Available online at www.sciencedirect.com

ScienceDirect

The Journal of Systems and Software 80 (2007) 1317–1327

The Journal of Systems and Software

www.elsevier.com/locate/jss

Evaluating performances of pair designing in industry

Gerardo Canfora [a], Aniello Cimitile [a], Felix Garcia [b,*], Mario Piattini [b], Corrado Aaron Visaggio [a]

[a] *RCOST – Research Centre on Software Technology, University of Sannio, Palazzo ex Poste, Viale Traiano, 82100 Benevento, Italy*
[b] *ALARCOS Research Group, Information Systems and Technologies Department, UCLM-Soluziona Research and Development Institute, University of Castilla-La Mancha, Paseo de la Universidad, 4, 13071 Ciudad Real, Spain*

Received 5 June 2006; received in revised form 31 October 2006; accepted 5 November 2006
Available online 19 December 2006

Abstract

Pair programming has attracted an increasing interest from practitioners and researchers: there is initial empirical evidence that it has positive effects on quality and overall delivery time, as demonstrated by several controlled experiments. The practice does not only regard coding, since it can be applied to any other phase of the software process: analysis, design, and testing. Because of the asymmetry between design and coding, applying pair programming to the design phase might not produce the same benefits as those it produces in the development phase. In this paper, we report the findings of a controlled experiment on pair programming, applied to the design phase and performed in a software company. The results of the experiment suggest that pair programming slows down the task, yet improves quality. Furthermore we compare our results with those of a previous exploratory experiment involving students, and we demonstrate how the outcomes exhibit very similar trends.
© 2006 Elsevier Inc. All rights reserved.

Keywords: Software engineering; Pair designing; Empirical studies

1. Introduction

Although software developers have been applying collaborative work in various forms for years (see Section 2), pair programming, one of the core practices of eXtreme Programming (Beck, 2000), has only recently attracted an increasing interest from practitioners and researchers. Advocates of pair programming claim that two people working together will develop as much functionality as two working separately, but their software quality will be better (Cockburn and Highsmith, 2001; Reifer, 2002). Detractors affirm that programming is slower and overall productivity decreases; quality improvement is also questioned, based on the argument that it depends on the maturity of the process and the skills of individuals, rather than on continuous cross-reviews (Rakitin, 2001).

Several structured experiments, most of them carried out in academic settings, have actually produced empirical evidence that pair programming can decrease delivery time, and increase the quality produced. As testing and removing errors are generally much more costly than coding, applying pair programming would lead to an increment of productivity.

According to previous results, the protocol of working in pairs promoted by the pair programming practice could be particularly helpful in the design phase, due to the fact that keeping effort and the produced quality within acceptable ranges, is not easy while performing design tasks, for at least two reasons:

– Software design requires dealing with many levels of abstraction: implementation, database, business logic, presentation, deployment, interaction with other systems,

* Corresponding author. Tel.: +34 926295300; fax: +34 926295354.
E-mail addresses: canfora@unisannio.it (G. Canfora), cimitile@unisannio.it (A. Cimitile), Felix.Garcia@uclm.es (F. Garcia), Mario.Piattini@uclm.es (M. Piattini), visaggio@unisannio.it (C.A. Visaggio).

0164-1212/$ - see front matter © 2006 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2006.11.004

and communication protocols. Mastering all these aspects and their integration is difficult especially in large systems: the larger the system, the more complex become analysis, communication, management, and maintenance of design products (Budgen, 2003).

– Design strategies and rationales are rarely dealt with in documentation and personnel turnover entails a severe loss of experience and knowledge which are difficult to replace. Design products maintenance requires a number of different views and diagrams in order to get the complete picture of the system's structure, behavior and functions (Clements et al., 2002; Ghezzi et al., 2003).

The application of working in pairs in the design phase is called *pair designing*: two designers work on the same design document, on the same machine and at the same time: the first designer denominated *driver*, actively writes the document and the other, denominated *observer*, reviews it. The two roles can be switched, should the need arise, during work: this usually happens when the driver does not know how to proceed, and when the observer has already elaborated a candidate solution for the problem. The observer can also accomplish different activities apart from reviewing, which might help to reach the goal of the current task.

In this paper, we explore to what extent pair designing can produce the same benefits, in terms of quality and effort, as that of pair programming, within an industrial setting. We also compare the results obtained from this empirical study, which involved professionals, with the results of a previous exploratory experiment, carried out in a University, which involved students (Canfora et al., 2006): we found that the outcomes exhibit very similar trends.

The paper proceeds as follows: Section 2 discusses related work. Section 3 outlines the characterization of the study. Section 4 presents the results of our study, and in Section 5 its validity is discussed. In Section 6, a comparison is made between our study and the academic exploratory experiment. Finally, Section 7 elucidates our conclusions.

2. Related works

Before pair programming became widespread as an Extreme Programming practice, Wilson et al. (1993) investigated *collaborative programming* in an academic environment. They found evidence that collaboration in pairs reduced problem-solving efforts, enhanced confidence in the solution and provided a better enjoyment of the process. Nosek (1998) confirmed the results in a controlled experiment, involving experienced developers, and he found that coupled developers spent 41% less time than individuals, and produced better codes and algorithms.

It is important to highlight that collaborative programming is not the same as pair programming. The former refers to a group of two or more people involved in coding, without adopting a specific working protocol; the latter is a practice involving only two people and with a precise protocol which prescribes to continuously overlapping reviews and the creation of artefacts. Williams et al. (2000) carried out one of the most well known experiments on pair programming, which involved the participation of senior software engineering students. By working in pairs, the subjects decreased development time by 40–50%, and passed more of the automated test cases; moreover, the results of the pairs varied less in comparison to those of the individual programmers. Several other investigations have highlighted the benefits of pair programming (Lui and Chan, 2003; McDowell et al., 2002; Srikanth et al., 2004; Williams and Kessler, 2003): effort is reduced and quality is improved. However, these results were not confirmed by two other experiments: the first experiment executed at the Poznan University (Nawrocki and Wojciechowski, 2001) demonstrated how pair programming was able to reduce rework, but it did not significantly reduce development time, and the second conducted by Heiberg et al. (2003) demonstrated how pair programming was neither more nor less productive than solo-programming. The relationship between pair programming and the geographic distribution of teams was explored by Baheti et al. (2002): distributed pair programming was comparable with co-located pair programming and fostered teamwork and communication within virtual teams. Other investigations have highlighted further benefits of pair programming, such as: fostering knowledge transfer (Williams and Kessler, 2000), in particular, leveraging of tacit knowledge, increasing job satisfaction (Succi et al., 2002), and enforcing student learning (Mendes et al., 2005; McDowell et al., 2002; Srikanth et al., 2004; Xu and Rajlich, 2005).

