Electronic Communications of the EASST
Volume 8 (2008)

Proceedings of the Third International ERCIM Symposium on Software Evolution (Software Evolution 2007)

The Use of Executable FIT Tables to support Maintenance and Evolution Tasks

Filippo Ricca, Marco Torchiano, Massimiliano Di Penta, Mariano Ceccato and Paolo Tonella

12 pages

Guest Editors: Tom Mens, Ellen Van Paesschen, Kim Mens, Maja D'Hondt
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

The Use of Executable FIT Tables to support Maintenance and Evolution Tasks

Filippo Ricca[1], Marco Torchiano[2], Massimiliano Di Penta[3], Mariano Ceccato[4] and Paolo Tonella[5]

[1] filippo.ricca@disi.unige.it
Unit CINI at DISI, Genova, Italy

[2] torchiano@polito.it
Politecnico di Torino, Italy

[3] dipenta@unisannio.it
University of Sannio, Benevento, Italy

[4] ceccato@fbk.eu
[5] tonella@fbk.eu
Fondazione Bruno Kessler—IRST, Trento, Italy

Abstract: Acceptance testing is a kind of testing performed prior to software delivery. In the agile approach, acceptance test suites are specified by analysts and customers during the requirement elicitation phase and used to support the development/maintenance activities. This paper reports an experiment with master students that investigates on the usefulness of executable acceptance test cases, developed by using FIT (Framework for Integrated Test), during software maintenance and evolution activities. The preliminary results indicate that FIT tables help students to correctly perform the maintenance/evolution tasks with no significant impact on time.

Keywords: Experiment with students, acceptance testing, FIT tables.

1 Introduction

FIT (Framework for Integrated Test) [MC05] is an open source framework used to express executable acceptance test cases in a simple way. FIT lets analysts write acceptance tests (FIT tables) using simple HTML tables. Programmers write code (Fixtures) to link the test cases with the System to verify. Then, in a test-driven development scenario, they perform their development or maintenance task being supported by the execution of these test cases.

In this paper we describe a controlled experiment aimed at assessing whether FIT tables are helpful in maintenance tasks. We asked some master students to execute four maintenance/evolution tasks (two corrective maintenance tasks and two evolution interventions), providing them two Java Systems to be maintained with and without the FIT Tables. The development environment is Eclipse with the plug-in Fitnesse[1] that implements the FIT table-based

[1] http://fitnesse.org/

approach.

The research questions that we are interested in answering are:

RQ1: Does the presence of FIT tables help programmers to execute maintenance tasks?

RQ2: Does the presence of FIT tables improve the *productivity* in the execution of maintenance interventions?

The dependent variable "correctness" was measured by exercising an alternative JUnit[2] acceptance test suite, the variable "productivity" using time sheets where students annotated start and stop time expressed in minutes.

Preliminary results of our experiment show that FIT tables help developers to correctly perform the four maintenance/evolution tasks given without affecting the productivity. The difference between the two groups considering the time to complete the tasks was similar and not significant.

The paper is organized as follows: Section 2, briefly, presents the Framework for Integrated Test (FIT) used in the experimental study. Section 3 describes the definition, design and settings of the proposed experiment. Results are presented in Section 4 while related works, conclusions and future works are given, respectively, in Section 5 and Section 6.

2 FIT tables, Fixtures and Test Runner

The FIT tables serve as the input and expected output for the tests. Figure 1 shows an example of *Column* FIT tables, a particular kind of table where each row represents a test case (inputs and output). The first five columns are input values (*Name*, *Surname*, *Address*, *Date of birth* and *Credit/Debit*) and the last column represents the corresponding expected output value (*Member number()*).

Other than Column FIT tables, it is possible to specify *Action* FIT tables, to test user interfaces or work-flows. 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. Others types of FIT tables (see [MC05]) are: *Row* FIT tables, to validate collection of objects produced as the result of a query, and *TimedAction* FIT tables to deal with non functional requirements. With this large set of different types of FIT tables we are fairly confident that each functional requirement of a traditional business systems (this could not to be true in other contexts, for example reactive systems) may be "transformed" in executable FIT tables.

