A Pilot Experiment to Quantify the Effect of Documentation Accuracy on Maintenance Tasks

Maurizio Leotta, Filippo Ricca, Giuliano Antoniol, Vahid Garousi, Junji Zhi, Guenther Ruhe

Abstract:

This paper reports the results and some challenges we discovered during the design and execution of a pilot experiment with 21 bachelor students aimed at investigating the effect of documentation accuracy during software maintenance and evolution activities. As documentation we considered: a high level system functionality description and UML documents. Preliminary results indicate a benefit of +15% in terms of efficiency (computed as number of correct tasks per minute) when a more accurate documentation is used. The discovered challenging aspects to carefully consider in future executions of the experiment are as follows: selecting "the right" documentation artefacts, maintenance tasks and documentation versions, verifying that the subjects really used the documentation during the experiment and measuring documentation-code alignment.

Digital Object Identifier (DOI):

http://dx.doi.org/10.1109/ICSM.2013.64

A Pilot Experiment to Quantify the Effect of Documentation Accuracy on Maintenance Tasks

Maurizio Leotta[1], Filippo Ricca[1], Giuliano Antoniol[2], Vahid Garousi[3,5], Junji Zhi[4], Guenther Ruhe[3,4]
[1] Dipartimento di Informatica, Bioingegneria, Robotica e Ingegneria dei Sistemi (DIBRIS), Università di Genova, Italy
[2] Département de Génie Informatique et Génie Logiciel (DGIGL), École Polytechnique de Montréal, Québec, Canada
[3] Dept. of Computer and Electrical Engineering, [4] Dept. of Computer Science, University of Calgary, Alberta, Canada
[5] Graduate School of Informatics, Middle East Technical University, Ankara, Turkey
maurizio.leotta@unige.it, filippo.ricca@unige.it, antoniol@ieee.org, vgarousi@ucalgary.ca, zhij@ucalgary.ca, ruhe@ucalgary.ca

***Abstract*—This paper reports the results and some challenges we discovered during the design and execution of a pilot experiment with 21 bachelor students aimed at investigating the effect of documentation accuracy during software maintenance and evolution activities. As documentation we considered: a high level system functionality description and UML documents. Preliminary results indicate a benefit of +15% in terms of efficiency (computed as number of correct tasks per minute) when a more accurate documentation is used. The discovered challenging aspects to carefully consider in future executions of the experiment are as follows: selecting "the right" documentation artefacts, maintenance tasks and documentation versions, verifying that the subjects really used the documentation during the experiment and measuring documentation-code alignment.**

***Keywords*—Documentation Accuracy; Maintenance Tasks; Controlled Experiment.**

I. Introduction

Several experiments have been conducted to investigate the costs and benefits associated with the UML usage during maintenance and evolution tasks [1], [2]. These works provide very clear insights in terms of the kinds of benefits that can be expected from using UML: for complex tasks and past a certain learning curve, it has been observed that the availability of UML documents could results in improvement of the functional correctness of changes as well as their design quality. These experiments [1], [2] were run providing the subjects (both students and professionals) with and without UML documents (use case, class and sequence diagrams), i.e., the treatment group had access to UML documents while the control group did not have any access. Both groups had access to high level system functionality description, manual pages, and source code.

However, these works did not address the problem of how *non-aligned UML documents* impacts on maintenance tasks. In practice, the experiments [1], [2] treat only the more extreme cases: no UML documents vs. (100%) aligned UML documents. The goal of our long term plan, about which this pilot study constitutes a first step, is to fill the gap left by the above papers, investigating the impact of partially *outdated UML documentation* on maintenance tasks. We believe that this kind of experiments is important because the *outdated documentation* problem is often the case in real-world projects in the software industry [3].

In this paper, we report the design and initial results of a pilot experiment[1] conducted with 21 bachelor students that aims to compare in a maintenance task scenario a "less" aligned technical documentation (i.e., designs documents) with a "more" aligned one. The contributions of this paper are two-fold: (1) the results of the experiment, and (2) a list of challenges we found in conducting the experiment.

The paper is structured as follows. Section II provides details on the design of the experiment. Section III presents some preliminary results, a list of non-trivial challenges that we encountered during the execution of the experiment, and threats to validity that could affect our results. Section IV concludes the paper with final remarks and future directions.

