Are Fit Tables Really Talking? A Series of Experiments to Understand whether Fit Tables are Useful during Evolution Tasks

Filippo Ricca[1], Massimiliano Di Penta[2], Marco Torchiano[3]
Paolo Tonella[4], Mariano Ceccato[4], Corrado Aaron Visaggio[2]
[1]Unità CINI at DISI, Genova, Italy
[2]RCOST – Dept. of Engineering – University of Sannio, Benevento, Italy
[3]Politecnico di Torino, Italy
[4]Fondazione Bruno Kessler–IRST, Trento, Italy
filippo.ricca@disi.unige.it, dipenta@unisannio.it, torchiano@polito.it,
tonella@fbk.eu, ceccato@fbk.eu, visaggio@unisannio.it

Test-driven software development tackles the problem of operationally defining the features to be implemented by means of test cases. This approach was recently ported to the early development phase, when requirements are gathered and clarified. Among the existing proposals, Fit (Framework for Integrated Testing) supports the precise specification of requirements by means of so called Fit tables, which express relevant usage scenarios in a tabular format, easily understood also by the customer. Fit tables can be turned into executable test cases through the creation of pieces of glue code, called fixtures.

In this paper, we test the claimed benefits of Fit through a series of three controlled experiments in which Fit tables and related fixtures are used to clarify a set of change requirements, in a software evolution scenario. Results indicate improved correctness achieved with no significant impact on time, however benefits of Fit vary in a substantial way depending on the developers' experience. Preliminary results on the usage of Fit in combination with pair programming revealed another relevant source of variation.

Categories and Subject Descriptors

D.2.1 [**Requirements/Specifications**]: Methodologies, Tools

General Terms

Experimentation, Measurement

Keywords

Empirical studies, Acceptance test, Software Maintenance

1. INTRODUCTION

When specifying requirements and change requests in natural language, analysts have to avoid several "sins" [9] that may bring about interpretation problems between analysts and developers. Some of them are *noise*, i.e., information not relevant to the problem or a repetition in the requirements, *silence*, when important information is missing, or *over-specification*, when portions of the solution are mentioned in the requirements. A substantial proportion of code defects, as high as 85%, originates at the requirement elicitation phase [18], both for initial requirements and for change requests, during software evolution. The root cause for such defects can be associated with ambiguous, incomplete, inconsistent, silent (unexpressed), unusable, over-specific or verbose requirements [9]. Test-driven development advocates a central role for testing and test cases, used to capture the features to be implemented in a form that can be checked automatically through execution. Unit test cases show the development progress for single modules. Similarly, executable acceptance test cases have been proposed to measure and describe precisely the level of progress in the implementation of the initial requirements or change requests. According to the agile methodologies [8], acceptance test cases are deemed more precise and accurate sources of information about the customer's requirements than their description in natural language. Acceptance test cases are "talking" representation of the requirements, which can be consulted whenever ambiguities or misinterpretations may arise.

Among the technologies for supporting automated acceptance testing, Fit (Framework for Integrated Test) [10] is one of the most popular and widely used. Fit helps analysts write acceptance tests by means of simple HTML tables (Fit tables), including input and expected output for each test scenario. Different kinds of tables are used for different testing conditions, e.g., testing the output for a given sequence of values vs. testing the result of a sequence of actions. Developers write glue code (called *fixtures*) to link the test cases expressed in the Fit tables with the system under development. Once fixtures are available, the *test runner* can execute them, comparing Fit table data (expected output) with actual values obtained from the execution.

In this paper, we measure the effects of the adoption of Fit in a series of three controlled experiments, varying by subjects involved and working conditions. We evaluated the usage of Fit with subjects having different levels of programming skills and we considered, in one replication of the experiment, subjects working in pairs (pair programming). This study has two objectives: on the one hand, we want to empirically evaluate the effects of Fit on the clarification of requirements, in terms of correctness of the resulting code. On the other hand, we want to also evaluate the impact of

fit_tests.DiscountStructure			
small bags	beverage	discount	total price()
2	Coffee	0	1.24
4	Tea	0	2.48
5	Coffee	1	2.1
5	Tea	0	3.1
7	Coffee	1	3.34
7	Tea	0	4.34

(a) Column Fit table. Each row represents a test case: Fit table column's names without parentheses represent input, parentheses indicate output.

fit.ActionFixture		
start	fit_tests.VerifySupply	
enter	type of product	Coffee
check	number of small bags remained	10
enter	number of boxes	5
press	buying boxes	
check	number of small bags remined	310
check	cash account	845

(b) Action Fit table. The table represents a test case and the commands simulate the user actions.

Figure 1: Examples of Fit tables.

Fit on the time necessary to complete the task (i.e., the time overhead involved, if any). Results indicate that benefits are obtained with no substantial increase of involved task execution time. However, results vary largely, depending on the level of expertise of the involved subjects. Expert subjects can take advantage of Fit much more than inexperienced ones. Moreover, for subjects working in pairs, the presence of Fit does not make any significant difference, in terms of time and of correctness of resulting code. This may indicate a complementary role of pair programming with respect to the availability of executable acceptance test cases: while a developer is coding, the other inspects the code to avoid the introduction of faults.

The paper is organized as follows: Section 2 provides some basic backrounds on Fit and can be skipped by the reader already familiar with it. Section 3 gives the details of the experimental design we used and Section 4 reports the experimental results. We discuss the results and threats to validity in Section 5. Related works are discussed in Section 6, and Section 7 concludes the paper.

2. FRAMEWORK FOR INTEGRATED TEST

Acceptance testing is a validation activity, performed by the customer, on the entire system, just before the system is delivered and aimed at judging if the software is acceptable.

Very often, acceptance testing is performed in a rather informal fashion, and it is no more than a software demonstration. However, it would be highly desirable to have acceptance test cases precisely defined, and to have the acceptance testing phase as more automated as possible. Clearly, it is impossible to expect that customers — in most cases lacking software development expertise — are able to develop test drivers for test case execution. Customers should be allowed for specifying acceptance test cases in an easy way, without having to deal with source code development, creation of drivers, scripts, etc.

To this aim, frameworks such as *Fit* [10] have been conceived. *Fit* (Framework for Integrated Test) is an open source framework used to express acceptance test cases, with the aim of improving the communication between analysts and developers. Fit lets customers and analysts write acceptance tests in the form of tables (*Fit tables*) using simple HTML or even spreadsheets.