Conversely, few studies focus on pair designing. Al-Kilidar et al. (2005) carried out an experiment in order to compare the quality obtained by solo and pair work in intermediate design products. The experiment showed that pair design quality was higher than solo design quality in terms of the ISO 9126 sub-characteristics: functionality, usability, portability and maintenance compliance. Muller (2006) presented the results of a preliminary study that analysed the cost of implementation with pair and solo design; the results of this study suggested that no difference exists, assuming that the programs have similar levels of correctness. The authors also concluded that the probability of building a wrong solution into the design phase might be much lower for a pair than for a single programmer. In previous works, we investigated how pair designing affects knowledge building processes and we obtained evidence that pair designing helps diffuse and enforce knowledge within development teams (Bellini et al., 2005). We also carried out an exploratory experiment at the University of Castilla-La Mancha in order to evaluate pair designing in an academic environment (Canfora et al., 2006); the outcomes of this study are recalled in Section 6.

Despite the growing interest, practitioners can face difficulties to making informed decisions about whether or not to adopt pair programming, because there is little objective evidence of actual advantages in industrial environments. Most published studies are based in Universities and involve students as their subjects. Hulkko and Abrahamsson (2005) state that "the current body of knowledge in this area is scattered and unorganized. Reviews show that most of the results have been obtained from experimental studies in university settings. Few, if any, empirical studies exist, where pair programming has been systematically under scrutiny in real software development projects". The same applies to pair designing: more experimentation in industry is needed in order to build a solid body of knowledge about the usefulness of this technique as opposed to traditional solo designing.

3. The experiment in industry

Using the template for goal definition proposed by Wohlin et al. (2000), which is based on the GQM (Basili and Rombach, 1988; Fenton and Pfleeger, 1997), the goal of the experiment was defined as

– *Analyse* the practice of **pair designing**.
– *For the purpose of* **evaluating**.
– *With respect to* **effort** and **quality**.
– *From the point of view of* **designers**.
– *In the context of* a group of **professionals** of software development **Information Systems**.

From this objective the following research questions were formulated:

– Does pair designing require less effort than solo designing for a given task?
– Is pair designing better than solo designing in terms of the quality of the produced artefacts?

With the aim of answering these questions, an experiment in industry was carried out. The overall design of the experiment is illustrated in Fig. 1.

The experiment was carried out in a software company, Soluziona Software Factory, located in Ciudad Real, Spain. This company develops and maintains software systems for different domains: gas, water, and electricity management systems, management of quality and environment, market simulators, economic-financial management, corporative systems, public health systems, e-commerce, and telecommunications. Currently, Soluziona occupies a high-ranking position in the market of software professional services with a sales volume of almost 800 million euros and, after a long expansion period, the company has offices in 28 countries in four different continents. As a result of its quality-focused policy, Soluziona has recently reached level 3 maturity of CMMI and based on the results obtained in the last assessment, it plans to reach level 4 by the year 2007.

The CEO of the company wanted to verify if pair programming was as beneficial as claimed by many scientific and technical papers and whether or nor it could be extended to the design phase: they wanted to evaluate the practice by observing its use by the engineers employed in the company. Furthermore, the technical managers of the company were seeking new practices, methods, and

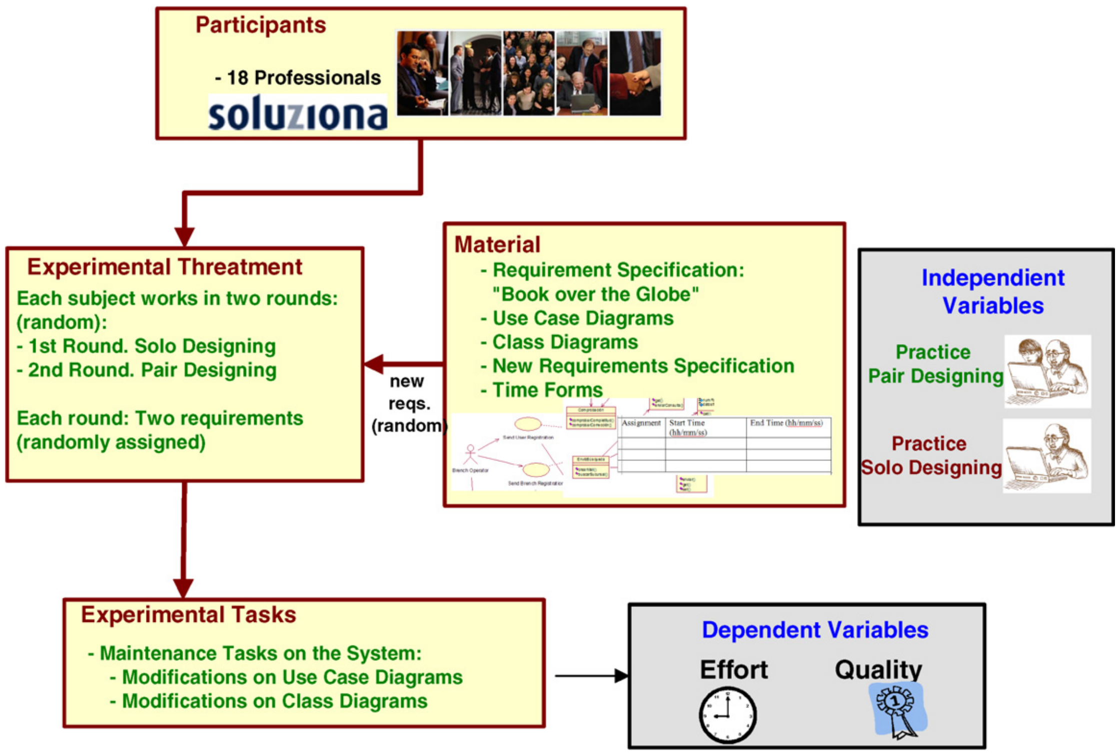

Participants
- 18 Professionals
soluziona
Experimental Threatment
Each subject works in two rounds: (random):
- 1st Round. Solo Designing
- 2nd Round. Pair Designing
Each round: Two requirements (randomly assigned)
new reqs. (random)
Material
- Requirement Specification: "Book over the Globe"
- Use Case Diagrams
- Class Diagrams
- New Requirements Specification
- Time Forms
Assignment
Start Time (hh/mm/ss)
End Time (hh/mm/ss)
Independient Variables
Practice Pair Designing
Practice Solo Designing
Experimental Tasks
- Maintenance Tasks on the System:
- Modifications on Use Case Diagrams
- Modifications on Class Diagrams
Dependent Variables
Effort
Quality

