Int. J. Engng Ed. Vol. 00, No. 0, pp. 1–12, 2007
Printed in Great Britain.

0949-149X/91 $3.00+0.00

Empirical Studies in Software Engineering Courses: Some Pedagogical Experiences*

FÉLIX GARCÍA, MANUEL SERRANO, JOSÉ A. CRUZ-LEMUS, MARCELA GENERO, CORAL CALERO, MARIO PIATTINI
Alarcos Research Group, University of Castilla-La-Mancha, Paseo de la Universidad, 413071 Ciudad Real, Spain. E-mail: felix.garcia@uclm.es

Empirical studies in software engineering are essential for the validation of different methods, techniques, tools, etc. Students play a fundamental role in carrying these studies out successfully and, as a consequence, most experiments connected with software engineering are conducted in academia. Benefits which are concerned exclusively with aspects of research are not the only ones to come from studies of this kind: it is very important also to consider benefits from a teaching point of view. Therefore, when experiments are conducted in academia, they must be planned not only to obtain insights into research but also to help students who participate as experimental subjects.

Keywords: computer science education; software engineering; student experiments

INTRODUCTION

A MAJOR PROBLEM with software engineering is that often a great diversity of methods, languages, tools, environments, etc. are proposed, without their usefulness having been demonstrated in practice. The current competitive market which the software world has become is forcing companies to improve their quality. In many situations this search for quality requires the adoption of new technologies where no evidence about their practical usefulness exists. Other proposals are not considered despite their practical validity. For example, the appearance of the Object Oriented (OO) paradigm is supposed to have had a great impact on software development. Among its advantages, it is said to produce less complex code and be more comprehensible, maintainable and reusable. Currently, however, questions about the extent to which OO technology has fulfilled its promises are answered more by intuitive feelings than by empirical and quantitative evidence [1].

Therefore, it is fundamental for company managers to adopt an 'evidence-based software engineering' approach when making decisions, which could very well bring about great benefits to the operation of the company [2]. For this reason, the focus on empirical methods in software engineering research has been gaining more relevance in the last ten years. By means of empirical methods it is possible to evaluate new contributions before they are introduced in the software processes of companies [3]. The most commonly used empirical studies in software engineering are: controlled experiments, case studies and surveys, which differ from each other with respect to their objectives and restrictions. In the academic environment, the most significant empirical studies, from the perspective of both the researcher and the instructor, are experiments [4].

When carrying out experiments, students can play a very important role. In fact, before performing empirical studies in industrial settings, (which requires a significant cost in terms of time, effort and resources), many researchers carry out pilot studies in academic environments [5]. In this way, most experiments connected with software engineering are conducted in academia [6], and so, students are used as experimental subjects. Many of the publications that present some empirical validation with students as empirical subjects, have focused on the presentation of the benefits that the studies have provided to research, leaving the students and their interests out of the equation. Thus it would appear that in many cases the students have been used in a selfish way, to attain research goals. For this reason we should not only focus egoistically on the role of researcher but also ensure that our students benefit from the experiments.

In fact, empirical studies not only contribute to research, but also provide important pedagogical benefits in software engineering courses, such as gaining practical experience in applying particularly state-of-the-art techniques, also using the experiments as exercises to validate some techniques [7]. Thus, there is a very important link between research and teaching which is important to analyse [5].

Though the potential benefits are great, the introduction of empirical methods into software engineering courses should not be taken lightly. Advantages must be presented to students when experiments are being planned and carried out in

* Accepted 25 March 2007.

their educational environments. Each new experiment must be integrated into the course, lectures and exercises must be prepared, a grading policy must be developed, etc. Generally though, the benefits outweigh the complications [8].