A Fit table specifies the inputs and expected outputs for the test. Figure 1-(a) shows an example of Column Fit tables, a particular kind of table where each row represents a test case. The first three columns are input values (*small bags*, *beverage* and *discount*) and the last column represents the corresponding expected output value (*total price()*). Other than *Column* Fit tables, it is possible to specify *Action* Fit tables (see Figure 1-(b)), to test user interfaces or workflows. An *Action* Fit table represents a test case where the first column contains commands (*start, enter, press*, and *check*) used to simulate the actions that a user would perform on a screen while the other columns contain the parameters. For example, the *press* command simulates the button click and the parameter is the name of the button. Others types of Fit tables (see [10]) are: *Row* Fit tables, to validate collection of objects produced as the result of a query and *TimedAction* Fit tables to deal with temporal, non functional requirements.

Fit tables cannot however be directly executed against the system under test. To this aim developers have to specify drivers, called *Fixtures*, to link the test cases to the system under test. A component in the framework, the *Test Runner*, executes the test cases by relying on the *Fixtures*, and compares Fit table data with actual values obtained from the System. The test runner highlights the results with colors (green = correct, red = wrong).

3. EXPERIMENT PLANNING

This section describes the definition, design and settings of the proposed experimentation following the guidelines by Wohlin *et al.* [17] and Juristo and Moreno [5]. Table 1 summarizes the main elements of the experimentation. For replication purposes, the experimental package has been made available[1].

3.1 Experiment definition and context selection

The *goal* of the study is to analyze the use of Fit tables with the *purpose* of evaluating their usefulness during maintenance tasks. The *quality focus* regards (i) the capability of Fit tables of supporting maintenance tasks, and (ii) the additional time that the use of Fit table might require. The *perspective* is both of *Researchers*, investigating on the effectiveness of Fit tables during a maintenance task, and of *Project managers*, evaluating the possibility of adopting the Fit tables in her/his organization. The *context* consists of two *objects* – two Java systems – and of *subjects*, i.e., three different classrooms of students (bachelor students, master students, and PhD students). The objects of the study are two simple Java programs, *LaTazza* and *AveCalc*.

LaTazza is an application for a hot drinks vending machine. LaTazza supports sale and supply of small-bags of beverages (Coffee, Tea, Lemon-tea, etc.) from the Coffee-maker. The application supports two kinds of clients: visitors or employees. Employees can purchase beverage paying by cash or on credit, visitors only paying by cash. The secretary can: sell small-bags to clients, buy boxes of beverages, manage credit and debt of employees, check the inventory and check the cash account. The system consists of 18 Java classes for a total of 1121 Lines of Code (LOC). Its requirement

[1] *http://www.rcost.unisannio.it/mdipenta/Fit-Package.zip*

Table 1: Overview of the experiment.

Goal	Analyze the support given by Fit tables on maintenance/evolution tasks.
Quality focus	Correctness of maintained code Maintenance time
Context	Objects: two Java systems. Subjects: bachelor, master and PhD students.
Null hypotheses	No effect on code correctness. No effect on maintenance time.
Main factor	Availability (or not) of Fit tables
Other factors	Subjects' Experience, System, Lab, type of task (corrective vs. evolution)
Dependent variables	(i) Code correctness assessed by executing a JUnit functional test suite. (ii) Time required to perform the maintenance tasks.

document comprises 9 requirements complemented with a total of 16 Fit tables.

AveCalc is a simple "desktop application" that manages an electronic record book for master students. A student can add a new exam to the register, remove an existing exam and remove all exams. An exam has a name, a CFU (a positive number that represents the University credits) and an (optional) grade. An exam without grade is an exam not taken. The grade is between 0 and 30 (inclusive). If the grade is greater than or equal to 18, the exam result is positive (passed), otherwise it is negative (failed). It is possible to save the register and to load it. AveCalc computes some statistics: average of the exams passed, total number of CFU, number of exams passed, graduation score and whether the student has passed a number of exams sufficient to defend his/her thesis. The system consists of 8 Java classes for a total of 1827 LOC. Its requirement document comprises 10 requirements complemented with a total of 19 Fit tables.

For both systems, four change requirements (CR) were defined. Two of them (CR1 and CR2) were related to corrective maintenance (e.g., in AveCalc an exam vote must be checked to be positive), while the other two (CR3 and CR4) were related to the introduction of new (simple) features, e.g., change of beverage selling policy for the LaTazza application. As for other requirements, change requirements were complemented with Fit tables (a total of 14 for AveCalc change requirements and 17 for LaTazza change requirements). All Fit tables considered were of different types, i.e., Column, Action and Row Fit tables.

The study was executed twice at the University of Trento, with different subjects, and once at the University of Sannio. The subjects participating in the two replications in Trento are 14 Master students (2nd year M.Sc.) attending the Laboratory of Software Analysis and Testing (*Exp I*) and 8 PhD students (*Exp II*). At the University of Sannio, the 18 Bachelor students (3rd year) attending the course of Databases (*Exp III*) were involved. In this last case students worked in pairs: one was responsible of writing the code, while the other of reading (change) requirement, supervising the developer's work and inspecting the code under development.

In each experiment, all the students are from the same class with, roughly, the same background. Bachelor students had attended previously Programming and Software Engineering courses, which is of course true also of Master and PhD students. All subjects, including the undergraduate, had a good knowledge on Java programming (they previously developed non-trivial systems as projects for at least 3 exams), and an average knowledge about software engineering topics (e.g., design, testing, software evolution). At minimum, undergraduate students followed one software engineering course where they learned analysis, design and testing principles. Subjects have been trained on the understanding and usage of Fit tables and FitNesse[2], i.e., the tool that implements the Fit table approach used in the experiments.

Table 2: Characteristics of the systems under study.

LaTazza			
Reqs	**Fit tables**	**Files**	**LOC**
9	18	18	1121

AveCalc			
Reqs	**Fit tables**	**Files**	**LOC**
10	19	8	1827

3.2 Hypotheses formulation and variable selection

The research questions this experimentation aims at addressing are:

RQ1: Does the availability of Fit tables help programmers to improve the correctness of maintained code after a maintenance/evolution tasks?

RQ2: Does the availability of Fit tables affect the time in the execution of maintenance interventions?

Once research questions are formulated, it is possible to turn them into null hypotheses that can be tested in an experiment:

- H_{0c} The availability of Fit tables does not significantly improve the correctness of the maintained source code.
- H_{0t} The availability of Fit tables does not significantly affect the time required for the maintenance task.