Developers write the Fixtures to link the test cases with the System to verify. A component in the framework, the Test Runner, compares FIT table data with actual values obtained from the System. The Test Runner highlights the results with colors (green = correct, red = wrong). See the relationships among FIT tables, Fixtures, Test Runner and System under test in Figure 2.

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

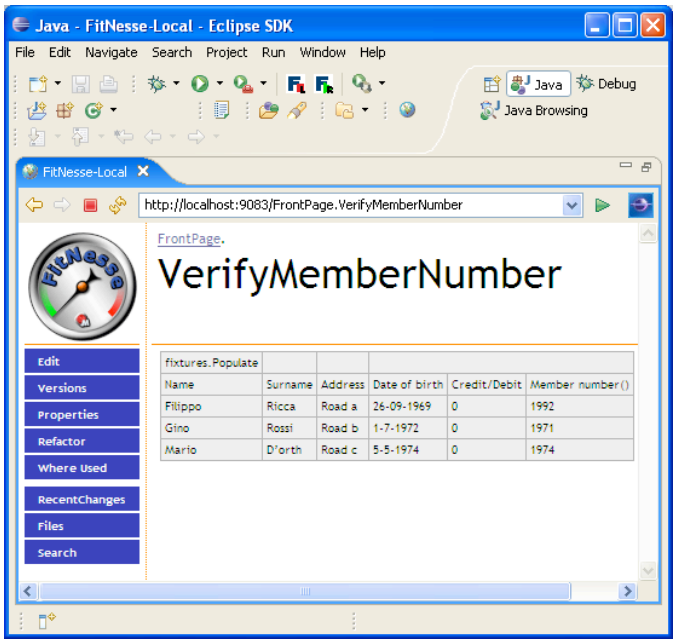

Figure 1: Example of Column FIT table. FIT table column's names without parenthesis represent input; parenthesis indicate output.

3 Experiment definition, design and settings

We conceived and designed the experiment following the guidelines by Wohlin *et al.* [WRH+00]. The *goal* of the study is twofold: to analyze the use of FIT tables with the *purpose* of evaluating their usefulness during maintenance tasks and to measure the effort (if any). The *perspective* is both of *Researchers*, evaluating how effective are the FIT tables during the maintenance activities, and of *Project managers*, evaluating the possibility of adopting the FIT tables in her/his organization. The *context* of the experiment consists of two *objects* – two Java systems – and of *subjects*, 13 students from a master course. All the material of the experiment (sources, documents, questionnaire, etc.) will be available for replications on a Website soon.

3.1 Hypotheses

The null hypotheses for the study are the following:

- H_{0a} The availability of FIT test cases does not significantly improve the correctness of the maintained source code.
- H_{0b} The availability of FIT test cases does not significantly affect the effort in the maintenance task.

The context in which we investigate the above question has the following characteristics: (1) system requirements have been written in detail, (2) automated acceptance tests have been pro-