Fig. 1. Overview of the experimental design.

tools to improve software design, which could be quickly introduced into the processes of the company. In particular, the company was introducing (UML-based) Model-Driven-Engineering principles and tools. The culture of process innovation and continuous communication is very widespread in the company: as a matter of fact, it employs a large number of young engineers, the work areas are mainly open spaces and team members meet frequently in order to brainstorm and discuss problems which arise in a project or to debate new ideas for improving the adopted solutions. The actual tasks of designing, coding, and testing are however seen as individual duties.

We discussed with them the results of some experiments, documented in the literature, which demonstrated the benefits of pair programming, but they were not convinced for three reasons. The first concerned the *population* of the experiments: the majority of the experiments involved students, who they considered not reliable because of their immature skills, and limited experience and commitment. The second concerned the *context* in which the experiments were run; they pointed out that students were often selected and gathered together "just because they were in the same class" and in general, they had no previous experience in working together. On the contrary, professionals in a company develop, over time, a shared culture, with a common language and understanding, based on background tacit knowledge. And the third reason was that most of the literature available focused on the programming phase but not on the design phase, this being the phase that was of particular interest to the company. Such a scenario seemed to us appropriate for carrying out an experiment to evaluate the development time and quality of pair designing.

3.1. Participants

Some professionals of Soluziona, including engineers and scientists, volunteered to take part in the experiment. Eighteen were selected, to form a homogeneous sample in terms of skills and experience: they were familiar with UML, and have worked in the company for on average two years. Before running the experiment, a preliminary seminar was given to introduce pair programming and pair designing, and the subjects were given training, with lab sessions, in how to properly apply the practice during the experimental runs.

3.2. Material

The documentation was prepared by the experimenters and comprised:

- a textual requirement specification of a software system called "Book Over the Globe", whose function is the worldwide buying and selling of used books via the Internet;
- analysis and design documents of the system: 2 Use Cases and 2 Class Diagrams (including presentation, domain and persistence layers) with additional textual information;
- change requests: subjects received maintenance interventions for improving the existing system's features. The change requests had different scopes: change requests R2 and R4 affected only use cases while requests R1 and R3 affected both use case and class diagrams;
- timesheet forms, in order to collect the times that subjects employed in the different runs.

In Appendix A, an excerpt of the design documentation and assignment are provided.

3.3. Experimental tasks

The subjects were required to perform one assignment per run: each assignment was made up of two change requests on the design of an existing software system, which they had never seen before. The experiment consisted of a three step process, as depicted in Fig. 2:

- the preparatory run; each subject studied the documentation for 30 min, individually;
- the first run; five pairs of subjects were formed randomly, while the others eight subjects worked as individuals;
- the second run; those who had worked in pairs in the first run, performed individual designing in the second, while the individuals of the first run where paired in the second, so there were four random pairs and 10 subjects working individually.

As illustrated in Fig. 2, the experiment was arranged so that each subject worked both individually and in pairs and performed both the assignments, but in two different runs. For example, the subjects C and D worked together on assignment 2 in the first run, while, in the second run, they worked separately on assignment 1. The experiment took place in one of the rooms of the company that was equipped for pair work. The two assignments were randomly distributed to the individuals or pairs in the two different rounds and the experimenters were present in the room to control the execution of the experiment: they checked that the subjects properly followed the protocol of pair designing.

3.4. Variables

The independent variable was the applied practice: pair designing or solo designing. The dependent variables were

- *Effort*, which was measured as the time taken by the subjects (in pairs or individually) to complete the assignments. These values were obtained from the assignation forms provided, in each run, to the subjects. Subjects were asked to take note of the exact time when they started and when they finished the implementation of

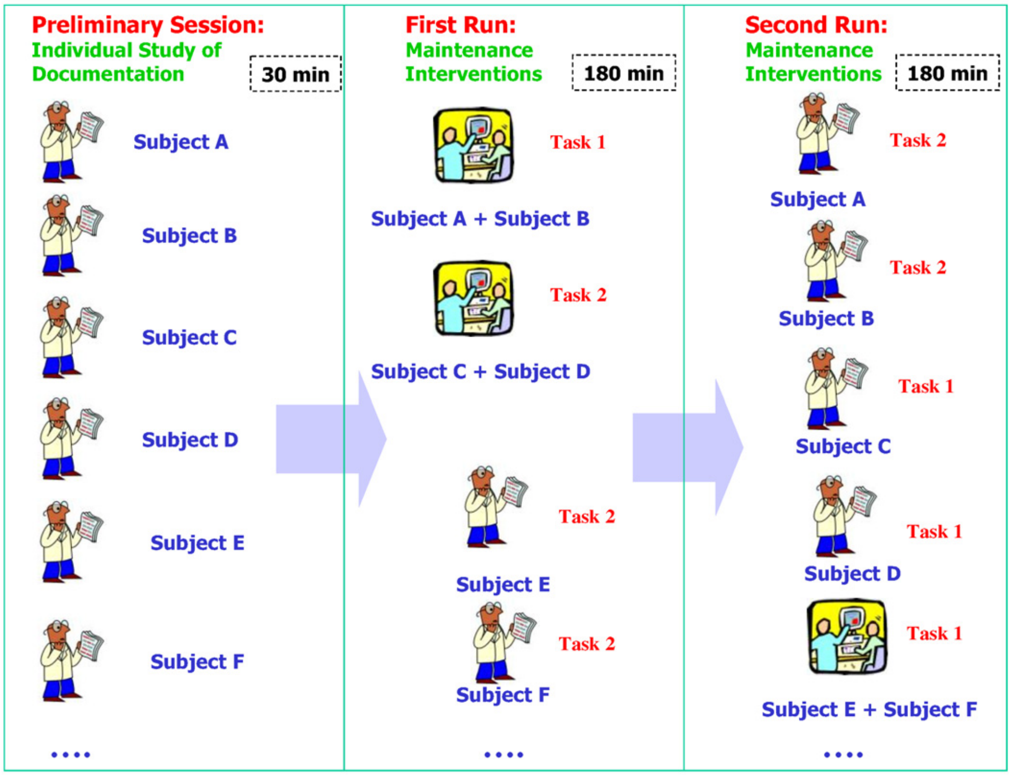


Fig. 2. Experimental tasks.

each task; this data was written on the time form for each run; for accuracy sake, all the subjects referred to their own computer system's clock. Therefore, this measure of the dependent variable was quantitative and objective.