STAKEHOLDERS OF EXPERIMENTATION IN SOFTWARE ENGINEERING

When experiments are conducted in software engineering courses, four main stakeholders can be identified [5]: the **Researcher**, who is responsible for the planning and execution of the experiment; The **Instructor,** who is in charge of the student group which constitutes the context of the experiment; The **Students,** who take part in the experiment; the **Company** which is often the sponsoring body. Each has different objectives and might benefit from the results in different ways. As our perspective in this work is to present the pedagogical benefits of experimentation, we will focus on the objectives and benefits for the students:

- Training in topics which are complementary to regular teaching, given that researchers carry out empirical studies on topics having close connections with the state of the art, since they investigate problems that still need to be solved;.
- Awareness of new problems to be solved in software engineering in general and in companies in particular;
- Better self-assessment of their level in some practical software engineering topics rather than in traditional lectures which focus more on theoretical items;
- Perception of the advantages of empirical studies. Students can gather empirical evidence about the relevance of testing hypotheses concerning new technologies. They appreciate the need to base software engineering improvements on evidence-based grounds. This perception can be achieved by a debate among the students or by means of specific questionnaires. Moreover, the attitude perceived by the instructor when carrying out the experiment is a good insight into the practical utility of this educational initiative;
- Preparation for students' professional future during which on many occasions they will have to face assessments through questionnaires, reports, surveys, etc. Empirical studies may show them that they should not be afraid of being the subjects of empirical studies and data collection activities.

A brief summary of the benefits for the instructors and researchers, is presented in Table 1 (see [5] for further details).

In spite of the benefits for students in being part of experiments, it is very important to know the negative effects that might occur so that care is taken to avoid them [5].

From the students' perspective, problems can arise from the belief that the experiments are a waste of time, especially if they require extensive training that involves several lectures. The alternative is for them to be taught interesting or useful topics more relevant to their future as professionals. Another potential problem occurs when the results of an experiment demonstrate that a method or technique is not useful. It is possible then to look to the pedagogical imperative, which requires that a new or promising technique or technology cannot be accepted without evaluating

Table 1. Benefits of Empirical Studies for Instructors and Researchers

Stakeholder	Benefits
Instructor	Stimulating them to use less conventional means of teaching, especially if the instructor is not used to promoting student participation. Experimentation can be a practical way to teach topics of the subject which are closer to scientific research.
	Encouraging them to introduce problem based software education. There is a universal agreement that learning-by-doing should be more emphasized in software engineering courses [29].
	Stimulating teamwork, to replicate industrial practice. This situation helps students to work in groups.
	Improvement of communication with students. By monitoring the experiment, the instructor obtains much better feedback about what the students have learned.
	Introduction of Empirical Software Engineering as part of the teaching in Software Engineering.
	Maintaining the students' interest, due to a better communication with the students.
	New ways of evaluating students in situations where they do not have the typical stress of a formal examination. Continuous monitoring and observation of students will certainly provide a more accurate and effective evaluation of the students' skills and knowledge.
Researcher	Preliminary evidence to accept or reject the hypotheses.
	Demonstration to companies of the relevance of the research and the usefulness of carrying out empirical studies in their own facilities.
	Allows the prediction of the resources needed to perform experiments in industrial settings and preparation of the required material
	Fine-tunes the organization and details of an empirical study, before it is carried out in an industrial environment.

it in practice. It might also be a good idea to look for any causes of failure.

The design of an experiment is another factor that may affect its outcome. If a within-subjects design is applied, i.e every subject works with all the techniques or methods under evaluation, students can be made aware first-hand of the advantages or disadvantages of using one proposal or another and they are trained in both proposals. However, when between-subject designs are involved, i.e. one group of students constitutes a control group which applies the traditional method and others the experimental group which applies the new method under test, it is fundamental that students are able to participate in an intensive post-experiment session in which both groups discuss the planning and analysis of the experiment in detail and deliver an opinion of the suitability or otherwise of the new technique or proposal. Thus, all students can have an overall rather than one-sided picture of an experiment which would, say, not be profitable in terms of future training.

From the researcher's perspective the drawback of using students as subjects in empirical studies is the effort necessary to prepare them, a fact that should never be ignored. The researcher must expend time and imagination to design the experiment, prepare the materials, run the experiment and analyse the results. Moreover, as a preliminary, the researcher needs to devise an intensive training session with the students, introducing the technique, method or tool that will be used in the experiment.

