, such as test prioritization based on data reuse [80], it is important to distinguish between test procedure time, test setup time, etc., making difficulty the data collection.

To solve these problems, we developed and used a tool to collect manual test execution data, as described in the next sections.

D.2 Manual Test Execution assiSTant (ManualTEST)

This section presents the main functionalities of ManualTEST, a tool developed to automate the collection of manual test execution data, avoiding the problems previously described. The tool was developed using the Eclipse Rich Client Platform (RCP) [88] and have two different perspective, one to select the tests to be executed and another one to support the test execution activity. Next, we present the details of these two perspectives and information about the data collected by the tool.

D.2.1 Test Selection

To automatically read test specifications, ManualTEST consider that these specifications are stored in spreadsheets files. In addition, the tool can be customized to read spreadsheets in different formats. Once a spreadsheet file is opened, the tests are listed to the user, as shown in Figure D.2. Then, the tester must double-click to open the tests that she is going to execute.

D.2.2 Test Execution

When the tester selects the Test Execution perspective, only the selected tests are presented. A Test Execution Controller view is also presented, as shown in Figure D.3. In this view, the play and the pause buttons are presented to the tester to control the counting of time. This integrated functionality make easy to the tester to start, pause and resume the test execution time (problem P1 and P2).

When the tester pushes the play button, the chronometer is activated and the first line of the test specification is highlighted. Using keys $\uparrow$ and $\downarrow$ from the keyboard, the tester executes the highlighted test step and go to the next one or go back to the previous one, as presented in Figure D.4. This functionality will ensure that the tester will read and execute each test step. Also, the time spent in each step is properly recorded (problem P5). For pausing the

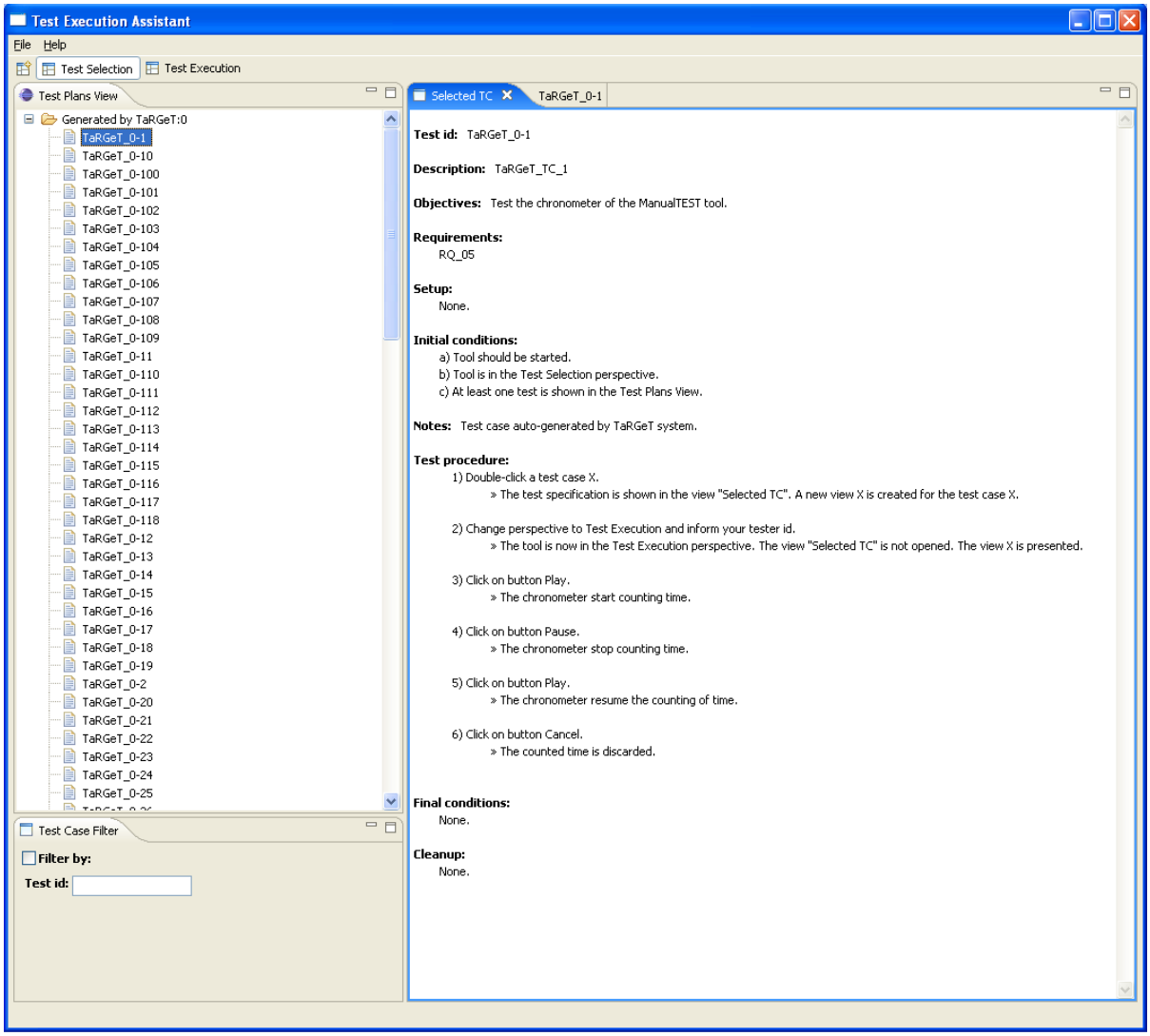


Figure D.2 Test selection perspective of ManualTEST.

chronometer, the tester should push the pause button and push the play button later to resume the time counting (problem P2).