– *Quality*. This variable was measured by rating the modified design artefacts delivered by the subjects. As previously stated, the four change requests were grouped into two assignments (one per run). Change requests 1 and 3 involved modifications on both the use case and class diagrams and change requests 2 and 4 required the modification of only the use case diagrams. Each assignment included one change request of each type and each diagram modification intervention (use case or class diagram) was rated in accordance with a scale composed of three values: 0 (incorrect), 0.5 (neither incorrect nor completely correct) and 1 (correct). So the highest possible rate for each assignment was 3 (one point for the right modification of a change request involving only a use case diagram and two points for the change request affecting use case and class diagrams). In order to reduce the subjectivity of this evaluation, two independent evaluators were involved and, whenever differences arose, a joint review was performed.

3.5. Hypotheses

The experiment was executed with the purpose of testing the following set of hypotheses:

- Null Hypothesis, H_{0a}: there is no difference in the effort employed between pair and solo designing,

  $\mu_{\text{effort\_solo}} = \mu_{\text{effort\_pair}}$
- Alternative Hypothesis, H_{1a}: there is a difference in the effort employed between pair and solo designing,

  $\mu_{\text{effort\_solo}} \neq \mu_{\text{effort\_pair}}$
- Null Hypothesis H_{0b}: there is no difference in the quality produced between pair and solo designing

  $\mu_{\text{quality\_solo}} = \mu_{\text{quality\_pair}}$
- Alternative Hypothesis H_{1b}: there is a difference in the quality produced between pair and solo designing

  $\mu_{\text{quality\_solo}} \neq \mu_{\text{quality\_pair}}$

4. Results of the experiment

4.1. Descriptive statistics

Tables 1 and 2 provide a detailed characterization of the data sets and Figs. 3 and 4 illustrate a comparison of the results obtained by paired and individual designers for the quality and effort variables.

Looking at the data set for quality and time, two facts emerge: pair designing helps to increase the quality achieved, but this entails an increment of effort to complete the task. Since in the second run subjects increased the

Table 1
Statistical indicators for effort

Statistical indicator	First round		Second round	
	Pairs	Solos	Pairs	Solos
Avg	6.583333	7.083333	8.5	5.2
Standard deviation	3.105628	3.800917	1.977142	2.020726
Mode	7	12	11	4
Max	13	12	11	11
Min	2	2	6	2

Table 2
Statistical indicators for quality

Statistical indicator	First round		Second round	
	Pairs	Solos	Pairs	Solos
Avg	1.458333	1.333333	1.75	0.979167
Standard deviation	0.381881	0.258199	0.223607	0.772086
Mode	1.5	1.5	2	1.5
Max	2	1.5	2	2
Min	0.75	1	1.5	0

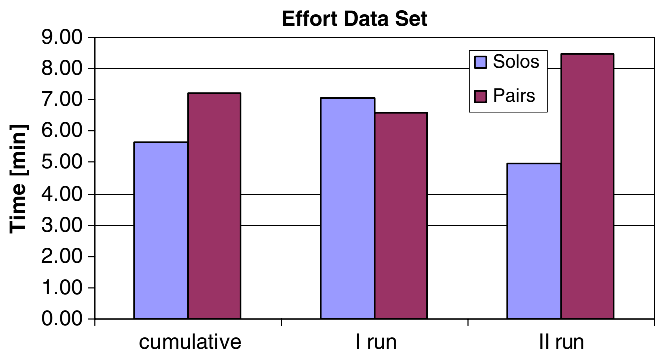


Fig. 3. Comparison of quality produced by paired and individual designers.

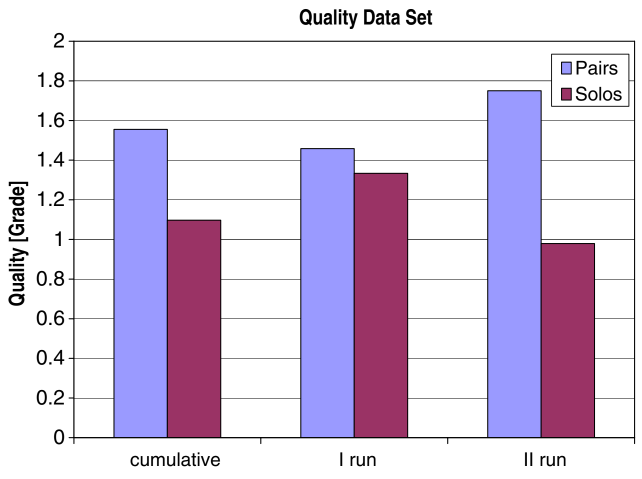


Fig. 4. Comparison of effort employed by paired and individual designers.

number of reviews and the discussions, both quality and effort were higher than in the first run. This suggests that by using pair designing, the company might, in the future, benefit from the standpoint of quality, but could experience an increment of effort required. From the outcomes of this experiment it is not possible to infer the general trend of quality or effort; consequently, it is not possible to establish whether the curve grows indefinitely or stabilizes at a certain speed. It could be possible to understand the shape of this curve by analyzing data from several replications of similar experiments, which will be considered in the future.

Fig. 5 provides a comparison between the standard deviation for quality and effort for solos and pairs; standard deviation describes the dispersion of data set around the central value of the sample, and it is considered a good indicator of predictability.

As can be observed in Fig. 5, standard deviation is lower in the pairs' data set than in that of the individuals.

Working in pairs seems to improve predictability of quality and effort. Pair designing leverages the performance profiles of subjects, which usually show many positive and negative peaks. Role switching reduces the latency time which occurs when a problem arises; this time is roughly composed of the time taken in searching for alternative strategies and the time needed to select the more suitable one. Besides reviewing the work of the driver, the observer analyses, in advance, the possible obstacles the pair may encounter, and elaborates the related possible solutions; as a result there is an overlapping of search time and selection time which has positive effects on quality. However, continuous discussion, regarding the strategy to adopt for dealing with a problem or the way to remove a defect, helps to share and leverage knowledge among developers.