From the instructor's perspective the problems that can arise are: the need to motivate the instructor to carry out the experiment due to the extra effort in preparation and training that the experiment implies in comparison with a normal lecture and the fact that the instructor has to motivate students accordingly by creating a suitable environment in class. In addition to this, the instructor must also have the ability to resolve any doubts that the students may have. These requirements are largely satisfied if the instructor and the researcher are the same person.

It is therefore important to consider the advantages and drawbacks reported in the planning of empirical studies, the aim being that students perceive the advantages and that, as far as is possible, we avoid the appearance of any problems which include, for example, lack of motivation and dissatisfaction on the part of the student.

THE EXPERIMENTAL PROCESS

The main advantage of experiments is that they can determine the situations in which certain claims or hypotheses are valid and can provide the context in which some standards, methods and tools are advisable. In order to obtain credible and useful results certain essential aspects must to be taken into account [9], [10], [11], [12], [13]. For this reason, it is advisable to follow a systematic process which provides checklists and guidelines about what to do and how to do it. Wholin et al. [10] proposed an experimental process with the following steps:

- **Definition**, in which the experiment is defined in terms of the problem and its objectives. This definition determines *why* the experiment is conducted.
- **Planning**, in which the experiment's design and instrumentation are determined. Planning prepares *how* the experiment is to be conducted.
- **Operation**, in which the experiment is run and the empirical data are collected.
- **Analysis and Interpretation**, in which the collected data are analysed and interpreted by using statistical techniques.
- **Validity Evaluation**, in which the issues that may threaten the experiment's validity are evaluated (construct, internal, external and conclusion validities).
- **Presentation and Diffusion**, where a report with the results is put together, allowing other researchers to replicate the experiment.

EXPERIENCES OF EXPERIMENTATION IN SOFTWARE ENGINEERING COURSES

Two controlled experiments were performed within normal class timetables, taking part in them was voluntary and the students were accordingly motivated as they perceived how potentially beneficial this kind of study could be for them. With the aim of avoiding any possible threats to the experiments' validity, the following steps were taken:

- The subjects who were called to do the experiment had similar experience and knowledge.
- The domains of the models and diagrams provided were simple and common enough to avoid problems of understanding on the part of the students.
- In order to avoid the effects of learning, the diagrams were given to the subjects in a different order.
- The subjects who performed each experiment had not previously participated in one which was similar or identical, so the effects of persistence were attenuated.
- The exercises were part of the knowledge they had to learn in their overall training.
- The students were not allowed either to talk to each other during the running of the experiments or to cheat.
- Any doubts were resolved by the instructor who supervised the experiment.
- The subjects did not have preliminary knowledge either of the aspects the experimenters aimed to study or of the hypotheses formulated.