ManualTEST counts the time based on three stages: Setup, Execution and Debug. The Setup stage is relative to the execution of steps required to build the test pre-conditions. The Execution stage is relative to the execution of the test procedure (execution of test steps and verification of expected results). Finally, the Debug stage is used for recording the time spent by the tester for confirming that a test failed. For instance, the tester may have to execute the test again to confirm the test failure or s/he may have to read the specification to confirm that the problem is a defect in the application and not a problem in the test specification. This approach is interesting when the study analyzes only one of these stages (problem P5). The total execution time is also presented by the tool.

By default, the initial test execution stage considered by the tool is Setup. To change the

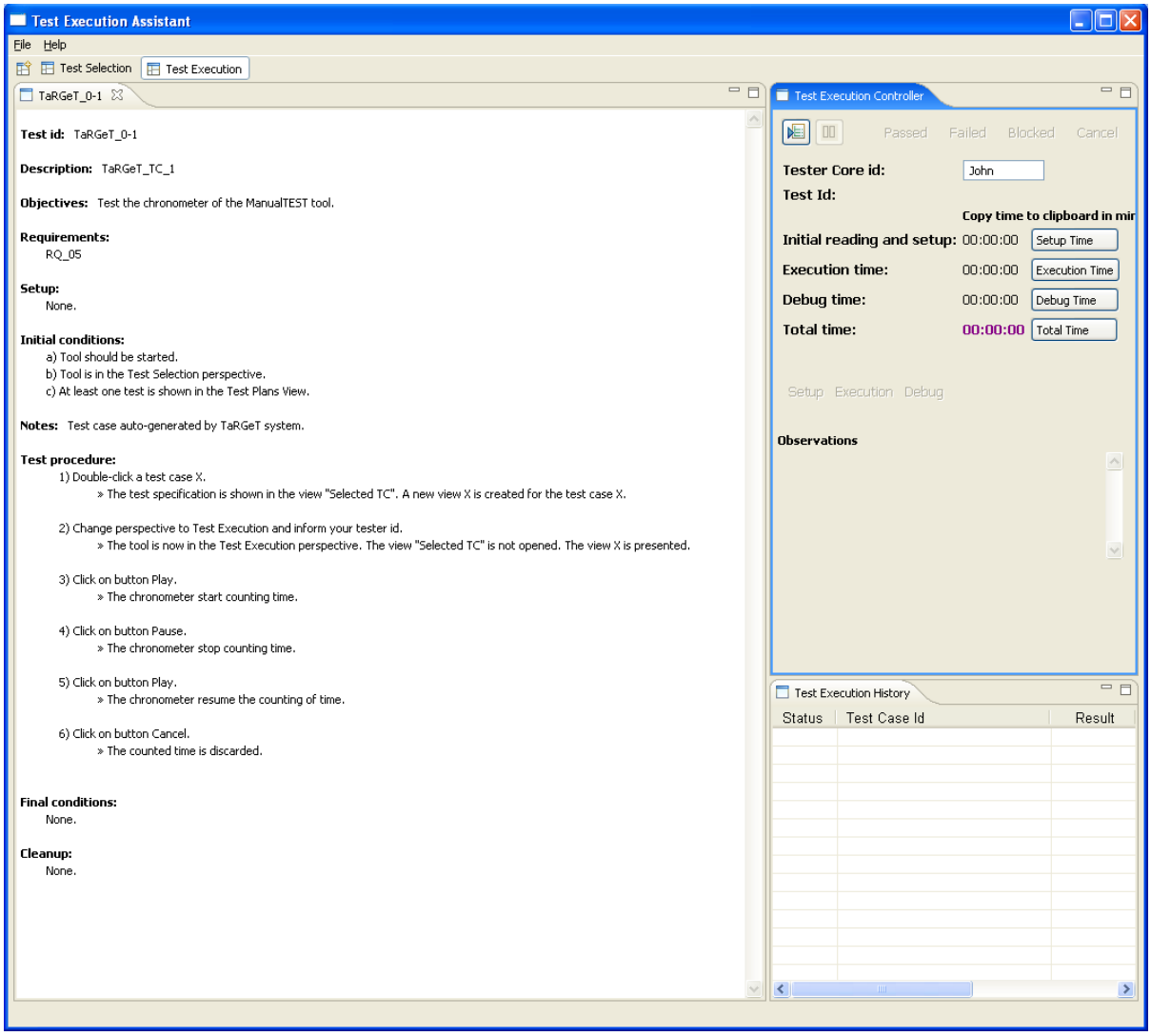


Figure D.3 Test execution perspective of ManualTEST.

stage, participants of the study can use the Setup, Execution or Debug buttons provided in the Test Execution Controller view. However, the tool automatically detects when the cursor is on the "Test procedure:" line and automatically consider the change from test setup to procedure execution, improving data collection accuracy. The test setup time, procedure execution time, debug time and total test execution time are presented to the tester in the Test Execution Controller view.

The field observations can be used to describe any problem occurred during the execution of the test. To finish the execution of a test case, the tester should press in button Passed, Failed or Blocked to indicate the correct test result. The Cancel button will discard the data collected for the current test execution.