4.2. Data analysis

Tables 3 and 4 report information concerning the tests of the hypotheses H_{0a} and H_{0b}. Mann–Whitney tests were used because data set distribution was not normal and the p-level was fixed at 0.05.

From the statistical tests it emerges that:

- the H_{0a} hypothesis can only be rejected with regards to the data set corresponding to the second run;

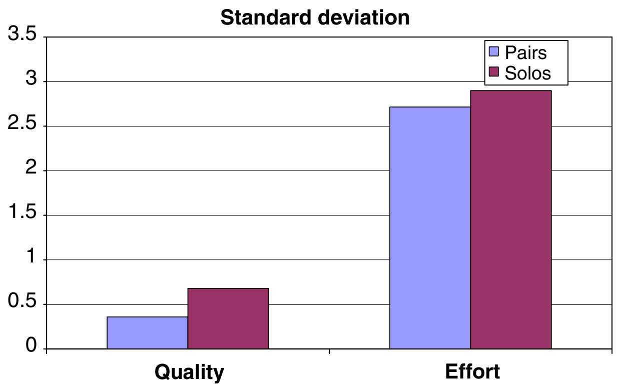


Fig. 5. Comparison of standard deviation for quality and effort.

Table 3
Statistical tests for effort data set

Testing	Rank sum (a)	Rank sum (b)	*p*-level	Comment
Pair(a)–Solo(b) cumulative	1537.00	1091.00	**0.011516**	There is **evidence** that solos outperformed pairs
Pair(a)–Solo(b) First run	231.000	435.000	0.76111	There is no evidence that pairs outperformed solos in the first run
Pair(a)–Solo(b) Second run	343.000	323.000	**0.00004**	There is **evidence** that pairs spent more time than solos in the second run
First run(a)–Second run(b)	941.000	1075.00	**0.003143**	There is **evidence** that results of the first run are different from those of the second run
Pairs(a)–Solos(b) in the assignment 1	450.000	291.000	0.079412	There is no evidence that the pairs required less time than the solos in completing the first assignment
Pairs(a)–Solos(b) in the assignment 2	335.000	260.000	0.056773	There is no evidence that the pairs required less time than the solos in completing the second assignment

Table 4
Statistical tests for quality data set

Testing	Rank sum (a)	Rank sum (b)	*p*-level	Comment
Pair(a)–Solo(b) cumulative	402.000	264.000	**0.019215**	There is **evidence** that the pairs outperformed the solos
Pair(a)–Solo(b) First run	122.000	49.000	0.371094	There is no evidence that the pairs outperformed the solos in the first run
Pair(a)–Solo(b) Second run	92.000	79.000	**0.039353**	There is **evidence** that the pairs outperformed the solos in the second run
First run(a)–Second run(b)	2387.00	3499.00	**0.003525**	There is **evidence** that results of the first run are different from those of the second run
Pairs(a)–Solos(b) in the assignment 1	435.000	231.000	**0.000423**	There is **evidence** that the pairs outperformed the solos while implementing the first assignment
Pairs(a) –Solos(b) in the assignment 2	441.000	225.000	**0.000633**	There is **evidence** that the pairs outperformed the solos while implementing the second assignment

- pair designing and solo designing did not produce significant differences regarding the time spent on the single assignment, as illustrated in the fifth and the sixth rows of Table 3;
- the H_{0b} hypothesis can only be rejected with regards to the data set corresponding to the second run;
- pair designing produced significantly higher quality than solo designing on each assignment, as illustrated in the fifth and the sixth rows of Table 4;
- the differences between the first and the second run are statistically significant, both for the effort and for the quality data set. These findings support the conjecture that the effects of pair designing are emphasized in the second run rather than in the first.

5. Limits of the experiment

The experiment presented in this paper has some limits that must be taken into account in order to understand to what extent the results are valid and how they can be used. Referring to the taxonomy of Wohlin et al. (2000), a discussion about validity threats follows.

- *Threats to Construct validity*: We measured quality (the dependent variable) by rating, which is a subjective measure. In order to increase the objectivity of the evaluation, all the artefacts underwent two independent reviews; whenever they differed, the reviewers accomplished together a joint review of the artefact.
- *Threats to Internal Validity*: The following issues were dealt with:
  – *Differences among subjects*: Using a within-subjects design, error variance due to differences among subjects is reduced. In this experiment all the professionals were familiar with the kind of work assigned.
  – *Learning effects*: The subjects executed introductory runs before the experiment, in order to become familiar with the practice of pair designing. It was not possible to avoid learning effects during the experiment: as a matter of fact, significant differences between the outcomes of the runs were detected (see row 4 of Tables 3 and 4). However, this did not invalidate the results of the experiment, as the analyses concerned pairs and solos of the same run.
  – *Fatigue effects*: On average the experiment lasted a short enough time to avoid fatigue thus making its effects irrelevant.
  – *Persistence effects*: In order to avoid persistence effects, the experiment was run with subjects who had never done a similar experiment.
  – *Subject motivation*: The participants were volunteers, thus assuring subjects with a high motivation. Professionals showed a great interest in taking part in a scientific experiment.
  – *The experimental package*: In order to clearly observe the results of the treatments on the subjects' performances, the assignments should be comparable in terms of effort and quality. If this condition is not respected, the differences among the assignments could be confounding factors and prejudice the analysis. Mann–Witney statistical tests confirmed that there is no significant difference between the two

Table 5
Statistical comparison of assignments

Testing	Rank sum (a)	Rank sum (b)	*p*-level	Comment
A1(a)–A2(b) effort	1301.500	1326.000	0.3348	There is no evidence that the effort data depends on the assignment
A1(a)–A2(b) quality	20228.000	18552.000	0.2111	There is no evidence that the quality data depends on the assignment

assignments, both in effort and in quality. Table 5 illustrates the details of the test. As a consequence, the results are independent from the experimental package.

- *Threats to External Validity*:
  - *Materials and tasks used*: Experimenters prepared the documentation of system design. Although the subjects worked during the experiment with an existing system, the tasks were limited to the available time. Therefore real scenarios must be considered, as they are supposed to be more complex and articulated.
  - *Subjects*: Professionals are helpful for enforcing external validity. Unfortunately, the subjects had no experience in pair programming or pair designing; therefore they were properly trained in order to participate in the experiment. Since the practice needs the capability to work in strict collaboration, we selected employees with a shared culture, i.e. engineers who: were familiar with the same processes, worked in the same teams, took part in the same projects and had similar skills and a similar professional background.

