JOURNAL OF SOFTWARE MAINTENANCE AND EVOLUTION: RESEARCH AND PRACTICE
J. Softw. Maint. Evol.: Res. Pract. 2005; **17**:401–423
Published online 1 November 2005 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/smr.322

Research

Pair designing as practice for enforcing and diffusing design knowledge

Emilio Bellini[1], Gerardo Canfora[1,*,†], Félix García[2], Mario Piattini[2] and Corrado Aaron Visaggio[1]

[1]*RCOST-Research Centre on Software Technology, University of Sannio, Palazzo ex Poste, Viale Traiano, 82100 Benevento, Italy*
[2]*Alarcos Research Group, University of Castilla-La-Mancha, Paseo de la Universidad, 4, 13071 Ciudad Real, Spain*

SUMMARY

Evolving software's design requires that the members of the team acquire a deep and complete knowledge of the domain, the architectural components, and their integration. Such information is scarcely addressed within the design documentation and it is not trivial to derive it. A strategy for enforcing the consciousness of such hidden aspects of software's design is needed. One of the expected benefits of pair programming is fostering (tacit) knowledge building between the components of the pair and fastening its diffusion within the project's team. We have applied the paradigm of pair programming to the design phase and we have named it 'pair designing'. We have realized an experiment and a replica in order to understand if pair designing can be used as an effective means for diffusing and enforcing the design knowledge while evolving the system's design. The results suggest that pair designing could be a suitable means to disseminate and enforce design knowledge. Copyright © 2005 John Wiley & Sons, Ltd.

KEY WORDS: pair programming; pair design; experimental software engineering; design evolution; knowledge management

1. INTRODUCTION

Documentation is the major source of information to successfully accomplish software evolution tasks. Unfortunately, in certain cases reading the documentation is not enough to understand all of the aspects of a software system. The evolution of software design can particularly benefit from a

*Correspondence to: Gerardo Canfora, RCOST-Research Centre on Software Technology, University of Sannio, Palazzo ex Poste, Viale Traiano, 82100 Benevento, Italy.
†E-mail: canfora@unisannio.it

Received 20 December 2004
Revised 29 April 2005
Accepted 8 June 2005

practice that fosters and fastens the knowledge transfer among the team's members, for several reasons as follows.

- Software evolution, as well as many other software engineering activities, is a cooperative effort [1] and the executed process is not linear: changing the design as the first activity, then documenting changes, writing the code, and finally regression testing the system. The team usually follows a form of interleaved process: making decisions about small changes, codifying and checking whether the changes make sense and work, confirming the design, documenting the modifications, and completing and testing the implementation. The decisions made are the result of a collective activity of trials and tests; as a consequence, the path of reasoning that led to the decision is not formalized and remains embedded in the brains of the people who decided and approved the modifications.
- The turnover of maintenance personnel determines a serious loss of experiences and knowledge that is difficult to replace; it is more efficient and effective to ask people who know the system to explain the tips and the traps of the architecture, rather than to attempt to extract them from (huge and often out-of-date) documentation.
- Software's design requires the capability to deal with different levels of abstractions: implementation, database, business logic, presentation, deployment, interaction with other systems, and communication protocols. Mastering all of these aspects and their integration is not easy and, usually, the documentation does not address them explicitly.
- Documentation has a low communication bandwidth: acquiring information through documentation requires time and in some situations time is a very scarce resource. 'Face-to-face channels offer the prospect of richer communication because of the ability to transmit multiple cues[. . .]important when there are high levels of equivocality (ambiguity) and uncertainty' [2].

A strategy for diffusing the knowledge built by the individual engineers during their daily work and consolidating it at the team or corporate level could be helpful; such a strategy is not intended as an alternative to documentation, but as a means to complement and to improve the use of documents as a source of information by fostering the 'sense-making' process.

Pair programming is an agile practice [3], consisting of two developers working on the same code on the same machine: one develops the code, the other reviews it. Among the number of benefits derived from the adoption of this practice enumerated in literature, *knowledge transfer* [4] is worth our attention. The continuous collaborative work of the pair's components fosters the flow of the knowledge that is difficult to transmit with documentation, such as strategies, rationales, and techniques.

We applied pair programming practice to the design phase and named it 'pair designing'. Pair designing means that two designers work on the same design's document on the same machine and at the same time: the first designer named '*driver*', actively writes the document and the other, named '*observer*', reviews it. The two roles can be switched during the work as needed: this usually happens when the driver does not know how to proceed, whereas the observer knows the way to solve the problem. The observer can also accomplish different activities from reviews which are helpful for reaching the goal of the current task. Our conjecture is that pair designing can help a team's members to quickly obtain complete and correct knowledge of the software.

This paper aims at investigating whether the pair designing can be used as a means for enforcing and diffusing the design's knowledge among the members of the project's team.

Diffusing knowledge concerns the dissemination of knowledge among the team's members. Thus, it is essentially a process of sharing knowledge by socialization. This may be useful at the initial phases of the process when the team is required to reach a certain level of knowledge about the system in order to properly accomplish maintenance tasks.

Enforcing knowledge means increasing the overall knowledge of team's members by combining individual pieces of knowledge. It may be useful when the team is already familiar with the system, but some specific features and characteristics are still unclear. The enforcement of knowledge is required after having diffused the knowledge; it is intended as a means for reducing reworks and latency times due to the limited knowledge of the design of software.

We define the 'design's knowledge' as the body of information concerning the architecture of a software system, the business rules describing the domain in which the system operates, and the rationales that the architecture is based on.

We have realized an experiment and a replica with the aim of answering the following research questions.

(1) Is pair designing effective for enforcing the design's knowledge of the project team's members during design evolution tasks?
(2) Is pair designing effective for diffusing the design's knowledge of the project team's members during design evolution tasks?

In this paper we discuss the outcomes of the experiment and the replica addressing the concerns listed above. The paper proceeds as follows: Section 2 discusses the related work; Section 3 shows the experiment's and the replica's design; Section 4 analyses the outcomes; Section 5 discusses the validity of the experiments' outcomes; and, finally, Section 6 draws the conclusions.

2. RELATED WORK

The attention of researchers has initially focussed mainly on the quality and productivity of pair programming [5–7]. Recently, the target of pair programming investigation has turned to learning and knowledge transfer [8–10]. Williams and Kessler [11] found that pair programming fosters knowledge leveraging between the two programmers, particularly tacit knowledge.