We can notice that H_{0c} is *one-tailed*, since we are interested to investigate whether Fit improves the correctness, while H_{0t} is *two-tailed*, since we do not know whether the use of Fit requires additional time or, on the other hand, might reduce the time needed for the maintenance task.

The treatments for the main factor (availability of test cases) are:

- (+) application requirements and change requirements enhanced with fit tables and fixtures, thus enabling test case execution;
- (-) requirements and change requirements only expressed in a textual form.

Textual requirements and change requirements were detailed, so that subjects having Fit tables available did not receive additional information not available in the textual requirements as well. The dependent variables to be measured in the experiment are the code correctness and the time required to perform the maintenance task. The code correctness is assessed by executing a JUnit test suite and measuring the fraction of test cases passed. Such JUnit test suites are different from the acceptance test suites and, to avoid bias, they have been developed independently from the Fit tables — i.e., by different people — following the category partitioning black-box strategy [12]. A total of 25 test cases for AveCalc and 24 for LaTazza have been developed. The time needed to perform the

[2]http://www.fitnesse.org

Table 3: Post-experiment survey questionnaire.

ID	Question
Q1	I had enough time to perform the lab tasks (1–5).
Q2	The objectives of the lab were perfectly clear to me (1–5).
Q3	The description of the System was clear (1–5).
Q4	The change requirements were perfectly clear to me (1–5).
Q5	I experienced no difficulty in reading the source code (1–5).
Q6	I experienced no difficulty in correcting the defects and implementing the changes (1–5).
Q7	How many executions (i.e., run of the System) have you done on average before having implemented the change? (A. =1; B. >=2 and <4; C. >=5 and <7; D. >=7 and <10; E. >=10)
Q8	Did you find change fit tables useful in correcting defects? (a–e).
Q9	Did you find running change fit tables tests useful in correcting defects? (a–e).
Q10	Did you find running requirements fit tables tests (regression) useful ? (a–e).
Q11	Did you find running requirements fit tables useful in understanding the application? (a–e).
	1 = strongly agree, 2 = agree, 3= not certain, 4 = disagree, 5 = strongly disagree. a = very much, b = enough, c = undecided, d = little, e = definitely not.

Table 4: Experiment design
(+) = with Fit tables, (-) = without Fit tables.

	Group A	Group B	Group C	Group D
Lab 1	LaTazza+	LaTazza-	AveCalc-	AveCalc+
Lab 2	AveCalc-	AveCalc+	LaTazza+	LaTazza-

tasks was measured by means of time sheets; students marked start and stop time for each change requirements implemented. Time is expressed in minutes. The independent variables considered in this paper are: the Objects (AveCalc and LaTazza), the Labs (as it will be shown in the next section, each experiment was organized in two laboratories, *Lab 1* and *Lab 2*), the subjects' experience (bachelor students, master students and PhD students), and the type of maintenance task performed (i.e., corrective for tasks CR1 and CR2, evolution for tasks CR3 and CR4).

3.3 Experimental design and procedure

We adopted a balanced experiment design intended to fit two Lab sessions (2-hours each). Subjects were split into four groups, each one working in Lab 1 on a system with a treatment and working in Lab 2 on the other system with a different treatment (see Table 4).

Subjects used the Eclipse Java Development Toolkit as development environment, with the FitNesse plugin[3] used to browse requirements and Fit tables, and to execute test cases. Each subject received an Eclipse project containing the software source code, and a FitNesse Wiki with both requirements and change requests. The experimental package also comprised a short textual description of the application, instructions on how to set-up the environment, a time sheet where subjects annotated the starting and completion time for each change requirement, and a post experiment questionnaire.

Before the experiment, subjects were trained by means of introductory lectures, followed by laboratories where they had to understand and develop Fit tables. For each Lab the subjects had two hours available to complete the four maintenance tasks (CR1-CR4).

After having configured the environment and read the system description, for each change requirement, subjects had to:

1. record the starting time on the time sheet;
2. read the change requirement and look at the Fit tables (if available);
3. implement the change requirement;
4. if Fit tables were available, run test cases of the application requirements (for regression testing purposes) and of the change requirement, possibly returning to steps 2 and 3 in case test case failed; and
5. record the completion time on the time sheet.

[3]http://www.bandxi.com/fitnesse/

During the experiment, teaching assistants and professors were in the laboratory to prevent collaboration among subjects, and to check that subjects properly followed the experimental procedure (e.g., they implemented the changes in the given order, they correctly annotated the time spent). After the experiment, subjects had to fill a post-experiment survey questionnaire. It aimed at both gaining insights about the students' behavior during the experiment and finding justifications for the quantitative results. It includes questions (see Table 3) about the tasks and systems complexity, the adequacy of the time allowed to complete the tasks and the perceived usefulness of the provided Fit tables. More precisely, the questionnaire consists of 7 common questions plus 4 questions (Q8-Q11) answered only by subjects using Fit tables. Answers to Q1-Q6 and to Q8-Q11 are expressed in a Likert scale [11] from 1 (strongly agree) to 5 (strongly disagree). Answers to Q7 are based on a 5-levels ordinal scale: $\{A, B, C, D, E\}$.

4. EXPERIMENTAL RESULTS

This section reports the results from the three experiments, analyzing the effect on the dependent variables of the main factors treatment and of other factors. Finally, results from the analysis of survey questionnaires is reported. Results of statistical tests are intended to be significant for a significance level of 95%.

4.1 RQ1: Does the presence of Fit tables help programmers to improve the correctness of maintained code after maintenance/ evolution tasks?

Figure 2 and Table 5 show boxplots, descriptive statistics and results of the unpaired, one-tailed Mann-Whitney test. Results indicate a significant difference between Fit and Text on the overall

Table 5: Fraction of test cases passed: descriptive statistics per Method, and results of Mann-Whitney test (one-tailed).