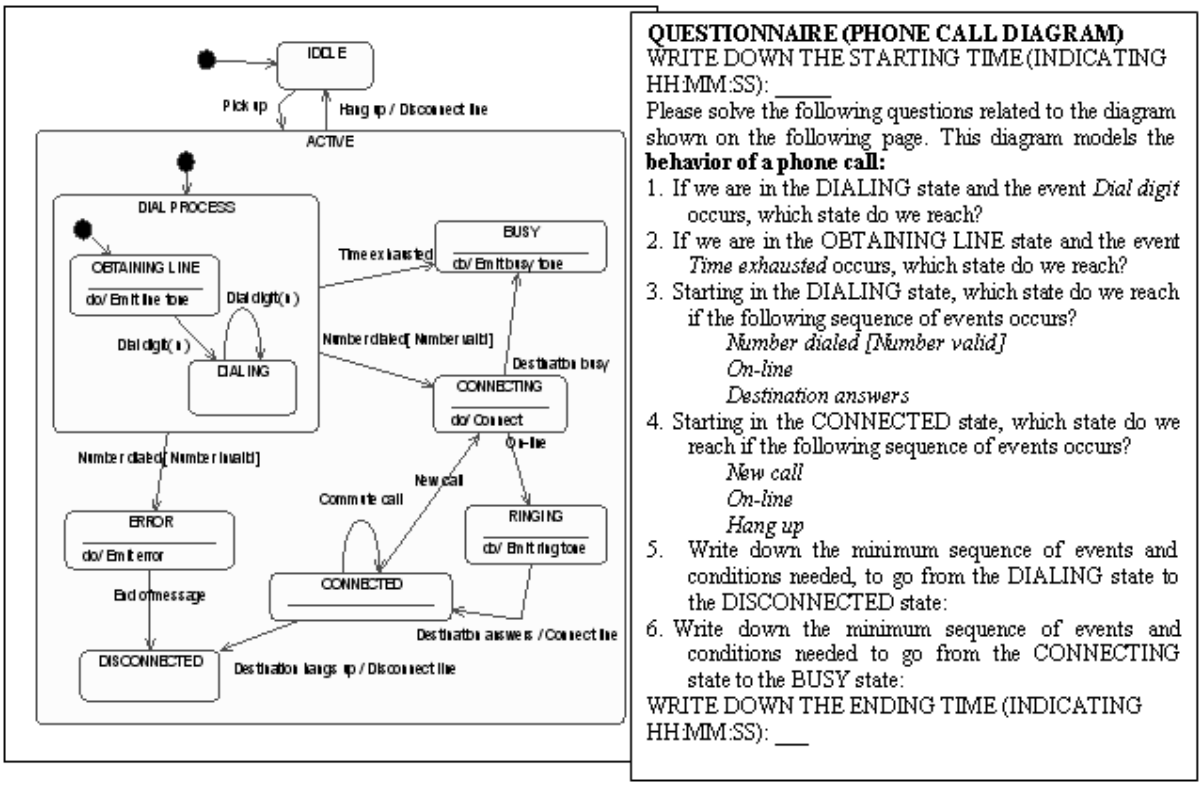

Fig. 1. SPMs Maintainability: Overview of Experimental Design.

Experiment about software process models

The modelling and measurement of software processes have become one of the main levers in promoting their improvement. For the purpose of evaluating the structural complexity of software process models (SPMs), a set of significant metrics has been defined [14], [15]. As is indicated in the literature, the structural complexity of a software product may affect the ease of its maintenance, i.e. its maintainability. However, there is no confirmation of this in the domain of software processes. In order to find out which of the metrics can be applied as useful maintainability indicators, an experiment was carried out [14]. An impression of the experimental design is illustrated in Fig. 1.

Design and execution: planning

The subjects who took part in the experiment were two groups of students enrolled in the Department of Computer Science at the University of Castilla-La Mancha in Spain. The first group was forty-six students in their final (third) year of Computer Science (BSc) in the speciality of management; the second group had forty-one students in their final-year in the systems speciality of Computer Science (BSc). All the subjects had previous knowledge of product modelling (structured analysis and design, UML, databases, etc.) but no experience of software process modelling. For this reason, a lesson was given in which the importance of effective management of software processes in improving the quality of software products was explained, by emphasizing the influence of appropriate process modelling. The language of software process modelling learned was SPEM (*Software Process Engineering Metamodel*) [16].

The experimental material was composed of ten SPMs with different values of structural complexity (obtained by varying the metric values). The models were based on different methodologies and SPMs found in the relevant literature such as, for example, PMBOK [17], Rational Unified Process [18], the Spanish methodology METRICA 3 [19], etc. The objective of the experiment consisted of evaluating whether the structural complexity of the models (independent variable) could affect the understandability and the modifiability of the models (dependent variables). For each model two different exercise sheets were prepared: one on which it was necessary to answer the understandability questions and the other containing modification tasks. The material also included an already- solved example which indicated how to do the experiment. Each subject received material composed of ten models, five containing only understandability questions and the other five consisting only of modification tasks. An excerpt of the material is illustrated in Fig. 2.