6. Comparison with the previous exploratory experiment in Academia

One of the most controversial issues in the area of empirical software engineering is the suitability of students as subjects in controlled experiments. Experiments with students as subjects can seriously threaten external validity given that the results have little probability of being generalized in industry. However, students are more often used as subjects in comparison to professionals in controlled experiments (Sjøberg et al., 2002). This is due to the fact that it is difficult to find professionals for empirical studies, whereas students are more accessible, easier to organize, and cheaper.

On the other hand, as reported by Host et al. (2000), it can be a too simplistic view to disregard experiments which use students. The important thing is to understand when students are suitable and how the results may be generalized. As a matter of fact, students can play a very important role in experimentation in the field of software engineering (Basili et al., 1999; Kitchenham et al., 2002).

We contrasted the experiment involving professionals, reported in this paper, with an exploratory experiment involving students, undertaken in the same conditions at the University of Castilla-La Mancha in Spain (Canfora et al., 2006). The material, experimental tasks and dependent and independent variables in the two experiments were identical. As a matter of fact, the experiment in industry was a replica of the experiment in academia with the sole exception of the variation of the context variables (professionals as the subjects) and the environment in which the solution was evaluated (Basili et al., 1999).

The subjects of the University experiment were two groups of students of the Department of Computer Science: the first group was composed of 29 students enrolled in the final-year (third) of the Computer Science (B.Sc.) degree course which specialised in Management and the second group was composed of 41 students enrolled in

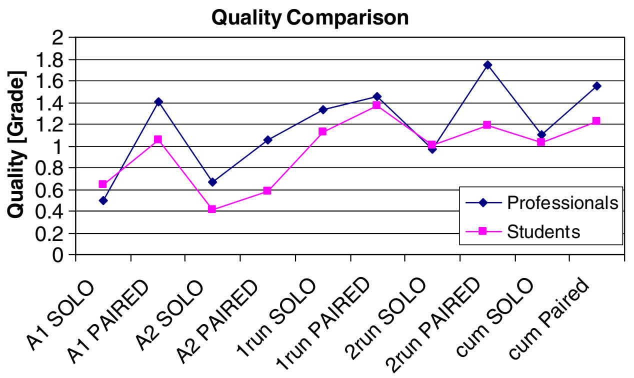


Fig. 6. Comparison of students and professionals quality results.

Table 6
Comparison of statistical test results of students and professionals

Test	Professionals	Students
Time Pair–Time Individuals (Cumulative)	✓ ($p = 0.0115$)	✓ ($p = 0.0103$)
Time Pair–Time Individuals (Run I)	($p = 0.7611$)	($p = 0.9890$)
Time Pair–Time Individuals (Run II)	✓ ($p = 0.0000$)	✓ ($p = 0.0108$)
Quality Pair–Quality Individuals (Cumulative)	✓ ($p = 0.0192$)	✓ ($p = 0.0462$)
Quality Pair–Quality Individuals (Run I)	($p = 0.3710$)	✓ ($p = 0.0007$)
Quality Pair–Quality Individuals (Run II)	✓ ($p = 0.0393$)	✓ ($p = 0.0928$)

the final-year in the Computer Science (B.Sc.) degree course which specialised in Systems.

This experiment produced similar results from both the students and the professionals. With regard to quality, the results of the students and the professionals showed the same pattern, as illustrated in Fig. 6. The same analysis was made for the effort variable with similar findings.

As far as the statistical significance of results is concerned, the two experiments produced similar results, as shown in Table 6. These results provide some evidence which supports the fact that students can be useful as subjects and in certain contexts it is possible to generalize their results to industry.

Although more experiments must be performed to obtain more solid evidence, it seems that pair designing helps to improve design quality, regardless of the experience of the designers.

7. Conclusion and future work

In this paper, we have investigated the difference between pair designing and individual designing in terms of the time required to perform a software task and the quality of task deliverables. An experiment with professional developers in a Spanish company suggests that pair designing slows down the task, but improves quality. According to our study, pair programming applied to the design phase appears less efficient than when it is applied to the coding phase as reported in Section 2. Only in the first run, the time required to complete the task is less for the pairs than for the individuals, yet still no way near the level of 50% reported by Williams et al. (2000). Conversely, quality improvement is higher than the 15% reported by Williams et al. (2000). Finally, pair designing is more predictable than individual designing with regards to quality, but it decreases the predictability of development time.

The findings of the experiment suggest that the practice of working in pairs is not limited to writing code in agile processes. Such a practice might be adopted also for successfully working at design documentation, and not necessarily only within agile processes. Since the experimental subjects were professionals with a good degree of experience, the conclusions of the experiment can be considered valid enough for the professional population. These results confirmed the results previously obtained in an academic exploratory experiment. However, some limits of the experimentation concerned of the fact that is was controlled study: small time windows and the kind of tasks.

Some issues remain still open:

– The measurement of the quality dependent variable can be enhanced in order to provide a more complete picture of this factor, as done by Al-Kilidar et al. (2005). In this research work, the quality was evaluated by the functional compliance of the modified artefacts given by subjects. According to our experimental design, since the modifications had a limited scope, other measures like usability or maintainability of the UML designs were not significant. In the future, a new design will be used in order to measure the functionality, usability and maintainability of the resulting artefacts for which a set of measures to evaluate the Use Case Diagrams and Class Diagrams (Genero et al., 2005) will be applied.
– As the practice could be adopted in different kinds of processes, the features of each operative context could affect the performances of pair designing. Consequently the relationship, between the characteristics of the process and the practice, should be better understood.
– The data set was collected in a limited time window, as it was an experiment in vitro. Since the practice could produce different performances, in the long-term, experiments in vivo could be helpful in building a complete picture of the matter.

According to the issues previously outlined, we are planning to carry out a family of experiments, which will include also experiments in vivo, with the twofold aim of: (i) obtaining a more complete evaluation of the quality variable, and (ii) enforcing the generalization of our conclusions.

Acknowledgements

This research was partially supported by the projects: ESFINGE (Dirección General de Investigación of the Ministerio de Educación y Ciencia, TIN2006-15175-C05-05); MECENAS (Junta de Comunidades de Castilla-La Mancha, Consejería de Educación y Ciencia, PBI06-0024); and FAMOSO, partially funded by Ministerio de Industria, Turismo y Comercio, FIT-340000-2006-67 Plan Nacional de Investigación Científica, Desarrollo e Innovación Tecnológica 2004–2007 and "Fondo Europeo de Desarrollo Regional (FEDER)", European Union.