In [8], the authors investigated, through an experiment, which knowledge needs to be addressed in order to effectively implement pair programming when the pair's components are distributed. Williams and Upchurch [12] examined the ways pair programming may enhance teaching and learning in computer science education. Students were able to complete programming assignments faster, with higher quality, and appeared to learn faster. McDowell *et al.* [4] investigated the effects of pair programming on student performance in an introductory programming class. The results show that students who worked in pairs produce better programs. The research reported in [13] concluded that students perceived pair programming as valuable to their learning.

Some authors emphasize the need for establishing a common culture of software design and the spread of knowledge about the architecture among the members of a software project. In [14] the authors proposed an approach based on Design Decision Trees in order to: consciously consider the choices they make; explicitly specify their goals and assumptions; and consider and specify alternative and solutions. Another debated issue is the linkage between the domain's knowledge and the software's components. A possible solution for this problem is provided by Li *et al.* [15] and it is based on representing domain knowledge with simplified semantic networks. Using visualization tools is a direction that many researchers are taking: Jiawei *et al.* [3] described a method to visualize the requirement's specification with the aim of supporting the reuse of design knowledge.

Organizational literature in the last 20 years has offered several relevant contributions about the social dimension of tacit/explicit knowledge conversion processes [16], the different nature of information and of knowledge [17], and the relevance of contexts affecting the knowing process, such as mediated, situated, provisional, pragmatic, and contested process [18]. The continuous discussion between pair's members allows the knowledge to flow from one partner to the other naturally, enhancing the sense making of explicit knowledge formalized in a number of tools (handbooks, procedures' descriptions, and many others) and the sharing of tacit knowledge embedded in organizational routines.

The current work is part of a research plan, aiming at evaluating the relationship between the practice of pair designing and the knowledge building about the 'big picture' of the system. A preliminary experiment provided conclusions which led the authors to afford more systematic research, addressed with this paper: the results were discussed in [19]. Two main outcomes were obtained. First, all of the experimental subjects who worked in pairs showed greater knowledge with respect to those who worked as singletons. Second, the process of knowledge building was more stable for pairs than for singletons: the knowledge growth of pairs can be predictable and repeatable within certain limits. The experiment discussed in [20] investigated how the educational background of pair's components can affect the effectiveness of the practice if used for leveraging knowledge. We found confirmations that forming pairs with individuals with the same educational background emphasizes the expected benefits of pair designing.

3. THE EXPERIMENTS

This section illustrates the experiments and discusses the outcomes.

3.1. Definitions

The experiments were executed with the purpose of testing the following null hypotheses:

$\mathbf{H_{0a}}$: the pair designing does not affect the diffusion of design knowledge when performing evolution tasks;

$\mathbf{H_{0b}}$: the pair designing does not affect the enforcement of design knowledge when performing evolution tasks.

The alternative hypotheses are:

$\mathbf{H_{1a}}$: the pair designing affects the diffusion of design knowledge when performing evolution tasks;

$\mathbf{H_{1b}}$: the pair designing affects the enforcement of design knowledge when performing evolution tasks.

3.2. The first experiment characterization

Subjects

The experiment was executed with the collaboration of the students of the Master[‡] of Technologies of Software (MUTS) and Master of Management and Technologies of Software (MUTEGS), higher education university courses for post-graduate students, at University of Sannio (http://www.ing.unisannio.it/master/). Students of MUTS hold a scientific degree (engineering, mathematics, physics), whereas students of MUTEGS hold an economic/humanistic degree (economics, philology, literature, philosophy). Both courses provide the same basic education in computer engineering (operating systems, programming languages, network, database, and software engineering), but MUTS students are educated for developing and maintaining software systems, whereas MUTEGS students deal with the economic and organizational issues of software lifecycles. The two Master courses are held contemporarily and both last one year, during which students attend theoretical classes and laboratory sessions, with the same professors and lecturers, develop a large and complex project in connection with an enterprise, participate in seminaries from international experts, and perform a three-month stage in software companies.

The subjects were organized as follows:

- five pairs with two MUTS students;
- five pairs with two MUTEGS students;
- the other 16 subjects (MUTS and MUTEGS) worked as solo designers.

All of the groups were formed randomly.

Questionnaires

We prepared two questionnaires, QA and QB (see Appendix A), in order to measure the dependent variables. Both questionnaires were distributed as entry and exit questionnaires, so that each subject randomly had QA (or QB) at entry and, conversely, QB (or QA) at exit. This avoided any dependency of the results on the questionnaire itself. The questions concerned architectural and functional aspects of the system.

Variables

There were two dependent variables: the knowledge diffusion and the knowledge enforcement. The subjects studied the design of the system before starting the assignment in order to acquire an initial knowledge of the system. This level of knowledge was measured with the entry questionnaires. The subjects answered the exit questionnaires after having performed evolution tasks on the system. The *knowledge diffusion* was measured by calculating the grades of the exit questionnaires. The *knowledge enforcement* was calculated as the difference between the exit questionnaire's grade

[‡]The Italian educational system defines 'Master' as a post-graduation course aimed at specializing students in certain topics.

and the entry questionnaire's grade. The questionnaires were evaluated in this way: each correct answer was evaluated 1; each incorrect answer was evaluated 0.

Rationale for the sampling of the population

Students of software engineering courses are suitable for such an experiment because they study software architecture and software system design. Furthermore, they are usually employed as software architects or designers after graduation. MUTS and MUTEGS students are a fine population's sample, considered that they experienced actual project work during the Master's course, established together with the enterprises funding the courses. Since the students have comparable curricula, there is no relevant bias in the samples of pairs and solos; however, the statistical tests to ascertain that the randomization of pair's samples and solos' samples was realized are discussed in advance.

Assignment

In order to evaluate the knowledge built while evolving the system design, the assignment for the subjects consisted of improving the design of a system. The design of the system was formalized in UML and included textual specification of the system's requirements, two use cases diagrams, and two class diagrams (for a total of 15 classes). An excerpt of the design documentation is shown in Appendix A. Experimenters developed the design. Considered the time available, we preferred to avoid bulky documentation.

There were basically two maintenance tasks:

- *reduce complexity*, by erasing entities or relationships between entities not fundamental for understanding;
- *improve readability*, by changing existing entities (use cases, actors, classes, methods) or adding new ones.

This kind of assignment was targeted at maximizing the knowledge built by doing; as a matter of fact, evolving software systems needs the programmer to analyse the system in depth. The system design was realized by taking into account the knowledge of subjects, with the aim of making the objects representative of the population.