Exp	Fit				Text				p	Effect
	Subjects	Median	Mean	σ	Subjects	Median	Mean	σ	value	size (d)
I	14	0.7	0.6	0.1	14	0.5	0.5	0.2	**0.04**	0.54
II	8	0.9	0.8	0.2	8	0.6	0.6	0.1	<**0.01**	0.57
III	6 pairs	0.7	0.6	0.1	7 pairs	0.6	0.6	0.1	0.1	1.22
All	28	0.7	0.7	0.2	29	0.6	0.6	0.2	<**0.01**	0.72

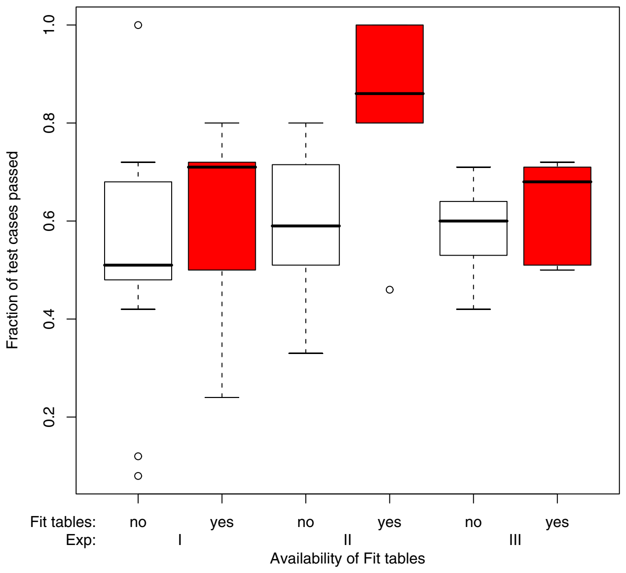


Figure 2: Boxplots for fraction of test cases passed.

data set (*All data*), for *Exp I* and for *Exp II*. While for master and PhD students (*Exp I* and *Exp II*), Fit tables had a significantly positive effect, this is not the case for undergraduate students (*Exp III*). Although subjects with Fit tables performed better than subjects without Fit tables, such a difference was not significant in *Exp III*.

The statistical significance alone does not tell anything about the practical impact of the treatment: it is important to measure the effect size of the main factor over the dependent variables, i.e., the magnitude of a main factor treatment effect on the dependent variables. We used the Coehn standardized difference between two groups [3], defined as the difference between the means (M_1 and M_2), divided by the pooled standard deviation (σ) of both groups $d = (M_1 - M_2)/\sigma$. The effect size is considered small for d=0.2, medium for d=0.5 and large for d=0.8. We observed a positive (i.e., Fit is better than only Text) and *medium* effect size for *Exp I* (0.54), and *Exp III* (0.57), and a *large* effect for the *Exp II* (1.22). The effect size for the overall data set is positive, and *medium* (0.72).

Since subjects performed the task, over the two different *Systems*, with the two possible treatments (i.e. Fit and Text), it is possible to use a paired, one-tailed Wilcoxon, test to compare the effects of the two treatments on each subject. As shown in Table 6, only for *Exp II* there is a significant difference between Fit and Text (p-value=0.01). The median difference is positive for *Exp I* and *Exp II*, while it is negative for *Exp III*. When using Fit, on average subjects improved of 15% for *Exp I*, 44% for *Exp II*, while a 6% decrease was observed for *Exp III*.

In summary, using unpaired statistical tests hypothesis H_{0c} could be rejected for *Exp I*, *Exp II*. Using paired statistical tests we can

Table 6: Fraction of test cases passed: within subject difference descriptive statistics and results of Wilcoxon test (one-tailed).

Exp	Subjects	Diff Med.	Diff Mean	Diff σ	p value
I	14	0.19	0.09	-0.09	0.17
II	8	0.26	0.27	0.03	**0.01**
III	5 pairs	-0.028	-0.08	0.02	0.78
All	27	0.13	0.12	-0.009	**0.01**

reject H_{0c} only for *Exp II*. Overall (i.e., considering all data) using unpaired and paired statistical tests we can reject H_{0c}. In particular, unpaired tests permit the hypothesis rejection for *Exp I* and *Exp II*, while paired tests only for *Exp II*.

4.2 RQ2: Does the presence of Fit tables affect the time in the execution of maintenance interventions?

To answer **RQ2**, we analyzed the time spent by subjects when Fit Tables were available or not. As described in Section 3.3, we asked subjects to annotate the time needed to setup the environment, and the time needed to perform each maintenance task. For sake of simplicity, we only report the total time of all tasks, including the setup time, since this result could be validated by the researchers and results obtained by excluding the setup time were consistent with these (R^2=0.95, p-value<0.01). Descriptive statistics and results of the Mann-Whitney test are shown in Table 7, while boxplots are shown in Figure 3. In all the three experiments no significant difference was found, however:

- In *Exp I* and *Exp II* median and mean times with Fit tables were slightly higher than values without Fit tables. The effect size was small for both experiments (d=0.16 for *Exp I* and 0.28 for *Exp II*). In these two experiments, as shown in Section 4.1, subjects with Fit tables were able to deliver source code for which a significantly higher fraction of test cases passed. In other words, Fits required subjects to spend some more time, although this was paid back by a higher code correctness.

- In *Exp III* subjects with Fit spent less time than subjects without Fit. The effect size was high (d=-1.4) however not statistically significant due to the small number of subjects. Of course, absolute times cannot be compared with the other experiments since in this case subjects worked in pair (although times without Fit are comparable with other experiments), however such a difference is worth of a deeper discussion. We believe that, when Fit was not available, subjects, who were less expert than in *Exp I* and *Exp II*, spent more time inspecting the code and testing it to increase its correctness, while, when Fit was available, they spent less time and relied on the Fit table.

Table 7: Total time spent (in min.): Descriptive statistics per Method, and results of Mann-Whitney test (two-tailed).

Exp	Fit			Text			p	Effect
	Med.	Mean	σ	Med.	Mean	σ	value	size (d)
I	82.0	77.7	18.5	80.0	74.9	14.5	0.5	0.16
II	79.5	82.2	23.2	75.0	75.0	28.3	0.5	0.28
III	51.5	49.5	18.2	94.0	92.0	25.0	0.1	1.4

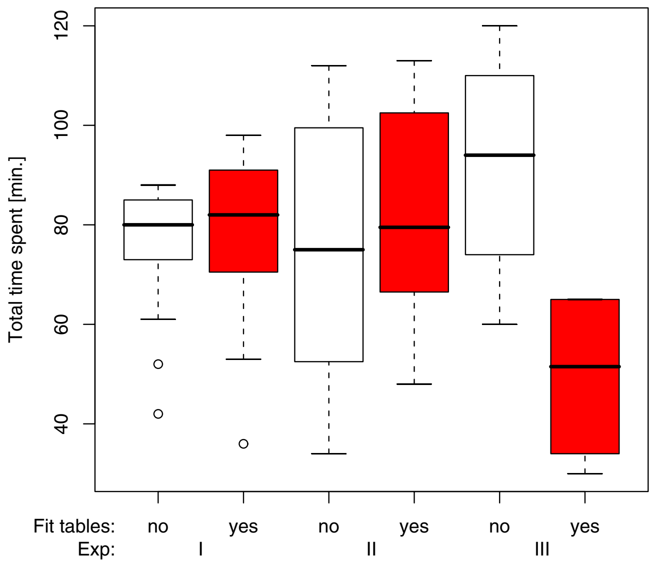


Figure 3: Boxplots of total time spent.