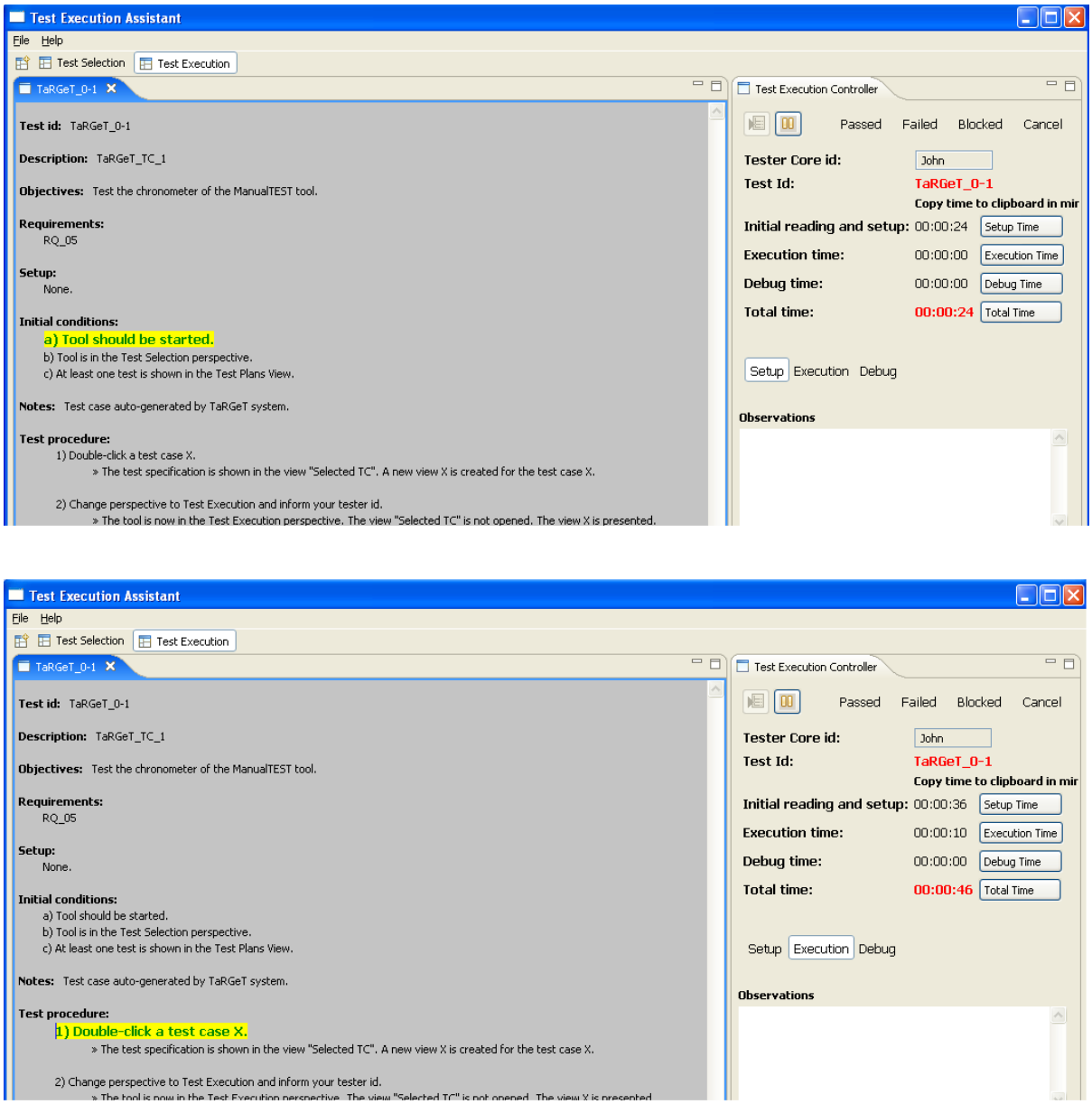


Figure D.4 ManualTEST at different moments: test step under execution is highlighted and time is automatically counted as setup time or procedure execution time.

D.2.3 Collected Data

As test execution is finished, ManualTEST stores the test result, execution times and other related information in two different spreadsheet files. The first one is presented in Figure D.5, which is basically the general information recorded for each executed test.

	A	B	C	D	E	F	G	H	I	J
1	Date	Time	Tester	Test	Result	Setup	Execution	Debug	Total time	
2	10/6/2008	19:47:44	John	TaRGeT_0-1	PASSED	00:00:29	00:00:51	00:00:00	00:01:20	

Figure D.5 Test result for a single test case.

The second spreadsheet file contains detailed information about the test execution. As presented in Figure D.6, ManualTEST stores the time spent to execute each step of the test specification. This characteristic is very important to investigate the presence of aberrant values in outlier analyses (problems P3 and P4) and to identify sources of variations on test execution time.

K	L
Step	Time
Description: TaRGeT_TC_1	00:00:03
Objectives: Test the chronometer of the ManualTEST tool.	00:00:00
Description: TaRGeT_TC_1	00:00:00
Objectives: Test the chronometer of the ManualTEST tool.	00:00:04
a) Tool should be started.	00:00:06
b) Tool is in the Test Selection perspective.	00:00:06
c) At least one test is shown in the Test Plans View.	00:00:09
Notes: Test case auto-generated by TaRGeT system.	00:00:01
Test procedure:	00:00:00
1) Double-click a test case X.	00:00:04
» The test specification is shown in the view "Selected TC". A new view X is created for the test case X.	00:00:04
2) Change perspective to Test Execution and inform your tester id.	00:00:04
» The tool is now in the Test Execution perspective. The view "Selected TC" is not opened. The view X is presented.	00:00:06
3) Click on button Play.	00:00:03
» The chronometer start counting time.	00:00:06
4) Click on button Pause.	00:00:04
» The chronometer stop counting time.	00:00:03
5) Click on button Play.	00:00:04
» The chronometer resume the counting of time.	00:00:03
6) Click on button Cancel.	00:00:03
» The counted time is discarded.	00:00:07

Figure D.6 Detailed test result includes time spent in each test step.

D.3 Advantages and Current Limitations of ManualTEST

We used ManualTEST in several studies carried out by researchers from a research group in testing [136]. These studies involved manual test execution and some of them are overviewed next.

- A case study and a controlled experiment designed to compare the time reduction for manual test setup provided by different test case prioritization techniques.