Appendix A

Time Form:

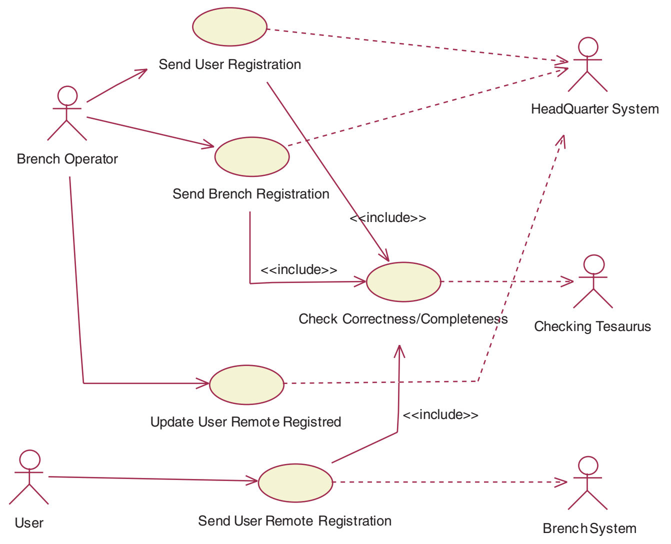
Name of the subject: ___________________________
If in a pair, the name of your companion:_______________
Number of the Run:______________

Assignment:

Assignment	Start Time (hh/mm/ss)	End Time (hh/mm/ss)

Assignments

R.1 The check for correctness/completeness must be launched by the Branch Operator with a button on the interface form.
R.2 The "Send User Registration" is realised only if the Branch is registered.
R.3 The Branch Operator can synchronise the Branch System with the HeadQuarterSystem by launching an appropriate program.
R.4 The search for a book should happen not only in the BranchSystem, but also in the HeadQuarterSystem. An excerpt of use case diagrams and specification.



Use case	Send User registration
Description	The Branch operator inserts data into the registration form, provided by the user. Validation of the form is launched
Exceptions	The form is not correct or complete. The sending of data is successful
Actors	BranchOperator, HeadQuarterSystem
Use case extends	Nn
Use case uses	Check correctness/completeness
Use case inputs	Name, address, offered books list (in case the user is a vendor) with specifications: title, author, publisher, language, publishing year, ISBN
Use case outputs	Recording of data of the new user
Criterion of acceptance	Data of the new user is stored in the database of the Local Branch
Related expectations	Database management system. Correctness and completeness checks. Data sending to the Headquarter
Related Reqs/ use cases	Check correctness/completeness

References

Al-Kilidar, H., Parkin, P., Aurum, A., Jeffery, R., 2005. Evaluation of effects of pair work on quality of designs. In: Proceedings of the 2005 Australian Software Engineering Conference (ASWEC 2005) Brisbane Australia. IEEE CS Press, pp. 78–87.

Baheti, P., Gehringer, E., Stotts, D., 2002. Exploring the efficacy of distributed pair programming. Proceedings of the Second XP Universe and First Agile Universe Conference on Extreme Programming and Agile Methods – XP/Agile UniverseLNCS, vol. 2418. Springer, pp. 208–220.

Basili, V., Rombach, H., 1988. The TAME project: towards improvement-oriented software environments. IEEE Transactions on Software Engineering 14 (6), 758–773.

Basili, V., Shull, F., Lanubile, F., 1999. Building knowledge through families of experiments. IEEE Transactions on Software Engineering 25 (4), 456–473.

Beck, K., 2000. Extreme Programming Explained: Embrace Change. Addison Wesley.

Bellini, E., Canfora, G., García, F., Piattini, M., Visaggio, C.A., 2005. Pair designing as a practice for enforcing and diffusing design knowledge. Journal of Software Maintenance and Evolution: Research and Practice 17 (6), 401–423.

Budgen, D., 2003. Software Design, second ed. Pearson Educational Limite.

Canfora, G., Cimitile, A., Visaggio, C.A., Garcia, F., Piattini, M., 2006. Performances of pair designing on software evolution: a controlled experiment. In: Proceedings of the 10th European Conference on Software Maintenance and Reengineering, CSMR 2006, 22–24 March, Bari, Italy, pp. 197–205.

Clements, P., Kazman, R., Klein, M., 2002. Evaluating Software Architectures: Methods and Case Studies. Addison–Wesley.

Cockburn, A., Highsmith, J., 2001. Agile software development: the business of innovation. Computer 34 (9), 120–123.

Fenton, N., Pfleeger, S.L., 1997. Software Metrics: A Rigorous and Practical Approach, second ed. International Thomson Computer Press, London, UK.

Genero, M., Piattini, M., Calero, C. (Eds.), 2005. Metrics for Software Conceptual Models. Imperial College Press.

Ghezzi, C., Jazazery, M., Mandrioli, D., 2003. Fundamentals of Software Engineering, second ed. Prentice Hall.

Heiberg, S., Puus, U., Salumaa, P., Seeba, 2003. A. Pair programming effect on developers productivity. In: Proceedings of Extreme Programming and Agile Processes in Software Engineering, Italy, May 2003, pp. 215–224.

Host, M., Regnell, B., Wohlin, C., 2000. Using students as subjects – a comparative study of students and professionalsLead-Time Impact Assessment Empirical Software Engineering, vol. 5. Kluwer Academic Publishers, pp. 201–214.

Hulkko, H., Abrahamsson, P., 2005. A multiple case study on the impact of pair programming on product quality. In: Proceedings of the 27th International Conference on Software Engineering (ICSE'05), May 15–21, 2005, St. Louis, Missouri, USA, pp. 495–504.

Kitchenham, B., Pfleeger, S., Pickard, L., Jones, P., Hoaglin, D., El Emam, K., Rosenberg, J., 2002. Preliminary guidelines for empirical research in software engineering. IEEE Transactions on Software Engineering 28 (8), 721–734.

Lui, K., Chan, K., 2003. When does a pair outperform two individuals? In: Proceedings of XP 2003LNCS. Springer-Verlag, pp. 225–233.

McDowell, C., Werner, L., Bullock, H., Fernald, J., 2002. The effects of pair-programming on performance in an introductory programming course. In: Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education. ACM, Cincinnati, KY, USA, pp. 38–42.