The process

The process of the experimental run was the following:

- each subject studied documentation for 30 minutes, individually;
- each subject answered an entry questionnaire, individually, for about 15 minutes. The entry questionnaire was aimed at establishing the baseline, i.e., the level of knowledge that the subjects built by reading the documentation before executing the tasks;
- the pairs and the solo designers performed the maintenance tasks for 2 hours;
- each subject answered an exit questionnaire individually, in order to understand the knowledge built by modifying the system according to two different styles, pair and solo.

Table I. Design of the first experiment.

Subjects	Treatment	Input	Output

Five MUTEGS
Five MUTEGS
	Paired MUTEGS
	MUTEGS
		Requirement specification;
		Use case diagram;
		Class diagram;
		Entry questionnaire QA (or QB);
		Exit questionnaire QB (or QA).
			Modifications to use case diagram and class diagram;
			Answered entry questionnaire QA (or QB);
			Answered exit questionnaire QB (or QA).

Five MUTS	Paired MUTS		
Five MUTS	MUTS		
Eight MUTS	Solo		
Eight MUTEGS	Solo		

Before running the experiments the subjects participated laboratory sessions for training them in pair designing.

Although we would have liked to use a CASE tool, such as Rational Rose [21] or ArgoUML [22], we finally decided to use only pen and paper. The reason was that some subjects could be more familiar with these kinds of tools and this could inject bias into the results. As a consequence, we would have needed more time for preparation in order to equalize the ability of subjects to work with tools. Appendix A shows an excerpt of the experimental material. In Table I the experimental design for the first experiment is provided.

3.3. The replica in Spain

The subjects were students enrolled at the Department of Computer Science at the University of Castilla–La Mancha in Spain. The first group was composed of 42 students enrolled in the final-year (third) of the Computer Science (BSc) in the specialization of Management (hereafter referred as *3BScMngmt*); the second group was composed of 39 students enrolled in the final-year (third) of the Computer Science (BSc) in the specialization of Systems (*3BScSys* in the following); and finally the third group consisted of 12 students enrolled in the final-year of the fifth year of MSc (named *5MSc* in the following). The replica had a different experimental design (Table II) with respect to the previous experiment: 32 randomized pairs where formed and 32 subjects were left working as solo designers. Table II shows the experimental design for the replica.

Dependent and independent variables, the process, the assignment, and the questionnaires remained the same, but the overall experiment lasted 2 hours and the experimental package was properly translated in Spanish by the native Spanish language speaking authors.

Table II. Experimental design of the replica.

Subjects	Treatment	Input	Output

64 students
3BScMngmt
3BscSys
5MSc
	Paired
	3BScMngmt–
	3BScMngmt
	3BscSys–
	3BscSys
	5MSc–5MSc
		Requirement specification;
		Use case diagram;
		Class diagram;
		Entry questionnaire QA (or QB);
		Exit questionnaire QB (or QA).
			Modifications to use case diagram and class diagram;
			Answered entry questionnaire QA (or QB);
			Answered exit questionnaire QB (or QA).

32 students 3BScMngmt 3BscSys 5MSc	Solo		

4. ANALYSIS OF DATA

In order to accept the outcomes of the experiments as valid, it is necessary to make sure that there are no relevant differences in the samples to compare: the samples of solos and pairs have to be equivalent for each experiment. If some differences on the entry questionnaires are detected, the randomization was not accomplished correctly. Table III shows this analysis for the experiment and for the replica. Mann–Whitney's method was used in all the tests because the data of samples were not normally distributed and the p-level threshold value was fixed at 5%.

The tests in Table III show that the MUTS subjects working as solos and those working in the pairs did not present significant differences at the entry questionnaire; similarly, the MUTEGS subjects of the solos' set and those of the pairs' set did not present significant differences. Similar considerations can be made for the randomization in the samples of the replica: also in this case there is no significant differences between the subjects performing solo and pair designing. It is possible to conclude that the randomization was realized correctly.

4.1. The knowledge diffusion

In Table IV the results of statistical tests for rejecting the null hypothesis H_{0a} are reported. We used the Mann–Whitney test with p-level at 0.05.

The null hypothesis can be rejected in the MUTS sample case, but it cannot be rejected in the MUTEGS sample case. This is due to the different skills of MUTS and MUTEGS students. MUTS students come from scientific studies whereas MUTEGS students come from other kinds of studies. MUTS students are supposed to be more familiar with algorithms and deductive reasoning than MUTEGS students. Our conjecture is that the effectiveness of the practice could be affected by the ability of subjects to perform pair design. As a matter of fact, there is statistical evidence that the performance of the two populations in terms of knowledge is different, as the third row in the

Table III. Tests for validating the randomization of sample.

Test between	Rank sum α	Rank sum β	p-level	Experiment
Entry Questionnaires of Subjects of MUTS Pairs sample (α) Subjects of MUTS Solos sample (β)	171.0000	39.0000	0.214 768	Italian experiment
Entry Questionnaires of Subjects of MUTEGS Pairs sample (α) Subjects of MUTEGS Solos sample (β)	112.0000	59.0000	0.130 919	
Entry Questionnaires of Solos of the 3BScSys sample (α) Pairs of the 3BScSys sample (β)	425.0000	395.0000	0.214 741	Spanish experiment
Entry Questionnaires of Solos of the 5MSc sample (α) Pairs of the 5MSc sample (β)	31.5000	465.0000	0.229 767	
Entry Questionnaires of Solos of the 3BScMngmnt sample (α) Pairs of the 3BScMngmnt sample (β)	425.0000	395.0000	0.321 966	

Table IV. Tests of the knowledge diffusion hypotheses.

Test between	Rank sum α	Rank sum β	p-level	Experiment
MUTS pairs (α) MUTS solos (β)	116.5000	54.50	0.049	Italian experiment
MUTEGS pairs (α) MUTEGS solos (β)	78.50	57.50	0.270	
MUTS pairs (α) MUTEGS pairs (β)	135.0000	75.00	0.023	
Pairs 5MSc (α) Solos 5MSc (β)	51.5000	26.5000	0.030 912	Spanish experiment
Pairs 3BScSys (α) Solos 3BScSys (β)	253.0000	567.0000	0.000 17	
Pairs 3BScMngmnt (α) Solos 3BScMngmnt (β)	447.0000	778.0000	0.000 00	

table shows. From the first experiment we can conclude that: (i) pair design can diffuse knowledge better than solo programming; and (ii) the effectiveness of pair design could be affected by the type of population.

The replica realized in Spain produced empirical evidence that pair designing can help the diffusion of design's knowledge in all three samples. The replica reinforces the findings of the experiment run in Italy.