Table 8: Two-Way ANOVA of Fit & Exp.

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Fit	1	0.27	0.27	9.27	**0.003**
Exp	2	0.20	0.10	3.45	**0.039**
Fit:Exp	2	0.08	0.04	1.33	0.27
Residuals	51	1.52	0.03		

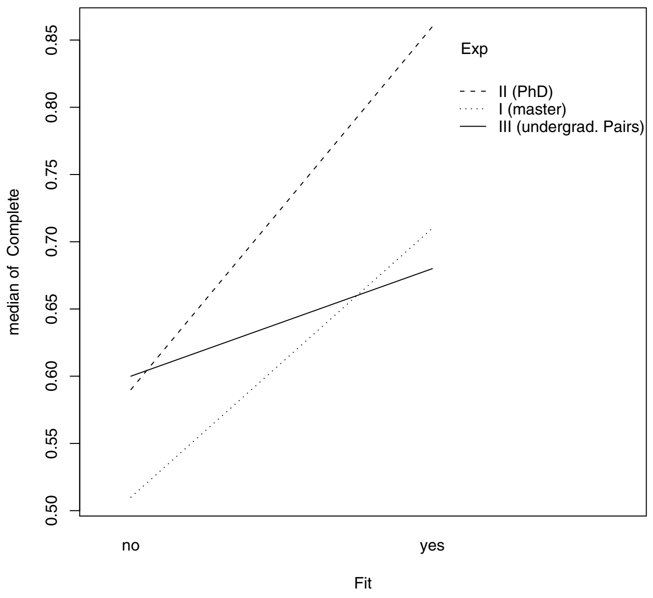


Figure 4: Interaction plot of main factor with Subjects Experience.

A paired analysis (Wilcoxon test) was performed for subjects participating in both labs of each experiment. Also in this case, results indicate no significant difference (p-value=0.66 for *Exp I*, 0.55 for *Exp II* and 0.25 for *Exp III*. Hypothesis H_{0t} cannot be therefore rejected.

4.3 Effect of other factors

This section analyzes the effect of other factors on the dependent variables (fraction of test cases passed and Time), namely of *Lab* (i.e., whether a result was obtained in the first or second session, to evaluate the learning effect), *System* and of the different subjects' *Experience*. The analysis was performed by using a two-way Analysis of Variance (ANOVA).

On the overall data set, we found no significant effect of the *System* on the fraction of test cases passed (p-value=0.97) nor any interaction with the main factor (p-value=0.88). Similarly, non significant results were obtained analyzing data for each experiment separately. Also the *Lab* factor did not produce any significant effect on the fraction of test cases passed (p-value=0.13) nor any interaction with the main factor (p-value=0.58). Results were confirmed analyzing each experiment separately. The third factor we considered was the type of subjects for the different experiments (*Exp*). In this case, as shown in Table 8, the effect is significant (p-value=0.039). In other words, subjects with different experience gained different benefits from the use of Fit tables. Figure 4 shows the interaction plot of the *Experience* factor with the main factor. Such an interaction is not overall statistically significant (p-value=0.27), although we can find a significant difference between *Exp I* and *Exp II* that is also graphically visible (Tukey multiple comparisons of means p-value=0.03). By observing the interaction plot we observed that the potential benefits gained with the presence of Fit tables are represented by the slope of the segments: the slope — and thus the benefit gained with Fit — is higher for high experience subjects (master students and above all PhD students).

Then, we analyzed whether the type of maintenance task performed in the four different tasks assigned to subjects (corrective maintenance or evolution) had an effect on the code correctness or an interaction with the main factor. To test the presence of such an effect, we performed a two-way ANOVA by *Method & Maintenance Type*. In all the three experiments, a significant influence of the type of maintenance task was found (p-value=$1.616 \cdot 10^{-14}$ for *Exp I*, 0.00023 for *Exp II*, and $7.744 \cdot 10^{-12}$ for *Exp III*). Instead, no interaction was found with the main factor (p-value=0.73 for *Exp I*, 0.36 for *Exp II*, and 0.92 for *Exp III*). In other words, although subjects were able to better perform (in terms of resulting code correctness) corrective maintenance tasks than evolution tasks this did not depend on the main factor treatment (availablility of Fit tables).

To analyze the effect of Time, a two-way ANOVA of Time by *Method & System* and by *Method & Lab* was performed for the three experiments. No significant effect of the *System* factor was found (p-value=0.21 for *Exp I*, 0.064 for *Exp II* and 0.48 for *Exp III*) on the Time, nor any significant interaction of these factors with the *Method* (p-value=0.66 for *Exp I*, 1.00 for *Exp II* and 0.068 for *Exp III*). Similarly, no significant effect of the *Lab* factor was found (p-value=0.35 for *Exp I*, 0.76 for *Exp II* and 0.49 for *Exp III*) nor any significant interaction with the *Method* (p-value=0.4 for *Exp I*, 0.65 for *Exp II* and 0.068 for *Exp III*). An analysis of the overall

Table 9: Spearman correlation between Time and fraction of test cases passed.

Experiment	Correlation	p-value
Exp I	-0.064	0.76
Exp II	0.17	0.53
Exp III	0.018	0.97

data set was not possible in this case since times for pairs (*Exp III*) cannot be compared with times of singles (other experiments). Results indicate that (i) the two different systems did not require a significantly different time to be maintained; and, (ii) there was no learning between the two laboratories.

The effect of maintenance task type (corrective maintenance or evolution) on the time needed to perform the task was significant for *Exp I* (p-value=0.01) and *Exp II* (p-value=$1.096 \cdot 10^{-6}$), while marginally significant for *Exp III* (p-value=0.06). No interaction was found with the main factor (p-value=0.77 for *Exp I*, 0.65 for *Exp II*, and 0.77 for *Exp III*). While subjects spent more time for evolution tasks, there was no significant difference where Fit tables were available or not.