II. Experiment and Planning

We conceived and designed the experiment following the guidelines by Wohlin *et al.* [4]. The *goal* of this study is to analyse the effect of documentation accuracy on the efficiency of software maintenance tasks, for the *purpose* of investigating different levels of alignment (or up-to-dateness) between UML documents and code and their impact on maintenance tasks, from the *point of view* of software managers and developers in the *context* of maintaining a software system. In what follows, design and other aspects of the experiment are briefly discussed. The experimental material is available at `http://softeng.disi.unige.it/2013-DocAccuracy.php`.

A. Research Question and Hypothesis

The research question for the study is the following:

Does the level of alignment between UML documents and code have any impact on efficiency of maintainers?

From the above stated research question, we derived the following null hypothesis:

– H_{0a} The level of alignment between UML documents and code has no impact on efficiency of maintainers.

We will consider H_{0a} as a one-tailed hypothesis because we expect a positive effect of a more aligned documentation on the efficiency of software maintainers.

B. Context (Objects and Subjects)

Two Java desktop applications comparable in size — EasyMarks (a software for handling student marks composed by 17KLoCs and 89 classes) and AMICO (a software for

[1] A pilot study is often used to evaluate the design of the full-scale experiment which then can be adjusted.

condominium management composed by 14KLoCs and 162 classes) — are the objects of the study. The documentation of EasyMarks and AMICO were developed during two editions (2006 and 2008) of the Software Engineering course at the University of Genova [5] and used within the laboratory part of that course to implement the corresponding systems. These two documents underwent a number of modifications (11 for AMICO and 12 for EasyMarks) since their first version until the execution of experiment presented here. The documentation of each system consists of a high level system functionality description and UML documents (class and sequence diagrams). As source code for the experiment, we selected (and refined) the implementation from the best group for each course edition (i.e., 2006, 2008). Some essential comments to the code were added.

The subjects were 21 students from the Software Engineering course, in their last year of the bachelor degree in Computer Science at the University of Genova (Italy). The subjects had a good programming knowledge[2], especially Java programming, and an average UML knowledge (which was explained during the course).

C. Treatment and Experimental Design

Two cases can be distinguished: (i) maintenance tasks executed using a "*less*" aligned documentation and (ii) maintenance tasks executed using a "*more*" aligned documentation. Thus, only one independent variable occurs in our experiment, which is nominal. It assumes two possible values: **LESS** or **MORE**. During maintenance tasks, both students with LESS and MORE treatments had access to the high level system functionality description, UML documents and source code. The only different thing is the level of alignment of the UML documents (i.e., class and sequence diagrams). In one case, they are more aligned with the code in the other case less. In our experiment, we used the last version of the documentation of each system for the MORE treatment and selected a previous version with a "sufficient distance" for the LESS treatment. To compute the alignment between design documents and code, we applied a preliminary measure (see also Section III-A). For each UML class diagram belonging to the LESS and MORE treatments: (1) we count the total number of classes (*CsM*), (2) we count the classes in the diagram that are also in the system's source code (*CsM&S*) and for them (3) we compute the Jaccard index[3] to quantify the similarity between model and source code for each class in terms of fields and methods and (4) we calculate its average value over all the *CsM&S* classes (*AJaccard*) and (5) we compute a total alignment index defined as: *CsM&S * AJaccard / CsM*, and finally (6) we compute the distance between LESS and MORE in percentage. The total alignment index takes into account the number of classes that are both in the model and in the source code, their similarity computed with the Jaccard index and the total number of classes. The total alignment is equal to 1 if and only if all the classes in the model have a match in the code and the averaged Jaccard index is equal to 1[4]. For the UML sequence diagrams we did not compute the alignment, assuming a misalignment similar to the one computed for the class diagrams. The assumption is motivated by the observation that sequence diagrams have been built starting from class diagrams. Table I reports the alignment measures.

TABLE I. UML CLASS DIAGRAMS AND SOURCE CODE ALIGNMENT

System	UML Model Version		*CsM*	*CsM&S*			Total Alignment	Distance
				#	%	*Ajaccard*		
EasyMarks	1.3l	LESS	30	11	36,7	0,8707	0,3192	27,90%
	2.0	MORE	43	28	65,1	0,6271	0,4083	
AMICO	2.0	LESS	85	52	61,2	0,7127	0,4360	31,02%
	3.7.1	MORE	84	60	71,4	0,7998	0,5713	

The experiment adopts a counterbalanced design. This design was chosen because: it controls order effects, mitigates possible learning effects between labs and ensures that each subject works with both the treatments (LESS and MORE) in the two lab sessions. Subjects were split into four groups[5], each one working in Lab 1 on the maintenance tasks of a system with a treatment and working on Lab 2 on the other system with a different treatment. Between the two laboratory sessions a break was given.