Table V. Descriptive statistics of the data.

Pairs	Standard deviation	Average	Maximum	Minimum	Experiment
MUTS pairs	1.75	5.8	9	4	Italian experiment
MUTEGS Pairs	1.60	3.9	7	1	
MUTS Solos	1.03	4.25	6.00	3.00	
MUTEGS Solos	1.55	5.13	7.00	3.00	
Pairs 3BScSys	1.02	6.00	7.00	3.00	Spanish Experiment
Solos 3BScSys	1.26	4.44	6.00	3.00	
Pairs 5MSc	0.98	6.17	7.00	5.00	
Solos 5MSc	0.82	5.33	7.00	5.00	
Pairs 3BScMngmnt	0.73	6.30	8.00	5.00	
Solos 3BScMngmnt	0.94	4.21	5.00	1.00	

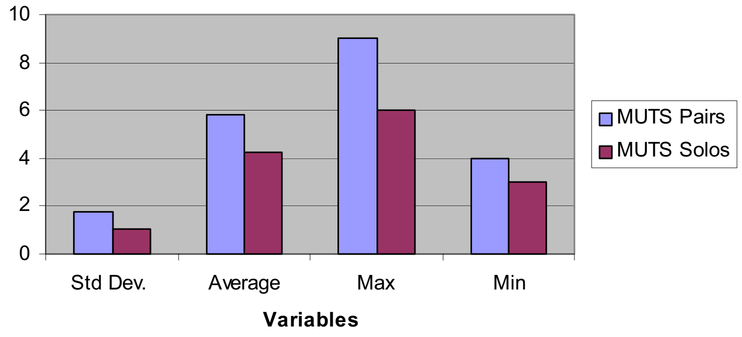

Figure 1. Descriptive statistics of MUTS samples.

Table V shows the descriptive statistics of data samples and Figures 1–5 compare the descriptive statistics of the five samples.

It appears that in all cases pairs outperformed solos in terms of knowledge built, as the values of average, maximum and minimum demonstrate. This does not happen with MUTEGS students: as previously mentioned, this could be due to the different skills MUTS and MUTEGS students have. As a matter of fact, there is empirical evidence that the MUTEGS population is different from the MUTS population.

The Spanish samples show that pairs constantly achieved higher values of knowledge diffusion than solos.

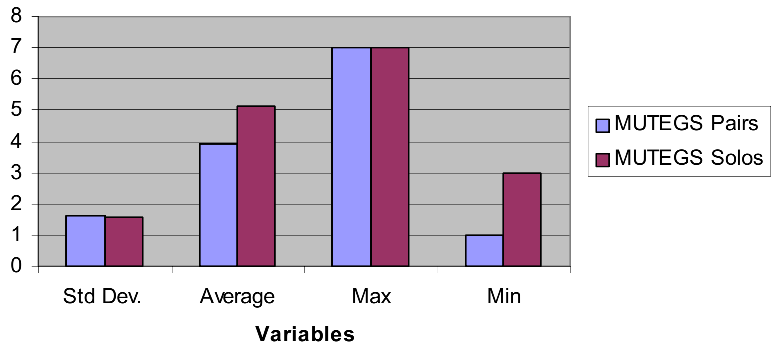

Figure 2. Descriptive statistics of MUTEGS samples.

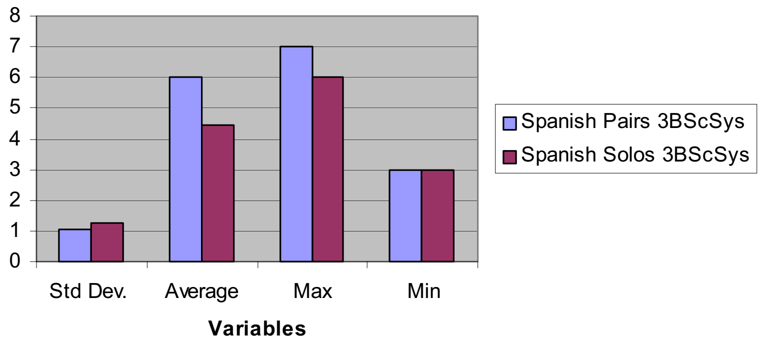

Figure 3. Descriptive statistics of 3BScSys samples.

4.2. The knowledge enforcement

Table VI reports the results of the tests of the H_{0b} hypothesis.

Table VI shows that pair design helps enforce the knowledge with statistical evidence for the MUTS sample, whereas the results of the MUTEGS subjects did not report empirical evidence. This confirms the conclusions of knowledge diffusion's analysis: the type of educational background affects the enforcement of knowledge obtained with pair designing. As a matter of fact, the third row demonstrates that the two samples obtained statistically different enforcement of knowledge after having performed pair design.

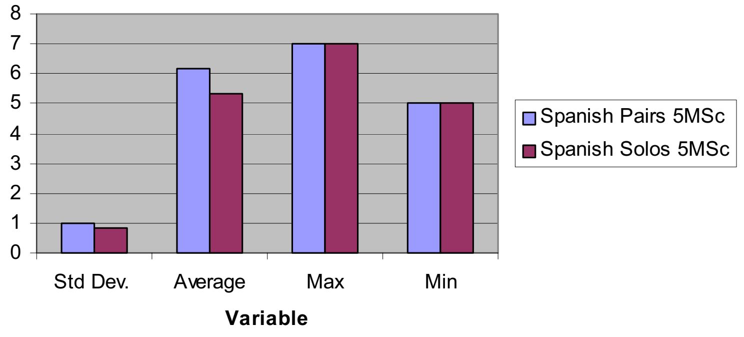

Figure 4. Descriptive statistics of 5MSc samples.

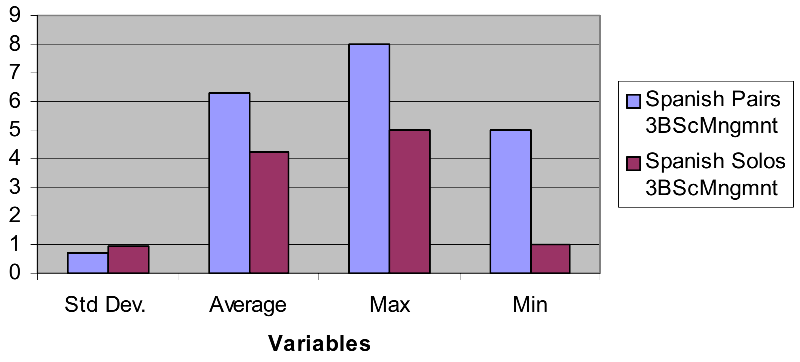

Figure 5. Descriptive statistics of 3BScMngmnt samples.