- A case study to compare functional tests generated manually or automatically by a model-based testing tool [109] with respect to their manual test execution time.
- Controlled experiments designed to analyze the relationship of different test size measures with manual test execution time.
- Controlled experiments designed to investigate the effect of cost drivers for manual test execution.
- Case studies to verify if code instrumentation has impact on manual test execution time when considering mobile phone applications.

These studies were run in industrial settings and most of them were carried out by researchers different from the authors of the tool. The researchers were asked to write the benefits and limitations of ManualTEST observed during the execution of their empirical studies. We received several feedbacks showing the advantages and limitations of ManualTEST. These feedbacks are summarized next.

- Automated test execution time collection with the click of few buttons.
- The integration of the chronometer and test specifications increased productivity and reduced the occurrence of problems (forgetting to start or pause the time, etc.). Also, this integration make easy to the researcher to monitor more than one tester at the same time.
- Detailed log files included all information required for data analysis. Due to the high level of detail, the effect of some confounding factors were detected and outliers were treated properly.
- With the detailed and automated data collection, it was also possible to study the impact of each test step on total execution time, including an analysis of the main sources of variation.
- The format of the log files was not easy to be understood by other researchers.
- There are possible improvements to implement and minor defects to correct in the tool, improving its usability.
- The available version of ManualTEST did not guide the tester through the correct sequence of tests to be executed, requiring a careful attention of testers and researchers. The implementation of this functionality can improve the benefits of the tool.
- After some interruptions, some testers forgot to resume the chronometer (push the the play button), requiring the intervention of the researcher that was monitoring the test execution. The tool can be improved by having warning messages blinking while the chronometer still paused.
- To write into the field Observations, the time should be paused by the tester. It would be interesting if the tool could do that automatically.

As we can see, ManualTEST can be used to avoid several problems during the execution of empirical studies related to manual test execution. However, its benefits depend on the correct use by the testers. For instance, the tester may still forget to push the play and pause buttons. As suggested in the feedbacks, some improvements in the tool should be done to avoid the reported problems.

D.4 Related Tools

Several testing tools are available in the marketing. For manual test authoring and execution, there is a tool that can be found in [55]. This commercial tool provides functionalities to execute and mark some steps of the test (comparisons and other verifications) as they are executed, as well as the storage of test results and test execution time. For collecting data in empirical studies, our tool presents better benefits due to characteristics such as the more detailed data collection and the highlighting of the test step being executed.

Some other researchers also have developed data collection tools for supporting their empirical studies. In [70], the authors reported a tool under development for collecting data during software engineering experiments. They were not interested in execution time, but in data concerning subjects, interactions between subjects and technology and changes in engineering artefacts. In [69], the authors proposed an unobtrusive method of collecting feedback from subjects during an experiment. They developed a tool to collect the feedbacks from experimental subjects and they identified several benefits of using the tool during the execution of four experiments, such as its use for validating the data obtained from other sources, checking process conformance and identifying problems with the experiments.

In [21], the authors developed a web-based support environment for planning and running software engineering experiments. One of its functionality is the collection of the experiment results, such as the answers of web questionnaires. The authors reported common interruptions (phone calls, lunch break, etc.) that occurred during the experiments run in real environments. In their approach, the subjects had to report the nature and time span of the interruptions. In our approach, the subject has to pause and resume the chronometer.

D.5 Final Considerations

This paper presented the main functionalities of ManualTEST, a tool developed for improving the collection of manual test execution data. This tool not only helps to collect data accurately, but it also provides information in different levels of detail, supporting the identification of problems in the collected data and analyses of outliers, sources of variability, etc. These benefits were observed during the execution of several case studies and controlled experiments in two real test sites. We intend to evolve the tool in order to overcome the current limitations and include the suggestions received during the empirical studies.

Despite of the use of ManualTEST for empirical studies, the tool can also be used to support the manual test execution activities. Its use can help, for instance, to have more accurate historical data in industrial settings. Although our tool is structured for experiments related to

manual test execution, we believe that it can be extended (or similar ones can be developed) to support data collection for other types of manual activities that can have problems similar to those reported here. Finally, the benefits observed during the use of ManualTEST justified the cost to develop the tool, which took approximatelly two months of work (partial time) of one experienced developer.

Bibliography

[1] Apache POI: Java API to access microsoft format files, 2007. http://poi.apache.org/.

[2] The Stanford parser: A statistical parser, 2007. http://nlp.stanford.edu/software/lexparser. shtml.

[3] Tukutuku project, 2007. http://www.metriq.biz/tukutuku/.

[4] Alain Abran and Pierre N. Robillard. Function points: a study of their measurement processes and scale transformations. *Journal of Systems and Software*, 25(2):171–184, 1994.

[5] Alain Abran and Pierre N. Robillard. Function points analysis: An empirical study of its measurement processes. *IEEE Trans. Softw. Eng.*, 22(12):895–910, 1996.

[6] Mohammed Abdullah Al-Hajri, Abdul Azim Abdul Ghani, Md Nasir Sulaiman, and Mohd Hasan Selamat. Modification of standard Function Point complexity weights system. *Journal of Systems and Software*, 74(2):195–206, 2005.

[7] Allan J. Albrecht. Measuring application development productivity. *Proc. IBM Applications Development Symp.*, 1979.

[8] Allan J. Albrecht. Ad/m productivity measurement and estimate validation-draft. *IBM Corp. Information Systems and Administration, AD/M Improvement Program*, 1984.

[9] Allan J. Albrecht and John E. Gaffney. Software functions, source lines of code, and development effort prediction: A software science validation. *IEEE Trans. Software Eng.*, 9(6):639–648, November 1983.

[10] Eduardo Aranha. Estimating test execution effort based on test specification. In *Fourth edition of the International Summer School on Software Engineering*, September 2007. Student talk.