Finally, we analyzed whether there was any correlation between the two dependent variables, i.e., the time spent and the fraction of test cases passed. This is because the code correctness could depend on the time subjects spent to perform the task, rather than on the main factor treatments. As shown in Table 9, the Spearman (non-parametric) correlation is low and not significant.

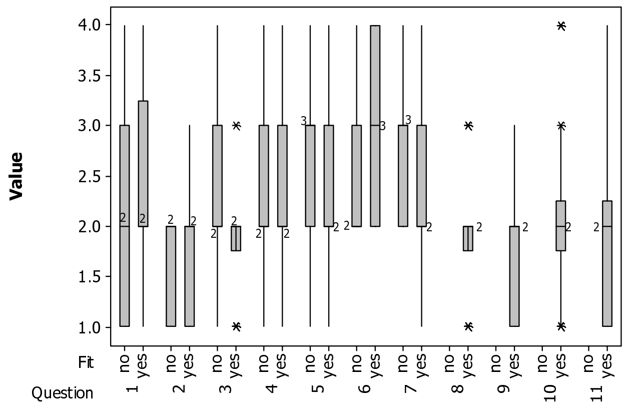
4.4 Survey Questionnaire Results

This section analyzes results from the survey questionnaire. We considered the answers from the questionnaire in agreement with the question sentence when the median was less than 3 (using the one-tailed Mann-Whitney test). Comparison among the three experiments was done using the Kruskal-Wallis test, while differences between results of different treatments was tested using a two-tailed Mann-Whitney test. Boxplots of survey questionnaire results for the three experiments are presented in Figure 5.

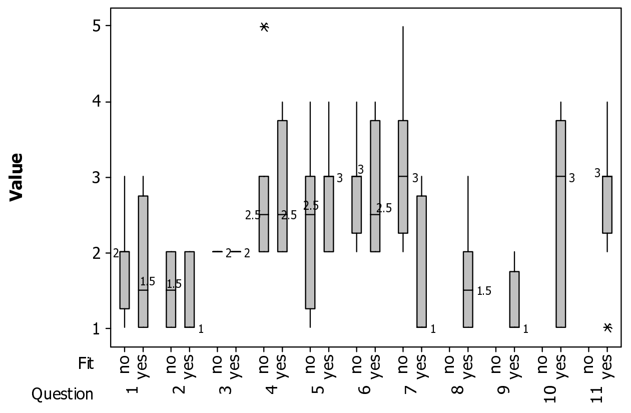
Besides the treatment received, for all the three experiments subjects felt to have had enough time to complete the task (**Q1**), and felt the laboratory objectives (**Q2**) and the system descriptions (**Q3**) clear. The change requirements (**Q4**) were clear for *Exp I* (graduate students) (median=2) and *Exp III* (pairs of undergraduate students) (median=1) but not for *Exp II* (PhD students) (median=2.5). The difference among the three experiments was significant (p-value<0.01).

In all three experiments subjects found the code easy to read (**Q5**) with no particular difference depending on the presence of Fit tables. Also, in all cases subjects partially agreed not to have had difficulties in implementing the changes (**Q6**), although for *Exp II* subjects with Fit seemed to perceive more difficulty (median=3) than subjects without Fit (median=2). In this case, difficulties were higher for subjects of *Exp III* (median=3), although the differences from agreements of the other two experiments were not significant (p-value=0.6 for *Exp I* and 0.5 for *Exp II*).

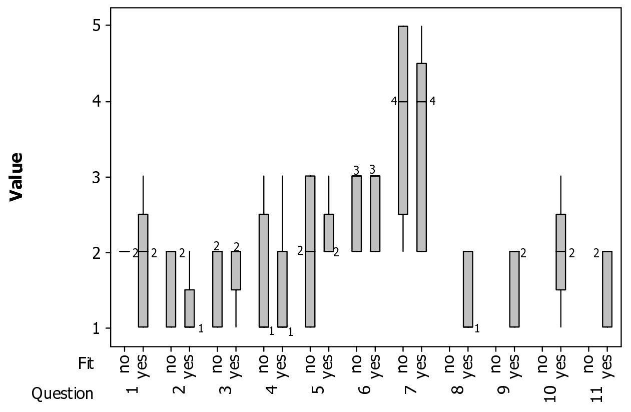
Q7 clearly indicated how subjects of *Exp I* and *Exp II* benefited more of the use of Fit than subjects of *Exp III* (p=0.04). In fact, despite the availability of Fit tables, subjects of *Exp III* required a median number of executions between 7 and 10 to implement the change. Such a number was between 5 and 7 for subjects of *Exp II* when Fit was not available, and decreased to 1 when Fit was available. For *Exp I* the median number was between 5 and 7 without Fit tables and between 2 and 4 with Fit tables. In other



(a) Exp I



(b) Exp II



(c) Exp III

Figure 5: Survey questionnaire results.

words, for *Exp I* and *Exp II*, Fit test case execution helped to reduce the number of manual test necessary before releasing the changed software.

Questions **Q8-Q11** were only asked to subjects having Fit tables available. Subjects for all three experiments felt Fit tables useful for defect correction (**Q8** and **Q9**). Subjects of *Exp I* and *Exp III* indicated Fit tables as useful for regression testing (**Q10**) and for comprehension (**Q11**) (median=2 in all cases), while subjects of *Exp II* were undecided (median=3 for both questions).

5. DISCUSSION

By looking at the experimental results, two notable facts emerge. First, experienced developers (graduate students and PhD students in particular) receive much more benefits from Fit tables than novice developers (undergraduate students). When Fit tables are available, PhD students are able to outperform all other subjects, while master students are able to perform as well as pairs of undergraduate students. This can be due to the expertise and level of maturity that may be needed to understand the change requirements "by example" and to assess the correctness of the maintained code through Fit table execution. Graduate students have previously taken classes on software engineering and have followed advanced courses on testing. PhD students even had the opportunity of applying testing in practice when working on research projects; this allows them to exploit the availability of test cases (Fit tables) during the maintenance task. Undergraduate students only know basic testing principles and techniques, and had less opportunity during their career to perform testing tasks. They felt more comfortable with inspecting the code and performing *ad hoc* testing, as visible from the larger number of executions needed before releasing the changed code (question **Q7** of the survey questionnaire). Overall, there is no significant increase of time needed to perform the tasks when Fit is available, and also the measured effect size is small. This indicates that, provided a previous training, the overhead introduced by understanding and executing Fit tables is limited and also counterbalanced by the need for *ad hoc* testing, and inspections when Fit tables are not available.