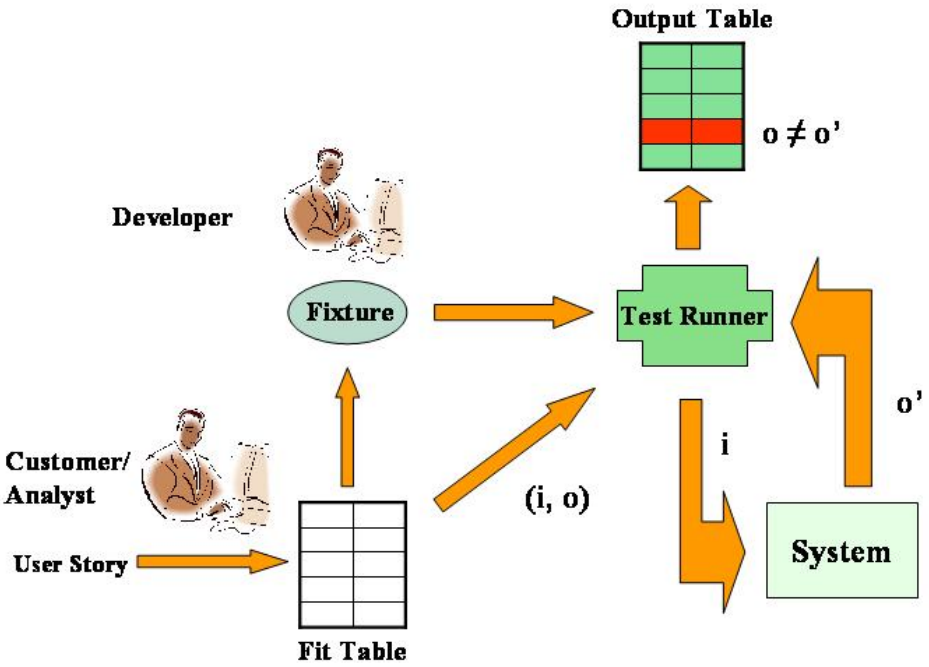

Figure 2: The complete testing process

duced in the form of FIT Tables and (3) some change requirements are expressed only in textual form while other include also an automated FIT test case.

3.2 Treatments

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

(+) textual change requirements enhanced with FIT tables and fixtures, thus enabling test case execution;

(-) only textual change requirements.

Other independent variables (not accounted in this paper) to be considered could be: the objects, the labs and the subjects' ability, if available.

3.3 Objects

The objects of the study are two simple Java programs realized by students: **LaTazza** and **AveCalc**.

LaTazza is a coffee maker management support application. LaTazza helps a secretary to manage the sale and the supply of small-bags of beverages (Coffee, Tea, Lemon-tea, etc.) for the

Table 1: Some Requirements for LaTazza.

R1	The secretary can sell small-bags of Coffee, Arabic Coffee, Tea, Lemon-tea and Camomile-tea. The cost of each small-bag is 0.62 euro. The secretary can select the kind of beverage and the number of small-bags and select the button *Sell*. If there are enough small-bags then the sale is done, otherwise the sale can not be done.
R2	The secretary can register a payment. She/He has to select the employee that perform the payment. This payment can extinguish a debt of the employee or it can used in future as advance fee. The payment must be > 0.
R3	The secretary can buy boxes of beverages. A box contain 50 small-bags of beverages all of the same kind (i.e, 50 coffee or 50 Arabic coffee, etc.). Each box cost 31 euro.
R4	The secretary can request the list of debtors with their debts.

Coffee-maker. The application supports two kinds of clients: visitors or employees (university employees and professors). Employees can purchase beverage cash or on credit, visitors only cash. The secretary can: sell small-bags to clients, buy boxes of beverages (a box contains 50 beverage of the same kind), manage credit and debt of the employees, check the inventory and check the cash account. The system consists of 18 Java classes for a total of 1121 LOCs. Its requirement document comprises 9 requirements (see Table 1 for the first four requirements) complemented with a total of 16 FIT tables.

AveCalc is a simple "desktop application" that manages an electronic register (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 represent the university credits) and a (optional) vote. An exam without vote is an exam not taken. The vote must be included between 0 and 30 (or equal). If the vote is $>= 18$ then the vote is positive, otherwise it is negative. It is possible to save the register and to load it (all data or only positive exams). AveCalc computes some statistics: average of the exams passed, total number of CFU, number of exams passed, (hypothetical) degree vote 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 LOCs. Its requirement document comprises 10 requirements complemented with a total of 19 FIT tables.

3.4 Population

The subjects were 13 students from the course of Laboratory of Software Analysis, in their last year of the master degree in computer science at the University of Trento. The subjects had a good knowledge about programming, in particular Java, and an average knowledge about software engineering topics (e.g. design, testing, software evolution). Subjects have been trained in meaning and usage of FIT tables and Fitnesse[3] with two theoretical lessons and two practical lessons (two hours each).