For each lab, the subjects had three hours available to complete four maintenance tasks: MT1-MT4. We designed the maintenance tasks in increasing order of difficulty. The maintenance tasks, for the two different systems, are very similar (in practice, three feature modifications and one feature insertion/restructuring for both systems) and of comparable difficulty. For example, a sample maintenance task (i.e., MT1) for the AMICO system asks to simplify the already implemented rule that checks the correctness of the Italian fiscal code of the persons (something similar to the SSN in the United States).

D. Dependent Variables

The only one dependent variable to be measured in the experiment is *Efficiency* in performing maintenance tasks. It measures the number of correctly performed tasks per minute, and it is defined as:

$$OverallEfficiency = \sum_{i=1}^{4} Corr_i \; / \; \sum_{i=1}^{4} Time_i \quad (1)$$

As it can be noticed from the above formula, the efficiency sums over all (four) maintenance tasks.

The correctness $Corr_i = 1$ if the *i-th* maintenance task was correctly performed, 0 otherwise. The time was measured by means of time sheets. Students recorded start and stop time for each implemented maintenance task.

E. Material, Procedure and Execution

The development environment was Eclipse 4.2. For each group, we prepared a zip file containing an Eclipse project of the software system (EasyMarks or AMICO) and the related documentation. The zip files were made available on a Web server. The experiment was introduced as a laboratory assignment about software maintenance. Before the experiments, all the subjects have been trained with a two hours tool-demo.

Every subject received:

– an Eclipse project, a high level system functionality description and UML documents (class and sequence diagrams);

[2]We estimated it by means of a pre-experiment questionnaire.

[3]http://en.wikipedia.org/wiki/Jaccard_index

[4]Note that, to compute the total alignment, we did not consider the classes that are in the source code but not in the model, since, the UML documents have been used to locate the portions of source code to modify. Thus, we calculated the alignment of the model with respect to the source code and not vice versa.

[5]We equally distributed (as much as possible) high and low ability students among these groups.

– instructions to set-up the assignment (how to download the zipped Eclipse project, import and execute it);
– a sheet containing the four maintenance tasks.

For each Lab session, the experiment execution followed the steps reported below:

1) Subjects were required to download from a given URL the zip file containing the Eclipse project and the documentation.
2) Subjects were given 15 minutes to: (*i*) read the high level description of the system, (*ii*) import the corresponding project in Eclipse and (*iii*) execute it.
3) A sheet containing the four maintenance tasks was delivered.
4) For each maintenance task (MT1-MT4) subjects had:
   a) to execute the maintenance tasks (for EasyMarks or AMICO) on the Eclipse project surfing the UML documents.
   b) to record the time they need to execute the modification (start/stop time).

Finally, subjects were asked to compile a post experiment questionnaire[6], aimed at both gaining insights about the students' behaviour during the experiment and finding motivations for the quantitative results.

III. Preliminary Results

This section reports some preliminary results from the performed experiment, analysing only the effect of the treatment (LESS or MORE) on the Efficiency dependent variable. More detailed analyses (e.g., analysis of co-factors) and analysis of the post experiment questionnaires are not presented here for space reasons. Results of statistical tests are considered to be significant for a significance level of 95%.

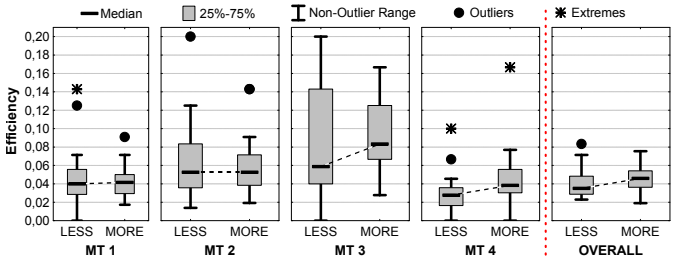

Fig. 1. Boxplots of Efficiency

Figure 1 summarizes the distribution of the Efficiency variable by means of boxplots. Observations are grouped by treatment and application and shown partitioning by maintenance task (MT1, MT2, MT3, MT4) and in aggregate form (Overall). The y-axis represents the efficiency computed as explained in the previous section. The boxplots show that the subjects achieved a better efficiency level when accomplishing the tasks with the MORE treatment. This is more evident for MT3 and MT4 tasks (see the slopes of the lines connecting the medians of the distributions in Figure 1).