The replica basically confirms the findings of the experiment, because all of the rows show the p-level under the threshold of 0.05. Only the first row reports a greater value, but it is relatively close to 0.05. The descriptive statistic, reported in Tables VII and VIII, can suggest further observations.

As Figure 6 shows, the standard deviation of pairs is less than the standard deviation of solos for the MUTS sample, but this does not occur in the MUTEGS sample: this points out that the knowledge enforcement in the (MUTS) pairs is more predictable than in the solos. Pair designing can be used for planning the individual growth of team's members.

The effectiveness of the practice could depend on the educational background of pair's members, as already observed. Figure 7 shows the average and the maximum values of the experiment's sample: pairs outperformed solos.

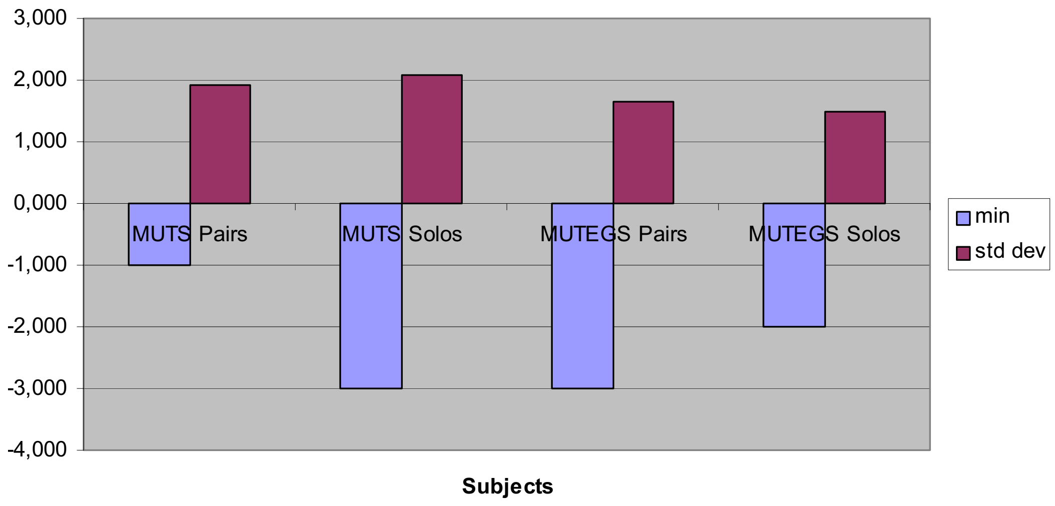

Figure 6. The standard deviation and the minimum values.

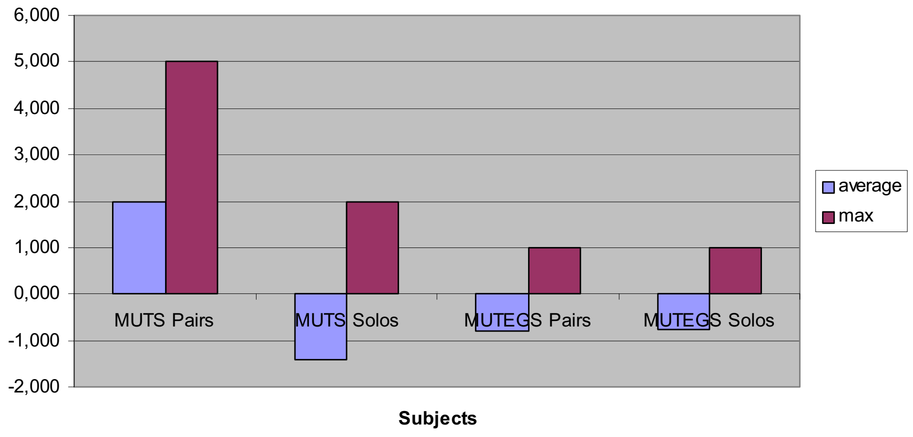

Figure 7. The average and the maximum values.

Table VI. Tests of the knowledge enforcement hypotheses.

Test between	Rank sum α	Rank sum β	p-level	Experiment
MUTS pairs (α) MUTS solos (β)	123.500	47.5000	0.0102	Italian experiment
MUTEGS pairs (α) MUTEGS solos (β)	53.500	66.5000	0.2164	
MUTS pairs (α) MUTEGS pairs (β)	110.500	42.5000	0.0428	
Spanish pairs 3BScSys (α) Spanish solos 3BScSys (β)	49.5000	28.5000	0.086 984	Spanish experiment
Spanish pairs 5MSc (α) Spanish solos 5MSc (β)	551.0000	269.0000	0.000 942	
Spanish pairs 3BScMngmnt (α) Spanish solos 3BScMngmnt (β)	51.5000	26.5000	0.042 337	

Table VII. Descriptive statistic of the experiment.

Statistical parameter	MUTS pairs	MUTS solos	MUTEGS pairs	MUTEGS solos
Average	2.000	−1.400	−0.800	−0.750
Maximum	5.000	2.000	1.000	1.000
Minimum	−1.000	−3.000	−3.000	−2.000
Standard deviation	1.915	2.074	1.643	1.500

Table VIII. Descriptive statistics of the replica.

Statistical parameter	5MSc pairs	5MSc solos	3BScSys pairs	3BScSys solos	3BScMngmnt pairs	3BScMngmnt solos
Average	1.167	−0.500	1.714	−0.579	1.111	−1.036
Maximum	3.000	3.000	4.000	3.000	3.000	2.000
Minimum	−1.000	−2.000	−1.000	−4.000	−1.000	−5.000
Standard deviation	1.722	1.871	1.736	1.865	1.278	1.856

The descriptive statistics of the replica in Spain are reported in Table VIII. The replica's results confirm the results of the experiment, as the graph in Figure 8 demonstrates: the pairs outperformed the solos in every sample involved in the replica.

Figure 9 shows that the standard deviations of the pairs of each sample are smaller than the corresponding value of the solo's sample: this suggests that the enforcement in the pairs is more predictable than in the solos.