Mendes, E., Al-Fakhri, L.B., Luxton-Reilly, A., 2005. Investigating pair-programming in a 2nd-year software development and design computer science course. In: Proceedings of the 10th Annual SIGCSE

Conference on Innovation and Technology in Computer Science Education (ItiCSE'05). ACM Press, pp. 296–300.

Muller, M., 2006. A preliminary study on the impact of a pair design phase on pair programming and solo programming. Information and Software Technology 48, 335–344.

Nawrocki, J., Wojciechowski, A., 2001. Experimental Evaluation of pair programming. In: Proceedings of the European Software Control and Metrics Conference (ESCOM 2001). ESCOM Press, 2001, pp. 269–276.

Nosek, J., 1998. The case for collaborative programing. Communication of ACM 41 (3), 105–108.

Rakitin, S.R., 2001. Manifesto elicits cynism. Computer 34 (12), 4.

Reifer, D.J., 2002. How good are agile methods? IEEE Software 19 (4), 16–19.

Sjøberg, D., Anda, B., Arisholm, E., Dybå, T., Jørgensen, M., Karahasanovic, A., Koren, E.F., Vokác, M., 2002. Conducting realistic experiments in software engineering. In: Proceedings of the 2002 International Symposium on Empirical Software Engineering (ISESE'02). IEEE CS Press, pp. 17–26.

Srikanth, H., Williams, L., Wiebe, E., Miller, C., Balik, S., 2004. On pair rotation in the computer science course. In: Proceedings of Conference on Software Engineering Education and Training 2004. IEEE CS Press, Norfolk, VA, USA, pp. 144–149.

Succi, G., Marchesi, M., Pedrycz, W., Williams, L., 2002. Preliminary analysis of the effects of pair programming on job satisfaction. In: Proceedings of the Fourth International Conference on eXtreme Programming and Agile Processes in Software engineering (XP2002), pp. 212–215.

Williams, L., Kessler, R.R., 2000. The Effects of "Pair Pressure" and "Pair-Learning" on software engineering education. In: Proceedings of the 13th Conference on Software Engineering and Education (CSEET 2000). IEEE CS Press, pp. 59–65.

Williams, L., Kessler, R.R., 2003. Experimenting with industry's pair programming model in the computer science classroom. Journal of Software Engineering Education 11 (1), 7–20.

Williams, L., Kessler, R.R., Cunningham, W., Jeffries, R., 2000. Strengthening the case for pair programming. IEEE Software 17 (4), 19–25.

Wilson, J., Hoskin, N., Nosek, J., 1993. The benefits of collaboration for student programmers. In: Proceedings 24th SIGCSE Technical Symposium on Computer Science Education, pp. 160–164.

Wohlin, C., Runeson, P., Höst, M., Ohlson, M., Regnell, B., Wesslén, A., 2000. Experimentation in Software Engineering: An Introduction. Kluwer Academic Publishers.

Xu, S., Rajlich, V., 2005. Pair programming in graduate software engineering course projects. In: Proceedings of the 35th ASEE/IEEE Frontiers in Education Conference, Indianapolis (Indiana), October 19–22, F1G-7-F1G-12.

Gerardo Canfora is a professor of computer science at the Faculty of Engineering and also the Director of the Research Centre on Software Technology (RCOST) of the University of Sannio in Benevento, Italy. He has a degree in Electronic Engineering from the Federico II University of Naples. His research interests include software maintenance and evolution, service oriented computing, metrics, and experimental software engineering. He is a member of the Editorial Board of the Journal of Software Maintenance and Evolution: Research and Practice and serves as a General Chairperson for WCRE'06 and a Program Co-Chair for ICSM'06. He is a member of the IEEE Computer Society. He can be contacted at RCOST – Research Centre on Software Technology, Viale Traiano 1, 82100 Benevento, Italy; gerardo.canfora@unisannio.it.

Aniello Cimitile is the Dean of the University of Sannio in Benevento, Italy, where he is a professor of computer science. Previously, he was part of the Department of "Informatica e Sistemistica" at the Federico II University of Naples. He has a degree in electronic engineering from the Federico II University of Naples ". He has been a researcher in the field of software engineering since 1973 and his list of publications contains more than 100 papers published in journals and conference proceedings. He serves on the program and organizing committees of several international conferences and on the editorial and review committees of several international scientific journals in the fields of software engineering and software maintenance. He is a co-editor-in-chief of the Journal of Software Maintenance and Evolution: Research and Practice. His research interests include software maintenance and testing, software quality, reverse engineering, and empirical software engineering. He is a member of the IEEE and the IEEE Computer Society. He can be contacted at RCOST – Research Centre on Software Technology, Viale Traiano 1, 82100 Benevento, Italy; cimitile@unisannio.it.

Félix García is a lecturer at the University of Castilla-La Mancha (UCLM). His research interests include business process management, software processes, software measurement and agile methods. He has a M.Sc. degree and a Ph.D. degree in Computer Science from the UCLM, and is a member of the Alarcos Research Group of that University, which specialises in Information Systems, Databases and Software Engineering. He can be contacted at Escuela Superior de Informática, Paseo de la Universidad 4, 13071 Ciudad Real, Spain; Felix.Garcia@uclm.es.

Mario Piattini is a professor at the UCLM. His research interests include software quality, metrics and maintenance. He has a Ph.D. degree in Computer Science from the Technical University of Madrid, and leads the Alarcos Research Group. He is CISA and CISM by ISACA. He leads the Joint SOLUZIONA-UCLM Software Research and Development Center. He is member of ACM and the IEEE Computer Society. He can be contacted at Escuela Superior de Informática, Paseo de la Universidad 4, 13071 Ciudad Real, Spain; Mario.Piattini@uclm.es.

Corrado Aaron Visaggio is a lecturer at the University of Sannio. His main research interests are empirical software engineering, agile methods, software process modelling and management, and knowledge management applied to software engineering. He has a Ph.D. degree from the University of Sannio and he graduated in Electronic Engineering at the Politecnico of Bari, Italy, in 2001. He developed his master thesis at the Fraunhofer IESE, Kaiserslautern, Germany, in the field of Software Process Modelling. He is a member of the Research Centre on Software Technology (RCOST) of the University of Sannio in Benevento, Italy. He can be contacted at RCOST – Research Centre on Software Technology, Viale Traiano 1, 82100 Benevento, Italy; visaggio@unisannio.it.