[11] Eduardo Aranha and Paulo Borba. Considering test execution complexity for estimating test execution effort. In *1st International Doctoral Symposium on Empirical Software Engineering (IDoESE 2006)*, September 2006.

[12] Eduardo Aranha and Paulo Borba. Measuring test execution complexity. In *2nd Intl. Workshop on Predictor Models in SE (PROMISE 2006), co-located with the 22nd IEEE Conference on Software Maintenance (ICSM'06)*, September 2006.

[13] Eduardo Aranha and Paulo Borba. Test execution effort and capacity estimation. In *17th IEEE International Symposium on Software Reliability Engineering (ISSRE 2006)*, November 2006.

[14] Eduardo Aranha and Paulo Borba. Empirical studies of test execution effort estimation based on test characteristics and risk factors. In *2nd International Doctoral Symposium on Empirical Software Engineering (IDoESE 2007)*, September 2007.

[15] Eduardo Aranha and Paulo Borba. An estimation model for test execution effort. In *1st International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*, pages 107–116, September 2007.

[16] Eduardo Aranha and Paulo Borba. Test effort estimation models based on test specifications. In *Testing: Academic & Industrial Conference - Practice And Research Techniques (TAIC PART 2007)*, September 2007.

[17] Eduardo Aranha and Paulo Borba. Manualtest: Improving collection of manual test execution data in empirical studies. In *5th Empirical Software Engineering Latin-american Workshop (ESELAW 2008)*, Salvador, Brazil, November 2008.

[18] Eduardo Aranha and Paulo Borba. Estimating manual test execution effort and capacity based on execution points. *International Journal of Computers and Applications, special issue on the International Summer School on Software Engineering*, 31(3), 2009.

[19] Eduardo Aranha, Paulo Borba, and Jose Lima. Model simulation for test execution capacity estimation. In *17th IEEE International Symposium on Software Reliability Engineering (ISSRE 2006)*, November 2006.

[20] Eduardo Aranha, Filipe de Almeida, Thiago Diniz, Vitor Fontes, and Paulo Borba. Automated test execution effort estimation based on functional test specifications. In *Brazilian Symposium on Software Engineering (SBES'08)*, 2008.

[21] Erik Arisholm, Dag I. K. Sj, Gunnar J. Carelius, and Yngve Lindsj. A web-based support environment for software engineering experiments. *Nordic J. of Computing*, 9(3):231–247, 2002.

[22] Rajiv D Banker, Hsihui Chang, and Chris F Kemerer. Evidence on economies of scale in software development. *Information and Software Technology*, 36(5):275–282, May 1994.

[23] Rajiv D. Banker, Robert J. Kauffman, and Rachna Kumar. An empirical test of object-based output measurement metrics in a computer aided software engineering (case) environment. *Journal of Management Information Systems*, 8(3):127–150, 1992.

[24] Victor Basili, Gianluigi Caldiera, and Dieter Rombach. The goal question metric approach. *Encyclopedia of Software Engineering*, 1:528–532, 1994.

[25] Kent Beck and Martin Fowler. *Planning Extreme Programming*. Addison-Wesley Professional, 2000.

[26] Barry Boehm. *Software Engineering Economics*. Prentice Hall, 1981.

[27] Barry Boehm, Chris Abts, and Sunita Chulani. Software development cost estimation approaches - a survey. *Annals of Software Engineering*, 10:177–205, 2000.

[28] Barry Boehm, Ellis Horowitz, Ray Madachy, Donald Reifer, Bradford Clark, Bert Steece, Winsor Brown, Sunita Chulani, and Chris Abts. *Software Cost Estimation with COCOMO II*. Prentice Hall, 2000.

[29] Barry Boehm and Ali Afzal Malik. System and software sizing tutorial. In *22nd International Forum on COCOMO and Systems/Software Cost Modeling*, 2007.

[30] Barry Boehm, Donald J. Reifer, and Ricardo Valerdi. Cosysmo: A systems engineering cost model. In *1st Annual Conference on Systems Integration*. Stevens Institute of Technology Campus, 2003.

[31] Bruce Bowerman and Richard O'Connell. *Linear Statistical Models, an Applied Aproach*. PWS-KENT, 2nd edition, 1990.

[32] Lionel Briand, Khaled El Emam, and Sandro Morasca. On the application of measurement theory in software engineering. *Empirical Software Engineering: An International Journal*, 1(1):61–88, 1996.

[33] Lionel C. Briand, Khaled El Emam, Dagmar Surmann, Isabella Wieczorek, and Katrina D. Maxwell. An assessment and comparison of common software cost estimation modeling techniques. In *Proceedings of the 21st international conference on Software engineering (ICSE '99)*, pages 313–322, New York, NY, USA, 1999. ACM.

[34] Lionel C. Briand, Tristen Langley, and Isabella Wieczorek. A replicated assessment and comparison of common software cost modeling techniques. In *Proceedings of the 22nd international conference on Software engineering (ICSE '00)*, pages 377–386, New York, NY, USA, 2000. ACM.

[35] Bart Broekman and Edwin Notenboom. *Testing Embedded Software*. Addison-Wesley, 2002.

[36] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. In *Data Mining and Knowledge Discovery*, pages 121–167, 1998.

[37] John M. Chambers. *Software for Data Analysis: Programming with R*. Springer, New York, 2008.

[38] Sunita Chulani. Constructive quality modeling for defect density prediction: Coqualmo. In *International Symposium on Software Reliability Engineering (ISSRE'99)*, November 1999.

[39] Code count tools. http://sunset.usc.edu/research/CODECOUNT/.

[40] Mike Cohn. *Agile Estimating and Planning*. Prentice Hall, 2005.