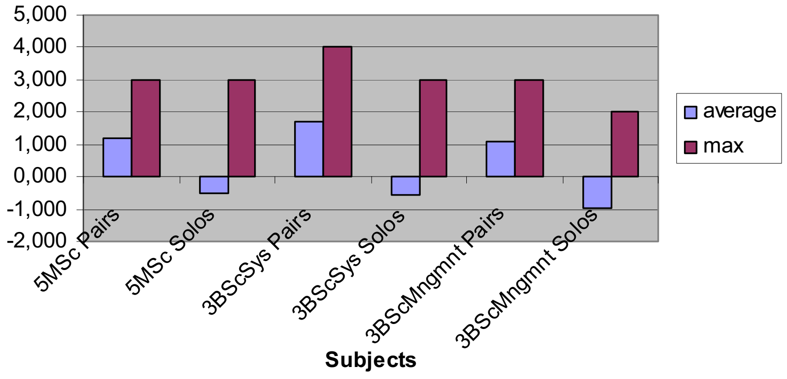

Figure 8. The average and the maximum values in the replica.

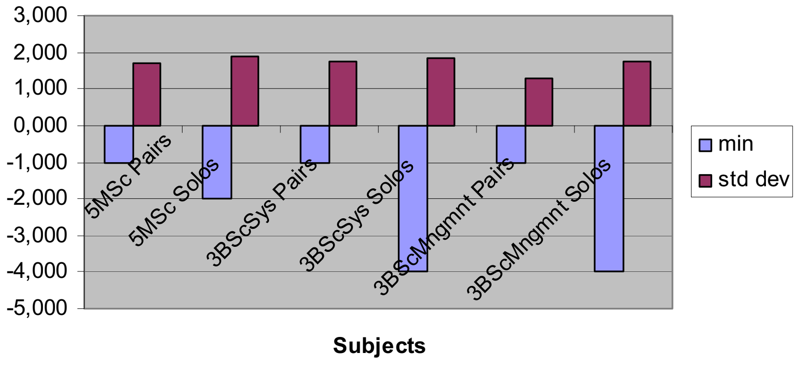

Figure 9. The minimum and the standard deviation.

4.3. Comparing knowledge diffusion and enforcement

The results for the diffusion and the enforcement report similar conclusions:

- pair designing is helpful both for diffusing the knowledge within the project team when a large number of team's members are not familiar with the software design and for enforcing the knowledge of each designer when they have built a preliminary idea of the software;
- the skills and individual abilities of team members could seriously affect the effectiveness of the practice; this entails that if the project manager plans to use pair designing for enforcing the design knowledge, an assessment of the team members is required; and
- the enforcement of knowledge is more predictable when using pair designing than the enforcement due to traditional designing.

5. EXPERIMENTAL THREATS

Threats to construct validity

The dependent variables aim at capturing the knowledge. We proposed questionnaire grading that surely cannot capture the overall aspects of the object to be measured. Tacit knowledge for its intrinsic nature is hard to formally describe and quantify. We consider what we measure as an approximation of what we intend to measure.

Threats to internal validity

The following issues have been dealt with.

- *Differences among subjects*. Using a within-subjects design, error variance due to differences among subjects is reduced. In this experiment, students had a degree with experience in using UML. It is one of the main topics of their curriculum.
- *Learning effects*. The subjects were required to deal with only one run with only one assignment, so the learning threat was cancelled.
- *Fatigue effects*. On average the experiment lasted a time short enough that fatigue was not relevant. As a confirmation, the students asked for a longer time to accomplish better the assignment.
- *Persistence effects*. In order to avoid persistence effects, the experiment was run with subjects who had never performed a similar experiment.
- *Subject motivation*. The participants were volunteers, in order to help us in our research. We motivated students to participate in the experiment, explaining to them that they were learning a practice that should be useful in their professional career.
- *The experimental package*. Both in the experiment and in the replica the results of the run were independent from the experimental package, as shown in Table IX. We performed a Mann–Whitney test with the p-level fixed at 0.05, and there is no evidence that the differences due to the questionnaires were statistically significant.

Table IX. Test on the questionnaires.

Test between	Rank sum α	Rank sum β	p-level
Questionnaire A (α) Questionnaire B (β) in the experiment	540.0000	406.0000	0.161
Questionnaire A (α) Questionnaire B (β) in the replica	598.0000	677.0000	0.2068

Threats to external validity

Two threats to validity have been identified which limit the possibility of applying generalization.

- *Materials and tasks used.* In the experiment we have used the system design's documentation prepared by experimenters. The system showed a discrete degree of complexity, because it describes an existing system.
- *Subjects*. Students play a very important role in the experimentation in software engineering, as pointed out in [23,24]. In situations in which the tasks to perform do not require industrial experience, experimentation with students is viable [23,25].

6. CONCLUSIONS

Evolving software's design requires that all of the members of the team reach a deep and complete knowledge of the domain, the architectural components, and their integration. A weak knowledge of these issues can lead to wrong choices and frequent reworks during evolution tasks on the software design. One of the expected benefits of pair programming is the enforcement of the software's knowledge within the project's team. We applied the paradigm of pair programming to the design phase and named it 'pair designing'. A preliminary experiment demonstrated that pair designing can be helpful for enforcing the knowledge of the software within the project's team.

We realized an experiment and a replica with the aim of gathering empirical evidence on the effectiveness of pair designing for diffusing and enforcing the design knowledge within the project team.

The experiments were realized in two universities: the first was realized at the University of Sannio in Italy and the second was realized in the University of Castilla–La Mancha in Spain.

The following conclusions can be drawn.

- There is empirical evidence that, in the given context, pair designing may help enforce the design knowledge when evolving systems. Moreover, the practice offers a good level of predictability.

This could be a success factor for evolving projects: it can decrease the number of reworks for specific tasks and can ease the understanding of maintenance requirements.

- Working at a task (in this case maintenance of the software's design) does not guarantee that the designer enforces its knowledge significantly. The standard deviations of solos' sample, in the experiment was higher than that of pairs' components. This entails that a strategy to make the designer learn by carrying out their work is necessary, and using the pair designing could be a suitable candidate.

These conclusions are strongly related to the context in which the experiment was executed.

- The subjects were graduate and post-graduate students; it is necessary to run similar experiments with professionals in order to verify if the outcomes also remain valid for practitioners.
- The assignments of the experiment concerned a software system with a lower complexity than that usually shown by industrial projects: this was necessary in order to perform the experiment in the planned time. The application of pair designing to larger projects requires further investigation.
- The type of evolution may influence the effectiveness of pair designing in knowledge enforcement and diffusion. Such aspects will be addressed with future research.

As future work we plan to execute further experiments in order to enforce the external validity of the outcomes obtained with this experimentation. There are two main issues to address.