[3] Fitnesse is the tool that implement the FIT table approach used in the experiment

Table 2: Experimental design (S1 = LaTazza, S2 = AveCalc; + = with FIT tables, - = without FIT tables).

	Group A	**Group B**	**Group C**	**Group D**
Lab 1	S1+	S1-	S2-	S2+
Lab 2	S2-	S2+	S1+	S1-

3.5 Variables and experiment design

The dependent variables to be measured in the experiment are the *code correctness* and the *effort* required to perform the maintenance task. The code correctness is assessed by executing a JUnit acceptance test suite — developed by someone different from who developed the FIT tables — and measuring the percentage of test cases passed and failed. The effort was measured by means of time sheets (students marked start and stop time for each change requirements implemented). Time is expressed in minutes. Since the acceptance test suite for AveCalc was made up of 25 test cases, while the one for LaTazza included just 24 test cases, a derived measure was adopted: the fraction of test cases passed.

We adopt a balanced experiment design (see [WRH+00]) intended to fit two lab sessions (2-hours each). Subjects were split randomly into four groups, each one working in Lab 1 on all tasks of a system with a treatment and working on Lab 2 on the other system with a different treatment (see Table 2 for a graphical representation of the design).

3.6 Material and Procedure

As already mentioned, the test cases are written in the form of FIT tables and the supporting environment is a FitNesse wiki. The development environment is based on the Eclipse IDE with the FitNesse plugin[4]. For each group we prepared an Eclipse project containing the software and a FitNesse wiki with both requirements and change requirements. The projects were zipped and made available on a Web server. The experiment was introduced as a Lab assignment about FitNesse.

Every subject received:

- summary description of the application
- instructions to set-up the assignment (download the zipped Eclipse project, import it, and start the embedded Fitnesse server)
- a post experiment questionnaire

For each Lab the subjects had two hours available to complete the four maintenance tasks: CR1 - CR4 (see Table 3) . The first two change requirements (*corrective maintenance*) are very easy to implement, while the third and fourth require more work to locate the code to be changed and implementing the change (*evolution*). The maintenance/evolution tasks, for the two different systems, are very similar and we think of comparable difficulty.

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

Table 3: Change requirements for LaTazza.

CR1	There is an error in show debtors. Only employees with negative balance must be visualized. Fix the error.
CR2	There is an error in update employees. Not all the fields are updated. Fix the error.
CR3	The vendor of boxes of beverages changed his selling policy. Each five bought boxes one is added as a gift.
CR4	Change price of small-bags. Now the total price of the beverages that an employee would like to buy depends on (i) the number of small bugs bought (ii) if the beverage is seasonal or not. If a employee buys a number of small bags minor than 5 no discount is applied. If a employee buys a number of small bags included between 5 and 10 of a seasonal beverage, no discount is applied; but if the beverages are not seasonal a 1 euro discount is applied.

The post experiment questionnaire aimed at both gaining insights about the students' behavior during the experiment and finding justifications for the quantitative results. It included questions about the task and systems complexity, the adequacy of the time allowed to complete the task and the perceived usefulness of the provided FIT tables.

Before the experiment, subject were trained by means of introductory lectures (2 lessons 2 hours each) and laboratories (4 hours) on FIT. After subject were randomly assigned to the four groups, the experiment execution followed the steps reported below:

1. We delivered a sheet containing the description of the system.
2. Subjects had 10 minutes to read the description of the system and understand it.
3. Subjects had to write their name and start time on the delivered sheet.
4. Subjects had to download at the given URL the eclipse project and import it.
5. Subjects had to launch the Fitnesse wiki of the application.
6. Subjects had to write the stop time for installing the application.
7. For each change requirement (CR1-CR4):
   (a) Subjects had to fix the application code (LaTazza or AveCalc) in order to make the test cases pass (treatment +) or to satisfy the change requirement (treatment -).
   (b) Subjects had to record the time they use to apply change task (start/stop time).
8. Subjects were asked to compile the Post Experiment Questionnaire.