We test the difference in efficiency with a paired non-parametric statistical test[7]. By applying a one-tailed Wilcoxon rank-sum test, we found the overall difference to be statistically significant (*p-value=0.04*), therefore we can reject our null hypothesis (H_{0a}).

[6] http://softeng.disi.unige.it/2013-DocAccuracy.php

[7] We opted for non-parametric tests because of the limited sample size.

TABLE II. Descriptive Statistics of the Efficiency Variable and One Way Wilcoxon Test

Exp	LESS				MORE				p-value
	Subjects	Median	Mean	SD	Subjects	Median	Mean	SD	
Overall	21	0.035	0.040	0.016	21	0.046	0.046	0.014	**0.04**
MT1	21	0.040	0.047	0.034	21	0.042	0.041	0.018	0.41
MT2	21	0.053	0.066	0.044	21	0.053	0.058	0.034	0.28
MT3	21	0.059	0.084	0.061	21	0.083	0.099	0.061	0.38
MT4	21	0.028	0.027	0.023	21	0.038	0.044	0.033	**0.009**

To complete the picture, Table II reports the essential descriptive statistics for the Efficiency variable. The variability, expressed as standard deviation, is mostly comparable across groups (except for MT1).

A. Discussion

Globally, the results confirm the general belief that a more aligned documentation helps more during maintenance tasks than a less aligned documentation, thus increasing our awareness on the benefit deriving from the usage of aligned UML documents. Looking at the means, students conducting the maintenance tasks with the MORE treatment obtained an overall benefit of +15%[8] in terms of efficiency. This improvement, that could appear small, has to be considered in relationship with the "distance" between the documentation versions. In fact, the bigger the distance between documentation versions is, the bigger the benefit in terms of efficiency should be. That distance, averaged by application and computed considering only class diagrams, is in our case 29.46% (mean of the values shown in the last column of Table I). Thus, in our case the subjects gained benefits in efficiency equal, in percentage, to approximately the half of the computed misalignment.

The study also opens a number of interesting discussion points and challenging aspects that we encountered during the design and execution of the experiment:

Documentation artefacts selection. During the design of the experiment, the first problem we had was which kind of documentation artefacts to consider and at which level of abstractness. Finally, we opted for a documentation composed by a high level system functionality description and UML diagrams (class and sequence diagrams) because this kind of documentation is often used in industry. Moreover, we decided to use software perspective UML diagrams at implementation level; these show in detail the classes in the system and how they interrelate. This last choice was done for two reasons: (1) because "low level" documentation is usually more used than "high level" one in maintenance tasks, (2) to simplify the documentation-code alignment measure (see below).

Maintenance tasks selection. It is difficult to select/find the "right" maintenance tasks for the experiment and fine-tune their difficulty/complexity. With the word "right" we mean tasks in which the selected documentation provides a concrete benefit. In particular, we would like that the two compared documentation versions differ in the portion used by the subjects to execute the maintenance tasks. In [1], the authors conclude that for complex tasks the availability of UML documents results in significant improvements of the functional correctness of maintenance tasks. On one hand, it is clear that the documentation is not very useful for really simple maintenance tasks but, on the other hand, it is not possible put too much stress on the experimental subjects with really

[8] Computed using the equation: $0.040 + 0.040x = 0.046$, see Table II.

complex tasks in a limited amount of time (we recall that in our case students had only 3 hours to complete the maintenance tasks). Moreover, measuring a priori the difficulty/complexity of a maintenance task is really difficult if not impossible. We designed the maintenance tasks in increasing order of complexity but checking it a posteriori we discovered that, for the students, MT3 resulted the simplest task (see Fig. 1). The problem of fine-tuning the difficulty of the maintenance tasks could explain why in our experiment only the difference in efficiency of completing MT4 was significant. We could speculate that the first three tasks were too simple for revealing a significant effect.

Documentation usage. It is difficult to measure how much the documentation has been really used by the subjects during the experiment. Clearly, the results of the experiment will be untrustworthy and useless if the documentation is not properly consulted and used during the maintenance tasks [3]. To try to measure it, we used a time tracking software[9], i.e., a software that allows its users to record time spent on different tasks. The tool we selected counts the time of the active window. In our case, this was only a partial successful solution because in some cases some students forgot to: (1) select the consulted window (and so render it active) and, (2) reset the time tracking tool when starting each task, thus, partially invalidating the tracking.