- To overcome the limitations of this research due to the specific context in which the experiments were run, concerning the subjects involved, the projects, and the kind of evolution tasks.
- To enlarge the number of subjects forming the sample under analysis. The samples examined for this experimentation are not large enough for ensuring a strong external validity. Further replications of the experiments may be useful.

In addition, we plan to study other aspects of the practice, as follows.

- To understand how to form the pairs in order to maximize the benefits in terms of knowledge enforcement and diffusion. Working in pairs could be seriously affected by the way pair's members achieve cooperation. Some variables can affect the success of the practice, some of them are related to psychological aspects, others to background aspects, and so forth. Research is needed to identify these factors and a means to handle them.
- The costs of the practice with regards to the benefits: pair designing means paying two people to work on the same document. In some situations it can pay off: the reworking of two people on two different documents can be more expensive than to pay two people to work on the same document but with a few reworks. This concern deserves deeper investigation.

APPENDIX A

Here we give some of the design documentation (Figure A1, Table AI) and the Questionnaire QB (Figure A2).

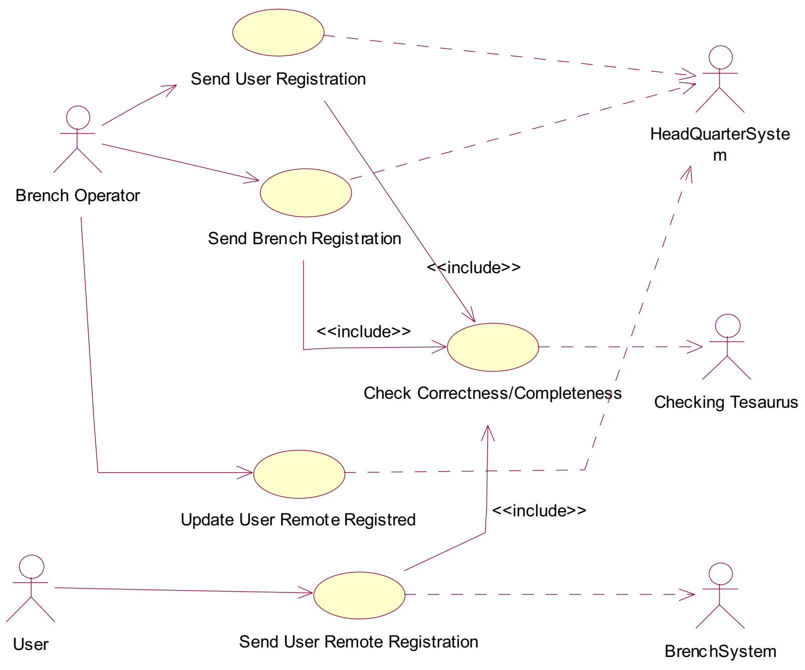

Figure A1. An excerpt of a use case.

Table AI. An excerpt of a use case 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	No use cases.
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 are stored in the database of the local branch.
Related expectations	Database management system. Correctness and completeness checks. Data sending to the headquarters.
Related requirements/use cases	Check correctness/completeness.

1. Could Remote Registration of User (User Remote Registration Sending) extend local user registration (User registration Sending)?		
a. Yes	b. No and it does not make sense	c. Possible, with proper modifications
2. Does the updating of user data (User Remote Registered Updating) require the correctness and completeness check (Correctness/Completeness Checker)?		
a. Yes	b. No and it does not make sense	c. Possible, with proper modifications
3. Could use cases Notification of Transaction To Buyer and Notification of Transaction To Branch be merged in one use case?		
a. Yes	b. No and it does not make sense	c. Possible, with proper modifications
4. Could the use case Update Database extends the use case Local Book Search?		
a. Yes	b. No and it does not make sense	c. Possible, with proper modifications
5. Given a transaction, can information concerning vendor be obtained through the Book (object)?		
a. Yes	b. No and it does not make sense	c. Possible, with proper modifications
6. A (object) Branch Operator must have executed at least one operation, otherwise it does not exist in the System.		
a. True	b. False	c. This information is not provided by documentation.
7. It is possible to obtain the list of registered users in a local branch through Data contained in a Branch (object).		
a. True	b. False	c. This information is not provided by documentation.
8. DataHandler (object) helps query the Database.		
a. True	b. False	c. This information is not provided by documentation.
9. Checker (object) verifies if all the fields of the form are filled in.		
a. True	b. False	c. This information is not provided by documentation.
10. The user interface is provided only for the remote part of the system.		
a. True	b. False	c. This information is not provided by documentation.

Figure A2. Questionnaire QB.