4 Experimental results

There were 13 subjects divided into three groups of three and one group of four. They took a median of 5 minutes to set up the environment and they worked for a median of 73 minutes on the tasks. The subjects deemed as complete an average of 2.75 tasks over four tasks assigned. The subjects worked on each task for a time ranging from 11 to 39 minutes with an average of 21 minutes. The distributions of passed tests and time required to complete tasks are not normal (Shapiro-Wilk test p=0.026 and p=$6.9 \cdot 10^{-6}$ respectively) therefore we will use the Mann-Whitney test for both hypotheses.

4.1 Data analysis

To test the first hypothesis (H_{0a}) we compared the number of acceptance tests passed by the program whose change requirements included FIT tables or not. The boxplot summarizing the percentage (expressed as fraction) of passed test cases is presented in Figure 3. The percentage of passed test cases for the Fit group is about 60% while that one for the Text only group is a little bit more than 40%. By applying a one-tailed Mann-Whitney test, we found this difference to be statistically significant (*p-value=0.03*), therefore we can reject the null hypothesis.

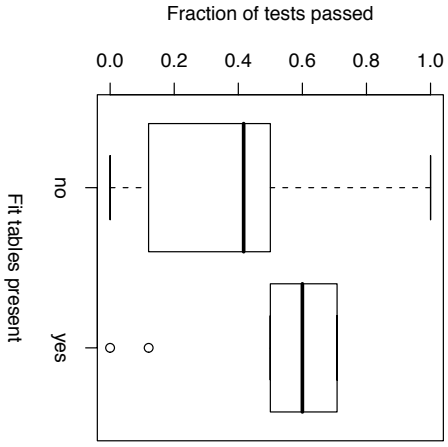

Figure 3: Boxplot of fraction of passed tests.

The second hypothesis can be tested by looking at the time required to complete the tasks. Since not all students completed all the tasks and since the tasks's difficulty varied both among tasks and systems, we analyzed the time for each task. Figure 4 shows the boxplot of times used by subjects to complete each task; filled boxes correspond to the presence of FIT tables. To test the second hypothesis we used a Mann-Whitney test. Table 4 reports the p-values of Mann-Whitney tests for each task. Overall in 5 cases out of 8 (see Figure 4) we observe a reduction of time (considering the median) when FIT tables are present but the only significant difference (highlighted in boldface in 4) is found for the first task on system AveCalc. With only these data we cannot reject the null hypothesis H_{0b}. Further experiments are necessary to answer our

second research question.