Documentation-code alignment measure. One of the problems we had was how to quantify the alignment between the two selected versions of the documentation and the code. We resorted to a simple, inaccurate and partial way to measure the alignment. However, several possible options are available to compute the alignment between UML class diagrams — e.g., one could use the algorithm proposed in [6] instead of using the Jaccard index or consider only the portions of the diagrams useful to execute the maintenance tasks and not the complete diagrams as we did (task related alignment vs. general alignment) — and it is not clear what is the more adapt for our problem. Moreover, given that the provided documentation includes also sequence diagrams, one should also compute the alignment for them and aggregate the results to obtain an overall measure. We omitted it in this preliminary work because computing the alignment between sequence diagrams and source code is not a trivial task and combining the measures of different types of alignments (class and sequence diagrams) is still more complex.

Documentation versions selection. Ideally, for a successful conduct of the experiment one should find/select two "right" documentation versions for the target maintenance tasks. Starting from the last version of the documentation (the more aligned), and selecting it as MORE treatment, there are two options: 1) artificially creating the LESS documentation with the wanted misalignment or 2) performing a careful manual analysis of all the available versions and select "the best one". Both the options have disadvantages. In the first case, we are not comparing two real versions, in the second one we can not execute the experiment with the wanted "distance" between the two versions. In this preliminary work we opted for the second solution but we are well-aware that the choice of the versions is extremely critical and different chosen versions may bring to different results of the experiment. Clearly, this aspect is related to the problem of selecting the maintenance tasks and measuring the alignment between versions.

[9]http://www.kviptech.com/time-tracking-software/free-time-tracker.shtml

B. Threats to Validity

This section discusses threats to validity that could have affected our results [4].

Internal validity threats. Since the students had to perform a sequence of four maintenance tasks, a learning effect may intervene. However, the students were previously trained and the chosen experimental design should limit this effect. Thus, we expect learning has not influenced too much the results.

Construct validity threats. We are well-aware that the adopted alignment measure is preliminary. However, the values in Table I correspond to our qualitative evaluation. Finally, the correctness assessment was performed by one of the authors who inspected the provided code and run a set of test cases. It is possible that the used test suite does not provide an adequate means to measure the quality of the maintenance tasks implementation.

Threats to *conclusion validity* can be due to the sample size (only 21 students) that may limit the capability of statistical tests to reveal any effect. We chose to use non-parametric tests due to the size of the sample.

Threats to *external validity* can be related to: (*i*) the choice of simple systems as objects, (*ii*) the use of students as experimental subjects and (*iii*) documentation used and tasks chosen. Further controlled experiments with larger systems and more experienced developers are needed to confirm or contrast the obtained results.

IV. Conclusion and Future Work

A pilot study is usually carried out before large-scale experiments, in an attempt to avoid time being wasted on an inadequately designed experiment. Thus, the result of our pilot have been twofold: (1) a first confirmation of the general belief that an aligned documentation helps during maintenance tasks and, (2) discovering a list of challenging aspects (and possible ways to face them) to consider in future. Future work will be devoted to: address the challenges that we faced and replicate this experiment in different contexts, i.e., with different applications and different subjects (e.g., professionals).

References

[1] E. Arisholm, L. C. Briand, S. E. Hove, and Y. Labiche, "The impact of UML documentation on software maintenance: An experimental evaluation," *IEEE Trans. Softw. Eng.*, vol. 32, no. 6, pp. 365–381, Jun. 2006.

[2] W. J. Dzidek, E. Arisholm, and L. C. Briand, "A realistic empirical evaluation of the costs and benefits of UML in software maintenance," *IEEE Trans. Softw. Eng.*, vol. 34, no. 3, pp. 407–432, May 2008.

[3] G. Garousi, V. Garousi, M. Moussavi, G. Ruhe, and B. Smith, "Evaluating usage and quality of technical software documentation: an empirical study," in *Proceedings of EASE 2013*. ACM, 2013, pp. 24–35.

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

[5] E. Astesiano, M. Cerioli, G. Reggio, and F. Ricca, "Phased highly-interactive approach to teaching UML-based software development," in *Symposium at MODELS 2007*. University of Goteborg, 2007, pp. 9–19.

[6] S. Kpodjedo, F. Ricca, P. Galinier, G. Antoniol, and Y.-G. Gueheneuc, "Madmatch: Many-to-many approximate diagram matching for design comparison," *IEEE Trans. Softw. Eng.*, vol. 39, no. 8, pp. 1090–1111, 2013.