The second result that emerges from the experiment is that, when working in pairs, the benefit introduced by Fit tables is reduced. Working in pairs make requirement understanding easier, hence reducing the requirement clarification role played by Fit tables. Benefits of Fit tables are inferior if developers can comprehend requirements in pairs. Feedback provided by subjects (questions **Q4**, **Q10** and **Q11**) were in agreements with quantitative results obtained in the experiment. The pair cooperation helps to better understand requirements and, above all, to improve the code correctness when performing the maintenance task. This because while one subject codes, the other inspects the code under development and provides feedbacks whenever necessary. Such an inspection acts as an alternative to Fit table execution, or at least reduces its capability to reveal failures since many programming errors are already corrected during the development. It looks like the observer in the pair is mentally simulating the execution of the code being written under relevant scenarios, hence reducing the role of Fit table execution. However, further experiments are needed to confirm this finding, since pairs (*Exp III*) were less experienced than subjects for other experiments. Because of that, such a result can just be due to the different experience level.

5.1 Threats to validity

This section discusses the threats to validity [17] that can affect our results: *construct*, *internal*, *conclusion*, and *external* validity threats.

Construct validity threats concern the relationship between theory and observation. In particular, this threat is related to how code correctness after maintenance interventions and time were measured. The code correctness was assessed in a objective way by executing a JUnit functional test suite and measuring the percentage of test cases passed and failed. To avoid bias, the JUnit test suites were developed independently from the acceptance test suites. The developed test suites are representative of functional test suites developed according to the category partition technique. By using a code coverage tool, we also verified that the developed test suites were able to ensure a high statement coverage (between 80% and 100%) for the methods impacted by the changes. Of course, other test suites could possibly produce different results. Time was measured by means of proper time sheets and validated qualitatively by researchers who were present during the experiment. Although this may not be very accurate, this is a widely adopted way of measuring performance (monitoring is often not possible for legal reasons), and teaching assistants checked that the forms were correctly filled.

Internal validity threats concerns external factors that may affect an independent variable. A proper analysis — i.e., two-way ANOVA —was performed to analyze the effect of these factors. Although it was impossible to properly discern the effect of pairs from that of experience: to achieve it we would need a specific experiment, that will be part of our future work. Other *internal validity* threats can be due to the learning effect experienced by subjects between *Labs*. Such an effect is mitigated by the chosen experimental design: subjects worked, over the two *Labs*, on different systems with different levels of the main factor (Fit vs. Text). Also, the *Lab* effect was measured with a two-way ANOVA and was found not to be significant. Also (see Table 4), the chosen design considers all combinations of systems and main factor treatments (i.e., Fit tables available or not). This mitigates the impact due to the fact that an individual/pair used one treatment before the other. To avoid social threats due to evaluation apprehension, students were not evaluated on their performance. Although subjects participating to each experiments had a similar background, ability was not explicitly assessed before the experiment. Finally, subjects were not aware of the experimental hypotheses.

Conclusion validity concerns the relationship between the treatment and the outcome. Proper tests were performed to statistically reject the null hypotheses. The small sample size (overall 40 subjects, however the analyses were applied to 32 points, since for *Exp III* subjects worked in pairs) could have limited the capability of statistical tests to reveal any effect. However, attention was paid to not violate assumptions made by statistical tests and, whenever conditions necessary to use parametric statistics did not hold, we used non-parametric tests, in particular the Mann-Whitney test for unpaired analyses, the Wilcoxon test for paired analyses and the Tukey test for multiple comparisons. The measure chosen to evaluate the code correctness, i.e., the fraction of JUnit test cases passed, allowed to evaluate the maintenance tasks executed by students in an objective manner, avoiding to give subjective scores. Survey questionnaires, mainly intended to get qualitative insights, were designed using standard ways and scales [11].

External validity concerns the generalization of the findings. Threats belonging to this category can be related to (i) the simple Java systems chosen and (ii) to the use of students as experimental subjects. The selected subjects represent a population of students specifically trained on software engineering tasks. Also, all subjects involved in *Exp I* (graduate students) and *Exp III* (PhD students) either had some professional experiences or worked on industrial projects during their thesis. This makes these students comparable to industry junior developers. Nevertheless, the working pressure and the overall environment in industry is different, thus replicating the study in industry is highly desirable. Further studies with larger systems and more experienced developers are needed to confirm or contrast the obtained results. Finally, our study focused on the use of *Fit*, although other acceptance testing frameworks (e.g., [6, 16, 2]) can also be used.

6. RELATED WORK

Although there are papers [1, 13] and books [10] describing acceptance testing with Fit tables, only a few works report empirical studies about Fit.

Read *et al.* [13] found that experience can affect the understandability and the capability of defining the functional requirements for writing acceptance test cases. In particular, authoring acceptance tests is more difficult than reading and understanding. Since writing acceptance tests requires practice and experience, the support of a tool is not enough. The need for experience is reflected in our results, where graduate students benefited more of Fit than undergraduate.

Melnik *et al.* [8] investigated on the use of Fit user acceptance tests for specifying functional requirements. They conducted experiments at the University of Calgary and at the Southern Alberta Institute of Technology. Results showed that the use of Fit tables and the possibility to execute them improve the comprehension of requirements. In this experiment, students worked on their own (off-line) for two weeks.

Also, Melnik *et al.* [7] investigated whether acceptance tests can be authored effectively by customers of agile projects. The authors carried out an experiment, involving 40 students as subjects, who were enrolled in business school for graduate students and computer science graduate and undergraduate students. The experiment did not support the hypothesis that the quality of executable acceptance tests produced by the customer team will be strongly and positively correlated with the quality of the implementation produced by the development team. However, results show that customers can specify functional requirements clearly. The main differences between [7, 8] and our work are the way subjects worked (off-line vs. on-line), the kind of task (development vs. maintenance), and the measurement instrument (questionnaires vs. code correctness measured through JUnit functional test suites).