REFERENCES

1. Bentley R, Rodden T, Sawyer P, Sommerville I. An architecture for tailoring cooperative multi-user displays. *Procedings ACM Conference on Computer-Supported Cooperative Work (CSCW 1992)*. ACM Press: New York NY, 1992; 187–194.
2. Melnik G, Maurer F. Direct verbal communication as a catalyst of agile knowledge sharing. *Proceedings IEEE Agile Development Conference (ADC 2004)*. IEEE Computer Society Press: Los Alamitos CA, 2004; 21–31.
3. Jiawei H, Bailey A, Sutcliffe A. Visualisation design knowledge reuse. *Proceedings 4th IEEE International Conference on Information Visualization*, 2004. IEEE Computer Society Press: Los Alamitos CA, 2004; 745–751.
4. McDowell C, Werner L, Bullock H, Fernald J. The effects of pair programming on performance in an introductory programming course. *Proceedings ACM Technical Symposium on Computer Science Education (SIGCSE 2002)*. ACM Press: New York NY, 2002; 38–42.
5. Abrahamsson P, Koskela J. Extreme programming: A survey of empirical data from a controlled case study. *Proceedings IEEE International Symposium on Empirical Software Engineering (ISESE 2004)*. IEEE Computer Society Press: Los Alamitos CA, 2004; 73–82.
6. Nosek JT. The case for collaborative programming. *Communication of ACM* 1998; **41**(3):105–108.
7. Williams L, Cunningham W, Jeffries R, Kessler RR. Strengthening the case for pair programming. *IEEE Software* 2000; **17**(4):19–25.
8. Canfora G, Cimitile A, Visaggio CA. Lessons learned about distributed pair programming: What are the knowledge needs to address? *Proceedings IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2003)*. IEEE Computer Society Press: Los Alamitos CA, 2003; 314–319.
9. Williams L, Krebs W, Layman L, Antón A. Toward a framework for evaluating extreme programming. *Empirical Assessment in Software Engineering (EASE 2004)*, 2004; 11–20. Available at: http://ease.cs.keele.ac.uk/ease2004/accepted.htm [31 May 2005].
10. Williams L, McDowell C, Fernald J, Werner L, Nagappan N. Building pair programming knowledge through a family of experiments. *Proceedings IEEE International Symposium on Empirical Software Engineering (ISESE 2003)*. IEEE Computer Society Press: Los Alamitos CA, 2003; 143–152.
11. Williams L, Kessler B. The effects of 'Pair-Pressure' and 'Pair-Learning'. *Proceedings Conference on Software Engineering Education and Training (CSEE&T 2000)*. IEEE Computer Society Press: Los Alamitos CA, 2000; 59–65.
12. Williams L, Upchurch RL. In support of student pair-programming. *Proceedings ACM Technical Symposium on Computer Science Education (SIGCSE 2001)*. ACM Press: New York NY, 2001; 327–331.
13. Van DerGrift T. Coupling pair programming and writing: Learning about students' perceptions and processes. *Proceedings ACM Technical Symposium on Computer Science Education (SIGCSE 2004)*. ACM Press: New York NY, 2004; 2–6.
14. Ran A, Kuusela J. Design decision trees. *Proceedings IEEE International Workshop on Software Specification and Design (IWSSD 1996)*. IEEE Computer Society Press: Los Alamitos CA, 1996; 172–175.
15. Li Y, Yang H, Chu W. Generating linkage between source code and evolvable domain knowledge for the ease of software evolution. *Proceedings IEEE International Symposium on Principles of Software Evolution (ISPSE 2000)*. IEEE Computer Society Press: Los Alamitos CA, 2000; 196–205.
16. Nonaka I. A dynamic theory of organizational knowledge creation. *Organization Science* 1994; **15**(5):14–37.
17. Choo CW. *The Knowing Organization*. Oxford University Press: Oxford, 1998.
18. Blackler F. Knowledge, knowledge work and organizations: An overview and interpretation. *Organization Studies* 1995; **16**(6):1021–1046.
19. Canfora G, Cimitile A, Visaggio CA. Working in pairs as a means for design knowledge building: An empirical study. *Proceedings IEEE International Workshop on Program Comprehension (IWPC 2004)*. IEEE Computer Society Press: Los Alamitos CA, 2004; 62–69.
20. Canfora G, Cimitile A, Garcia F, Piattini M, Visaggio CA. Confirming the influence on educational background in pair-design knowledge through experiments. *Proceedings Annual ACM Symposium on Applied Computing (SAC 2005)*. ACM Press: New York NY, 2005; 1478–1485.
21. Cernosek G. Next generation model-driven development. http://www3.software.ibm.com/ibmdl/pub/software/rational/web/whitepapers/rsa-cernosek-wp.pdf [8 July 2005].
22. Ramirez A, Vanpeperstraete P, Rueckert A, Odutola K, Bennett J, Tolke L, van der Wulp M. User Manual. http://argouml.tigris.org/documentation/defaulthtml/manual/ [8 July 2005].
23. Basili VR, Lanubile F. Building knowledge through family of experiments. *IEEE Transactions on Software Engineering* 1999; **25**(4):456–473.
24. Kitchenham B, Pfleeger S, Pickard L, Jones P, Hoaglin D, El Emam K, Rosenberg J. Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on Software Engineering* 2002; **28**(8):721–734.
25. Höst M, Regnell B, Wholin C. Using students as subjects—a comparative study of students and professionals in lead-time impact assessment. *Proceedings of the Conference on Empirical Assessment and Evaluation in Software Engineering (EASE 2000)*. Available at: http://ease.cs.keele.ac.uk/ease2000/Ease2000Extracts.htm [30 May 2005].

AUTHORS' BIOGRAPHIES

Emilio Bellini is a member of RCOST Research Centre on Software Technology at University of Sannio, where he is responsible for KLEOS-RCOST Knowledge Laboratory on Engineering for Organizational Studies. He is an Assistant Professor in Management Engineering in the School of Engineering at the University of Sannio (Italy) where he teaches Economics and Business Organization. He holds a Laurea in Economics and Business, and a PhD in Economic and Management Engineering. His research and professional interests include strategic management of organizational knowledge, small business economics, innovation management, university–industry relationships, ICT sector, and on these topics he has published several papers in international journals and conference proceedings.

Gerardo Canfora is a full professor of computer science at the Faculty of Engineering and the Director of the Research Centre on Software Technology (RCOST) of the University of Sannio in Benevento, Italy. He serves on the program committees of a number of international conferences. He was a program co-chair of the 1997 International Workshop on Program Comprehension; the 2001 International Conference on Software Maintenance; of the 2004 European Conference on Software Maintenance and Reengineering; the 2005 International Workshop on Principles of Software Evolution; and the 2005 International Conference on Software Maintenance. He was the General chair of the 2003 European Conference on Software Maintenance and Reengineering. His research interests include software maintenance, program comprehension, reverse engineering, workflow management, metrics, and experimental software engineering. He serves on the Editorial Board of the *IEEE Transactions on Software Engineering* and *The Journal of Software Maintenance and Evolution: Research and Practice*. He is a member of the IEEE Computer Society.

Félix García holds MSc and PhD degrees in Computer Science from the University of Castilla–La Mancha (UCLM). He is assistant Professor at the Department of Computer Science at UCLM and member of Alarcos Research Group. He is the author of several papers and book chapters on software processes management, from the point of view of their modeling, measurement and technology. His research interests are business process measurement, software processes, and software measurement.

Mario Piattini holds MSc and PhD degrees in Computer Science from the Politechnical University of Madrid, and an MSc in Psychology from the UNED. He is a Certified Information System Auditor and Certified Information Security Manager from ISACA (Information System Audit and Control Association). He is Full Professor at the Department of Computer Science at the University of Castilla–La Mancha in Ciudad Real, Spain. He is the author of several books and papers on databases, software engineering and information systems. He leads the Alarcos research group specialized in information system quality. His research interests are software quality, advanced database design, metrics, software maintenance, information system audit, and security.

Aaron Corrado Visaggio obtained his degree in Electronic Engineering at the Politecnico of Bari, Italy, in 2001. He developed his master thesis at the Fraunhofer IESE, Kaiserslautern, Italy, in the field of Software Process Modeling. He is now concluding his PhD course in Software Engineering at the University of Sannio, Italy. He currently works as a researcher at the Research Centre on Software Technology (RCOST), University of Sannio, Benevento, Italy. His main research interests are empirical software engineering, agile methods, software process modeling and management, knowledge management applied to software engineering.