[41] Common Software Measurement International Consortium (COSMIC). *The COSMIC Functional Size Measurement Method, Version 3.0*, September 2007.

[42] Gennaro Costagliola, Sergio Di Martino, Filomena Ferrucci, Carmine Gravino, Genoveffa Tortora, and Giuliana Vitiello. Effort estimation modeling techniques: a case study for web applications. In *ICWE '06: Proceedings of the 6th international conference on Web engineering*, pages 9–16, New York, NY, USA, 2006. ACM.

[43] Gennaro Costagliola, Sergio Di Martino, Filomena Ferrucci, Carmine Gravino, Genoveffa Tortora, and Giuliana Vitiello. Effort estimation modeling techniques: a case study for web applications. In *6th international conference on Web engineering (ICWE '06)*, pages 9–16, New York, NY, USA, 2006. ACM.

[44] Hassan Diab, Marc Frappier, and Richard St-Denis. A formal definition of function points for automated measurement of b specifications. In *4th International Conference on Formal Engineering Methods (ICFEM 2002)*, pages 483–494, 2002.

[45] Harris Drucker, Chris J.C. Burges, Linda Kaufman, Alex Smola, and Vladimir Vapnik. Support vector regression machines. In *Advances in Neural Information Processing Systems*, pages 155–161. MIT Press, 1996.

[46] Ibrahim K. El-Far and James A. Whittaker. Model-based software testing. In J. Marciniak, editor, *Wiley Encyclopedia of Software Engineering*. John Wiley & Sons, 2001.

[47] Angi Faiks and Nancy Hyland. Gaining user insight: A case study illustrating the card sort technique. *College and Research Libraries journal*, 61(4):349–357, July 2000.

[48] N. Fenton. Software measurement: A necessary scientific basis. *IEEE Transactions on Software Engineering*, 20(3):199–206, 1994.

[49] David Garmus and David Herron. *Function Point Analysis, Measurement Practices for Successful Software Projects*. Addison Wesley, 2001.

[50] A. Gray and S. MacDonell. Applications of fuzzy logic to software metric models for development effort estimation. In *Fuzzy Information Processing Society*, pages 394–399, 1997.

[51] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 2nd edition, 1998.

[52] Klaus Hinkelmann and Oscar Kempthorne. *Design and Analysis of Experiments, Introduction to Experimental Design*. Wiley-Interscience, 2nd edition, 2007.

[53] Ronald Hocking. *Methods and applications of linear models: regression and the analysis of variance*. New York: Wiley-Intercience, 2003.

[54] John Hoenig and Dennis Heisey. The abuse of power: The pervasive fallacy of power calculations for data analysis. *The american statistician*, 55(1), February 2001.

[55] IBM Rational Software. Rational Manual Tester, September 2008. http://www.ibm.com/software/awdtools/tester/manual.

[56] Ali Idri, Alain Abran, and Taghi M. Khoshgoftaar. Estimating software project effort by analogy based on linguistic values. *Eighth IEEE International Symposium on Software Metrics (METRICS'02)*, 00:21, 2002.

[57] IFPUG: International Function Point Users Group. http://www.ifpug.org/.

[58] International Function Point Users Group (IFPUG). *Function Point, Counting Practices Manual, Release 1.0*, 1986.

[59] International Function Point Users Group (IFPUG). *Function Point, Counting Practices Manual, Release 2.0*, 1988.

[60] International Function Point Users Group (IFPUG). *Function Point, Counting Practices Manual, Release 3.0*, 1990.

[61] International Function Point Users Group (IFPUG). *Function Point, Counting Practices Manual, Release 4.0*, 1994.

[62] International Organization for Standardization (ISO). *ISO/IEC19761:2003, Software Engineering – COSMIC – A Functional Size Measurement Method*, 2003.

[63] International Organization of Standardization. *ISO/IEC FCD 9126-1, Information technology - Software product quality Part 1: Quality Model*, 1998.

[64] ISO. *ISO/IEC 20968: Software Engineering - MkII Function Point Analysis - Counting Practices Manual*, 2002.

[65] D. Ross Jeffery, Graham C. Low, and M. Barnes. A comparison of function point counting techniques. *IEEE Trans. Software Eng.*, 19(5):529–532, 1993.

[66] D.R. Jeffery and J. Stathis. Specification-based software sizing: an empirical investigation of function metrics. In *Proc. NASA Goddard Software Eng. Workshop*, 1993.

[67] Magne Jorgensen and Martin Shepperd. A systematic review of software development cost estimation studies. *IEEE Trans. Softw. Eng.*, 33(1):33–53, 2007.

[68] Paul Jorgensen. *Software Testing, A Craftsmans Approach*. CRC Press, second edition, 2002.

[69] Amela Karahasanoviæ, Bente Anda, Erik Arisholm, Siw Elisabeth Hove, Magne J, Dag I. Sj, and Ray Welland. Collecting feedback during software engineering experiments. *Empirical Software Engineering*, 10(2):113–147, 2005.

[70] A Karahasanovic, D I. K Sjøberg, and M Jørgensen. Data collection in software engineering experiments. In , editor, *Managing Information Technology in a Global Economy, Information Resources Management Association International Conference IRMA 2001, Software Engineering Track*, pages 1027–1028, Toronto, Ontario Canada, 2001. Idea Group Publishing.

[71] Chris F. Kemerer. Reliability of function points measurement: a field experiment. *Commun. ACM*, 36(2):85–97, 1993.

[72] Barbara Kitchenham. Counterpoint: The problem with function points. *IEEE Software*, 14(2):29,31, 1997.

[73] Barbara Kitchenham and Kari Känsälä. Inter-item correlations among function points. In *ICSE '93: Proceedings of the 15th international conference on Software Engineering*, pages 477–480, Los Alamitos, CA, USA, 1993. IEEE Computer Society Press.