In Deng *et al.*'s study [4] a survey on Acceptance Test Driven Development has been conducted with 33 professionals by sending a questionnaire. The study concludes that the time frame between the definition of an acceptance test and its first successful pass is much longer than that of unit testing. The average time frame for acceptance testing is more than 4 hours, i.e., more than half a day. Starting from this result the authors built a new tool, named FitClipse, able to distinguish the two possible "failing cases". Color red indicated a regression failure (i.e., failure as a result of a recent change losing previously working functionality) while orange suggest that there is an unimplemented feature (not a failure). An initial self-evaluation shows that the distinction between the failure states and the use of FitClipse is useful.

Ricca *et al.* [14] report a controlled experiment with master students aimed at assessing the impact of Fit tables on the clarity of requirements. The main difference with the present study is that in [14] students could only use the Fit tables to better grasp the requirements while in our case, students had the Fit tables and the Fixtures with the possibility to execute them. The results obtained indicate that Fit helps in the understanding of the requirements but involves additional effort (even if not in significant way).

In a companion paper [15] some of the authors of the present paper reported some partial analyses and some results from the experiment conducted in Trento with master students (*Exp I*). The second and third experiment (*Exp II*, *Exp III*) are presented here for the first time.

7. CONCLUSIONS AND FUTURE WORK

This paper reported a series of experiments aimed at assessing the benefits obtained in performing maintenance tasks with the support of Fit tables. Results indicated that (i) the maintained code is more correct when Fit is available; and (ii) the overhead induced by the presence of Fit — measured in terms of time — is negligible.

In addition, interesting effects of two co-factors, experience of subjects and work organization, i.e., working in pairs or as singles — were found that:

1. the benefits of Fit were higher for subjects with a higher experience, such as PhD students. They were able to improve the correctness of the maintained code of 44% on average; and
2. working in pairs seemed to compensate the benefits of Fit in terms of correctness although it may reduce the time required to perform the task.

Work-in-progress is devoted to better analyze, with specific experiments, the effect of work organization (e.g., working in pairs) on the usefulness of Fit tables. In addition, it would be worth of investigating whether the progressive increase in benefits we observed in our experiments will keep for professional developers. Last, but not least, future empirical studies will aim at analyzing whether the additional effort and cost — encountered during the requirement elicitation phase — due to the development of Fit tables, will be paid back by a higher code correctness. In other words, if on the one hand the present experimentation has shown the benefits introduced by Fit tables, the adoption of such a technique in the software development life-cycle must be considered taking into account the costs it will introduce.

8. ACKNOWLEDGMENTS

Massimiliano Di Penta and Marco Torchiano were partially supported by the project METAMORPHOS (MEthods and Tools for migrAting software systeMs towards web and service Oriented aRchitectures: exPerimental evaluation, usability, and tecHnOlogy tranSfer), funded by MiUR (Ministero dell'Università e della Ricerca) under grant PRIN2006-2006098097.

9. REFERENCES

[1] J. Aarniala. Acceptance testing. In *whitepaper. www.cs.helsinki.fi/u/jaarnial/jaarnial-testing.pdf*, October 30 2006.

[2] G. Bache, J. Andersson, and P. Sutton. Xp with acceptance test driven development - a rewrite project for a resource optimization system. In *Proceedings of the 8th International Conference on Agile Processes in Software Engineering and eXtreme Programming (XP 2003)*, pages 180–188. Springer, 2003.

[3] J. Cohen. *Statistical power analysis for the behavioral sciences (2nd ed.)*. Lawrence Earlbaum Associates, Hillsdale, NJ, 1988.

[4] C. Deng, P. Wilson, and F. Maurer. Fitclipse: A fit-based eclipse plug-in for executable acceptance test driven development. In *Proceedings of the 8th International Conference on Agile Processes in Software Engineering and eXtreme Programming (XP 2007)*. Springer, 2007.

[5] N. Juristo and A. Moreno. *Basics of Software Engineering Experimentation*. Kluwer Academic Publishers, Englewood Cliffs, NJ, 2001.

[6] K. Leung and W. Yeung. Generating user acceptance test plans from test cases. In *Proceedings of the 31st Annual International Computer Software and Applications Conference - Vol. 2- (COMPSAC 2007)*, pages 737–742, Los Alamitos, CA, USA, 2007. IEEE Computer Society.

[7] G. Melnik, F. Maurer, and M. Chiasson. Executable acceptance tests for communicating business requirements: customer requirements. In *Proceedings of AGILE 2006 Conference (AGILE2006)*, pages 35–46, Los Alamitos, CA, USA, 2006. IEEE Computer Society.

[8] G. Melnik, K. Read, and F. Maurer. Suitability of fit user acceptance tests for specifying functional requirements: Developer perspective. In *Extreme programming and agile methods - XP/Agile Universe 2004*, pages 60–72, August 2004.

[9] B. Meyer. On formalism in specification. *IEEE Software*, January 1985.

[10] R. Mugridge and W. Cunningham. *Fit for Developing Software: Framework for Integrated Tests*. Prentice Hall, 2005.

[11] A. N. Oppenheim. *Questionnaire Design, Interviewing and Attitude Measurement*. Pinter, London, 1992.

[12] T. Ostrand and M. Balcer. The category-partition method for specifying and generating functional tests. *Communications of the Association for Computing Machinery*, 31(6), June 1988.

[13] K. Read, G. Melnik, and F. Maurer. Examining usage patters of the fit acceptance testing framework. In *Proc. 6th International Conference on eXtreme Programming and Agile Processes in Software Engineering (XP2005)*, pages Lecture Notes in Computer Science, Vol. 3556, Springer Verlag: 127–136 2005, June 18-23 2005.

[14] F. Ricca, M. Torchiano, M. Ceccato, and P. Tonella. Talking tests: an empirical assessment of the role of fit acceptance tests in clarifying requirements. In *International Workshop on Principles of Software Evolution (IWPSE 2007)*, pages 51–58. ACM Press, September 2007.

[15] F. Ricca, M. Torchiano, M. Di Penta, M. Ceccato, and P. Tonella. The use of executable fit tables to support maintenance and evolution tasks. *Electronic Communications of the EASST*, 8, 2008.

[16] J. Sauvè, O. Abath Neto, and W. Cirne. EasyAccept: a tool to easily create, run, and drive development with automated acceptance tests. In *Proceedings of 2006 international workshop on Automation of software test*, pages 111 – 117, New York, NY, USA, 2006. ACM Press.

[17] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering - An Introduction*. Kluwer Academic Publishers, 2000.

[18] R. Young. *Effective Requirements Practice*. Addison-Wesley, Boston, MA, 2001.