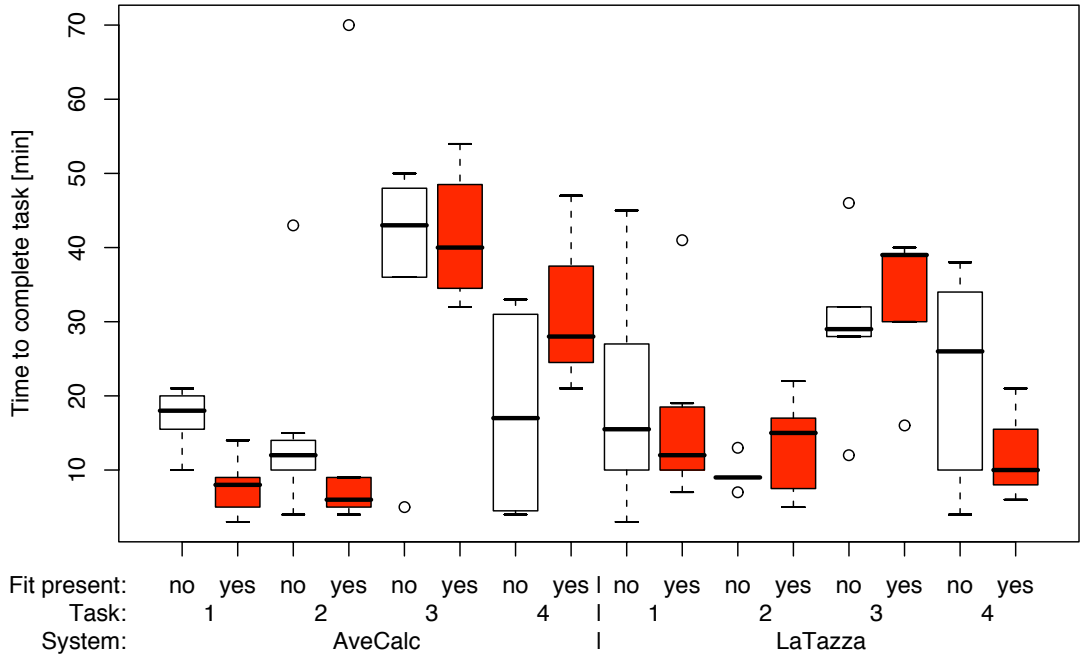

Figure 4: Boxplot of time required to complete task.

	System					
	AveCalc			LaTazza		
Task	p-value	median *yes*	median *no*	p-value	median *yes*	median *no*
1	**0.01**	8	18	0.83	12	15.5
2	0.33	6	12	0.57	15	9
3	1.00	40	43	0.53	39	29
4	0.63	28	17	0.45	10	26

Table 4: Analysis results on times to complete tasks.

4.2 Analysis of Survey Questionnaires

The analysis of the survey questionnaires that the subjects filled-in after each experiment can be useful to better understand the experimental results. In this paper the analyses are supported only by descriptive statistics. Answers are on a Likert scale [Opp92] from 1 (strongly agree) to 5 (strongly disagree).

Overall, all subjects agreed they had enough time to perform the tasks (*I had enough time to perform the lab tasks*, overall mean = 2.35) and the objectives were clear enough (*The objectives*

of the lab were perfectly clear to me, overall mean = 1.73). The description of the systems were clear (overall mean = 2.08) as the change requirements (overall mean = 2.35).

Similarly to Melnik *et al.* [MRM04], we can observe that the students deemed the FIT tables and the capability of running tests automatically useful enough. The possibility of executing FIT tables as tests was perceived useful for performing the change (*Running FIT tables are useful in maintenance/evolution tasks*, mean = 1.69). Moreover, FIT tables were also considered useful "per-se" to clarify change requirements (*FIT tables are useful to clarify change requirements*, mean = 1.92). See [RTCT07] for another experiment with students treating the research question: "FIT tables are able to clarify (change) requirements?".

4.3 Threats to Validity

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

Internal validity threats concerns external factors that may affect a dependent variable (in our case code correctness and effort). Since the subject had to perform a sequence of four tasks, a learning effect may intervene: the subjects were previously trained and the tasks were of progressively increasing difficulty, therefore we expect learning not to have influenced significantly the results. The experiment was proposed as an ungraded assignment within a university course, thus the student should not have been subject to evaluation apprehension.

Construct validity threats concern the relationship between theory and observation. It is possible that the Junit test suite does not provides and adequate means to measure the quality of change requirement implementation. We mitigated this risk by having the test suite and the Fit tables developed independently from different people.

Threats to *conclusion validity* can be due to the sample size (only 13 subjects) that may limit 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.

Threats to *external validity* can be related to (i) the simple Java system chosen and (ii) to the use of students as experimental subjects. We do not expect the absolute performance of students being at the same level as professionals, but we expect to be able to observe a similar trend of improvement. Another threat to external validity is that (iii) the results are limited to FIT-based acceptance test suites, which may be rather different from other approaches to acceptance testing. We don't think that obtained results could change using a different implementation of the FIT table based approach[5] or a different development environment. All the existing implementations of FIT are very similar to Fitnesse.

Further studies with larger systems and more experienced developers are needed to confirm or contrast the obtained results.

[5] for example FitLibrary, MavenFit, etc. See http://fit.c2.com/wiki.cgi?FitTools

5 Related Work

Although there are several papers [Aar06, RMM05] and books [MC05] describing acceptance testing with FIT tables, only a few works report empirical studies about FIT.