[74] Barbara A. Kitchenham. The question of scale economies in software why cannot researchers agree? *Information and Software Technology*, 44(1):13–24, January 2002.

[75] Robert Kuehl. *Design of Experiments: Statistical Principles of Research Design and Analysis*. Duxbury Press, 2nd edition, 1999.

[76] Satish Kumar1, B. Ananda Krishna2, and Prem S. Satsangi3. Fuzzy systems and neural networks in software engineering project management. *Journal of Applied Intelligence*, 1994.

[77] Kevin Lano. *The B Language and Method: A Guide to Practical Formal Development*. Springer-Verlag, 1996.

[78] Daniel Leitao. Translating natural language descriptions into formal test case specifications. Master's thesis, Federal University of Pernambuco/UFPE, 2006.

[79] Ghislain Levesque, Valery Bevo, and De Tran Cao. Estimating software size with uml models. In *2008 C3S2E conference (C3S2E'08)*, pages 81–87, New York, NY, USA, 2008. ACM.

[80] Lucas Lima, Juliano Iyoda, and Augusto Sampaio. A permutation technique for test case prioritization in a black-box environment. In *2nd Brazilian Workshop on Systematic and Automated Software Testing*, Campinas-SP, Brazil, October 2008. To appear.

[81] Harold Linstone and Murray Turoff. *The Delphi Method: Techniques and Applications*. http://is.njit.edu/pubs/delphibook, 2002.

[82] David H. Longstreet. *Function Points Analysis Training Course*. Longstreet Consulting Inc, February 2005.

[83] Graham C. Low and D. R. Jeffery. Function points in the estimation and evaluation of the software process. *IEEE Trans. Softw. Eng.*, 16(1):64–71, 1990.

[84] S. MacDonell and A. Gray. Alternatives to regression models for estimating software projects. In *Proceedings of the IFPUG Fall 1996 Conference*. IFPUG, 1996.

[85] G S. Maddala. *Introduction to Econometrics*. Wiley, 3 edition, May 2001.

[86] Sergio Di Martino, Filomena Ferrucci, Carmine Gravino, and Emilia Mendes. Comparing size measures for predicting web application development effort: A case study. In *International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*, 2007.

[87] Katrina Maxwell. *Applied Statistics for Software Managers*. Prentice Hall, 2002.

[88] Jeff McAffer and Jean-Michel Lemieux. *Eclipse Rich Client Platform: Designing, Coding, and Packaging Java(TM) Applications*. Addison-Wesley Professional, 2005.

[89] Jerry Mendel. *Uncertain Rule-Based Fuzzy Logic Systems: Introduction and New Directions*. Prentice Hall, 2000.

[90] Emilia Mendes. A comparison of techniques for web effort estimation. In *ESEM '07: Proceedings of the First International Symposium on Empirical Software Engineering and Measurement*, pages 334–343, Washington, DC, USA, 2007. IEEE Computer Society.

[91] Emilia Mendes. *Cost Estimation Techniques for Web Projects*. IGI Publishing, September 2007.

[92] Emilia Mendes. The use of a bayesian network for web effort estimation. In *International Conference on Web Engineering (ICWE)*, pages 90–104, 2007.

[93] Emilia Mendes and Steve Counsell. Web development effort estimation using analogy. In *Proceedings of the 2000 Australian Software Engineering Conference*, 2000.

[94] Emilia Mendes and Barbara Kitchenham. Further comparison of cross-company and within-company effort estimation models for web applications. In *METRICS '04: Proceedings of the Software Metrics, 10th International Symposium on (METRICS'04)*, pages 348–357, Washington, DC, USA, 2004. IEEE Computer Society.

[95] Emilia Mendes and Nile Mosley. Further investigation into the use of cbr and stepwise regression to predict development effort for web hypermedia applications. In *International Symposium on Empirical Software Engineering (ISESE 2002)*, 2002.

[96] Emilia Mendes, Nile Mosley, and Steve Counsell. Comparison of web size measures for predicting web design and authoring effort. *IEE Proceedings - Software*, 149(3):86–92, 2002.

[97] Emilia Mendes, Nile Mosley, and Steve Counsell. Investigating web size metrics for early web cost estimation. *J. Syst. Softw.*, 77(2):157–172, 2005.

[98] Emilia Mendes, Ian Watson, Chris Triggs, Nile Mosley, and Steve Counsell. A comparative study of cost estimation models for web hypermedia applications. *Empirical Software Engineering*, 8(2):163–196, 2003.

[99] George Milliken and Dallas Johnson. *Analysis of Messy Data: Designed Experiments*, volume I. Chapman Hall/CRC, 1993.

[100] Tom Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[101] Parastoo Mohagheghi, Bente Anda, and Reidar Conradi. Effort estimation of use cases for incremental large-scale software development. In *Proceedings of the 27th international conference on Software engineering (ICSE05)*, pages 303–311. ACM Press, 2005.

[102] Kjetil Molokken and Magne Jorgensen. A review of surveys on software effort estimation. *International Symposium on Empirical Software Engineering (ISESE'03)*, 00:223, 2003.

[103] Tridas Mukhopadhyay and Sunder Kekre. Software effort models for early estimation of process control applications. *IEEE Trans. Softw. Eng.*, 18(10):915–924, 1992.

[104] Glenford J. Myers. *The Art of Software Testing*. John Wiley & Sons, Inc., 2004.

[105] Suresh Nageswaran. Test effort estimation using use case points. In *14th International Internet & Software Quality Week 2001*, June 2001.

[106] Richard E. Neapolitan. *Learning Bayesian Networks*. Prentice Hall, 2003.

[107] NESMA. *Definitions and Counting Guidelines for the Application of Function Points Analysis*, version 1.0 edition, 1990.

[108] Vu Nguyen, Sophia Deeds-Rubin, and Thomas Tan. Sloc counting standards. In *22nd International Forum on COCOMO and Systems/Software Cost Modeling*, 2007.

[109] Sidney Nogueira, Emanuela Cartaxo, Dante Torres, Eduardo Aranha, and Rafael Marques. Model based test generation: A case study. In *1st Brazilian Workshop on Systematic and Automated Software Testing (SAST 2007)*, October 2007.

[110] Strategic Evolution of ESE Data Systems (SEEDS). Survey of cost estimation tools, 2001.

[111] Adriano L. I. Oliveira. Letters: Estimation of software project effort with support vector regression. *Neurocomputing*, 69(13-15):1749–1753, 2006.

[112] C. Pandian. *Software Metrics: A Guide to Planning, Analysis, and Application*. CRC Press, Inc., 2003.

[113] Robert E. Park, Robert E. Park, Thomas R. Miller, and Lt Col. Software size measurement: A framework for counting source statements. Technical report, Software Engineering Institute, 1992.

[114] Shari Lawrence Pfleeger and Barbara A. Kitchenham. Principles of survey research: part 1: turning lemons into lemonade. *SIGSOFT Softw. Eng. Notes*, 26(6):16–18, 2001.

[115] http://www.crisp.se/planningpoker.

[116] Alexander Pretschner. Model-based testing. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 722–723, 2005.

[117] Lawrence Putnam and Ware Myers. *Measures for Excellence*. Yourdon Press Computing Series, 1992.

[118] Quantitative software measurement, QSM Inc. http://www.qsm.com.

[119] John Ross Quinlan. Learning with continuous classes. In *Fifth Australian Joint Conf. Artificial Intelligence*, pages 343–348, 1992.

[120] Matthias Riebisch, Detlef Streitferdt, and Ilian Pashov. Modeling variability for object-oriented product lines. In *Object-Oriented Technology (LNCS)*, volume 3013, pages 165–178. Springer Berlin, 2004.

[121] Christian Ritz and Jens Streibig. *Nonlinear Regression with R*. Springer, 2008.

[122] Melanie Ruhe, Ross Jeffery, and Isabella Wieczorek. Using web objects for estimating software development effort for web applications. In *METRICS '03: Proceedings of the 9th International Symposium on Software Metrics*, page 30, Washington, DC, USA, 2003. IEEE Computer Society.

[123] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition, 2002.

[124] R. M. Sakia. The box-cox transformation technique: A review. *Journal of the Royal Statistical Society. Series D (The Statistician)*, 41(2):169–178, 1992.

[125] Chris Schofield. Non-algorithmic effort estimation techniques. Technical report, Empirical Software Engineering Research Group (ESERG) at Bournemouth University, 1998.

[126] Rolf 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.

[127] Carolyn Seaman. Qualitative methods in empirical studies of software engineering. *IEEE Trans. Softw. Eng.*, 25(4):557–572, 1999.

[128] David Seaver. Fast function points. http://sunset.usc.edu/Activities/oct24-27-00/Presentations/Seaver_FAST Function Points.pdf.

[129] Martin Shepperd, Michelle Cartwright, and Gada Kadoda. On building prediction systems for software engineers. *Empirical Software Engineering*, 5:175–182, 2000.

[130] Sloc count. http://www.dwheeler.com/sloccount.

[131] George W. Snedecor and William G. Cochran. *Statistical Methods*. Iowa State University Press, 8 edition, 1989.

[132] Ingo Steinwart and Andreas Christmann. *Support Vector Machines*. Springer, 2008.

[133] Charles Symons. The cosmic functional size measurement method version 3,0. In *IWSM-Mensura 2007*, 2007.

[134] Hee Beng Kuan Tan, Yuan Zhao, and Hongyu Zhang. Conceptual data model-based software size estimation for information systems. *ACM Trans. Softw. Eng. Methodol.*, 19(2):1–37, 2009.

[135] Dante Torres, Daniel Leitao, and Flavia Barros. Motorola specnl: A hybrid system to generate nl descriptions from test case specifications. *Sixth International Conference on Hybrid Intelligent Systems (HIS'06)*, page 45, 2006.

[136] Dante Torres, Sidney Nogueira, Emanuela Cartaxo, Eduardo Aranha, Paulo Borba, Flávia Barros, Patrícia Machado, Augusto Sampaio, and Alexandre Mota. Brazil test center research group. In *1st Brazilian Workshop on Systematic and Automated Software Testing (SAST 2007)*, October 2007.

[137] Frank Vogelezang. Cosmic full function points the next generation of functional sizing. In *Software Measurement European Forum - SMEF 2005*, pages 281–289, 2005.

[138] Yong Wang and Ian H. Witten. Inducing model trees for continuous classes. In *Proceedings of the poster papers of the 9th European Conference on Machine Learning (ECML 97)*, 1997.

[139] Ian Watson. *Applying Case-Based Reasoning: Techniques for Enterprise Systems*. Morgan Kaufmann, 1997.

[140] Michael Whalen, Ajitha Rajan, Mats Heimdahl, and Steven Miller. Coverage metrics for requirements-based testing. In *ISSTA '06: Proceedings of the 2006 international symposium on Software testing and analysis*, pages 25–36. ACM Press, 2006.

[141] C. F. Jeff Wu and Michael Hamada. *Experiments: Planning, Analysis, and Parameter Design Optimization*. Wiley-Interscience, 2000.

[142] Ales Zivkovic, Ivan Rozman, and Marjan Hericko. Automated software size estimation based on function points using uml models. *Information and Software Technology*, 47(13):881–890, 2005.