The most related work is the paper by Melnik *et al.* [MRM04]. It is a study focused on the use of FIT user acceptance tests for specifying functional requirements. It has been conducted at the University of Calgary and at the Southern Alberta Institute of Technology. In this experiment, the authors showed that the use of FIT tables and the possibility to execute them improve the comprehension of requirements.

Melnik *et al.* [MMC06] investigated whether acceptance tests can be authored effectively by customers of agile projects. Results show that customers can specify functional requirements clearly.

The paper [RTCT07] reports a controlled experiment with master students aimed at assessing the impact of FIT tables on the clarity of requirements. The results obtained indicate that FIT helps in the understanding of the requirements.

In another preliminary study [TRD07] some of the authors of the present paper found a statistically significant evidence that the availability of FIT tables allows the programmers to complete more maintenance tasks. However, they did not measure, as we did in the present study, whether completed maintenance tasks were correct.

6 Conclusion and Future Work

This paper reported a controlled experiment with 13 master students aimed at assessing the use of FIT executable acceptance test suites in the context of maintenance and evolution tasks.

The obtained results indicates that FIT tables significantly help developers to correctly perform the maintenance tasks. Other than looking at requirements, developers continuously execute FIT test cases to (i) ensure that FIT tables related to the change requirements passed and (ii) use requirement FIT tables to regression test the existing pieces of functionality.

Regarding productivity, FIT tables may or may not help: on the one hand, they provide a guideline to perform the maintenance tasks; on the other hand, they require time to be understood and executed. Further investigation is anyway necessary to answer our second research question.

Future work aims at replicating this study with a larger population of students, with professionals and by using larger and more realistic software systems. Also, other metrics (e.g., number of change requirements completed) and other factors such as subjects' ability and experience will be taken into account.

Acknowledgements: We thank all the students of the course of Laboratory of Software Analysis at the University of Trento who participated in the experiment. Without them this work would not have been possible.

Bibliography

[Aar06] J. Aarniala. Acceptance testing. In *whitepaper. www.cs.helsinki.fi/u/jaarnial/jaarnial-testing.pdf*. October 30 2006.

[MC05] R. Mugridge, W. Cunningham. *Fit for Developing Software: Framework for Integrated Tests*. Prentice Hall, 2005.

[MMC06] G. Melnik, F. Maurer, M. Chiasson. Executable Acceptance Tests for communicating Business Requirements: customer requirements. In *Proceedings of AGILE 2006 Conference (AGILE2006)*. Pp. 35–46. IEEE Computer Society, Los Alamitos, CA, USA, 2006.

[MRM04] G. Melnik, K. Read, F. Maurer. Suitability of FIT user acceptance tests for specifying functional requirements: Developer perspective. In *Extreme programming and agile methods - XP/Agile Universe 2004*. Pp. 60–72. August 2004.

[Opp92] A. N. Oppenheim. *Questionnaire Design, Interviewing and Attitude Measurement*. Pinter, London, 1992.

[RMM05] K. Read, G. Melnik, 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)*. Pp. Lecture Notes in Computer Science, Vol. 3556, Springer Verlag: 127–136 2005. June 18-23 2005.

[RTCT07] F. Ricca, M. Torchiano, M. Ceccato, P. Tonella. Talking Tests: an Empirical Assessment of the Role of Fit Acceptance Tests in Clarifying Requirements. In *9th International Workshop On Principles of Software Evolution (IWPSE 2007)*. Pp. 51–58. IEEE, September 2007.

[TRD07] M. Torchiano, F. Ricca, M. Di Penta. ”Talking tests”: a Preliminary Experimental Study on Fit User Acceptance Tests. In *IEEE International Symposium on Empirical Software Engineering and Measurement*. (to appear) 2007.

[WRH+00] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, A. Wesslén. *Experimentation in Software Engineering - An Introduction*. Kluwer Academic Publishers, 2000.