The independent variable was measured through the defined metrics and the dependent variables were measured by calculating the time that the subjects spent in answering the questions (understanding time) and in carrying out the required modifications (modifying time). The assumption was that the less time a subject took to understand

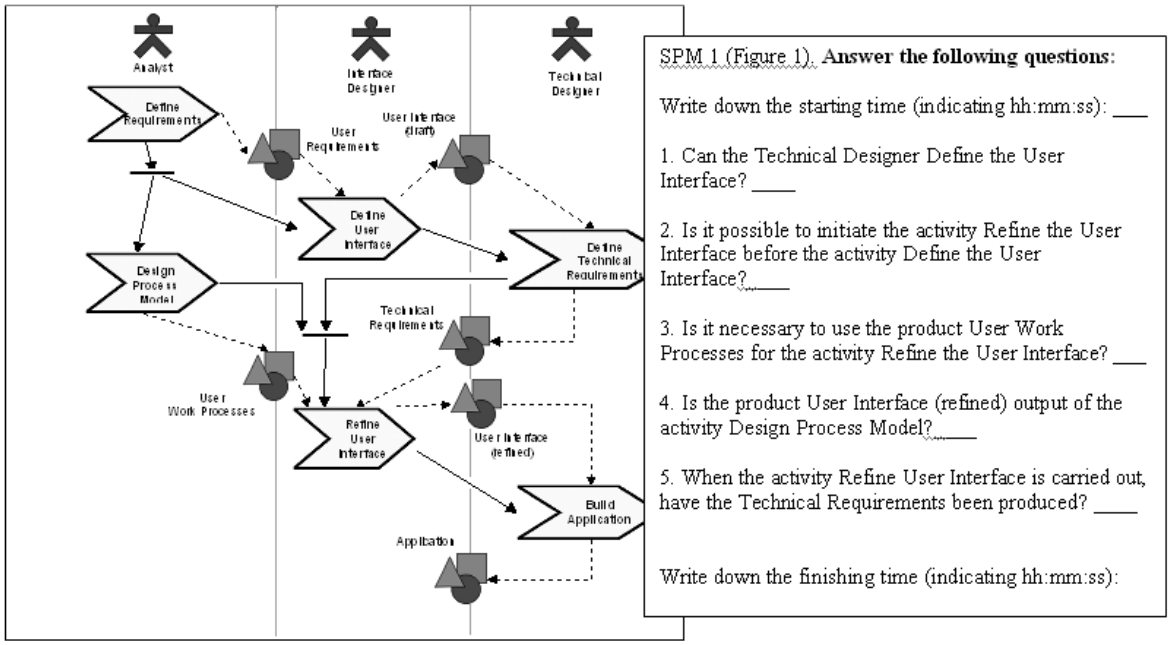

Fig. 2. SPM Experiment Material. Excerpt of Model 1.

and modify a process model, the easier it would be for this model to be maintained.

Design and execution: operation

First of all, a training session was given in which an example, similar to the required tasks of the experiment, was worked through to its solution. In this session the students were familiarized with the kind of tasks to be carried out in the experiment.

Design and execution: analysis and interpretation

By using the Spearman correlation coefficient the relationship between the proposed metrics and the understanding and modifying times was analysed. Table 2 shows the defined metrics and the analysis results.

Table 2 shows, from the researcher's perspective, that significant conclusions were obtained, due to the validation of several metrics (see bold-faced

Table 2. Spearman Correlation Results (a= 0.05).

Metric	UND Time	MOD Time
NA (Number of Activities)	**0.841**	**0.640**
	p = 0.002	p = 0.046
NWP (Number of Work Products)	**0.826**	**0.650**
	p = 0.003	p = 0.042
NPR (Number of Process Roles)	0.074	0.377
	p = 0.838	p = 0.283
NDWPIn (Number of input dependences of the Work Products with the Activities)	**0.786**	**0.738**
	p = 0.007	p = 0.015
NDWPOut (Number of output dependences of the Work Products with the Activities)	**0.886**	**0.791**
	p = 0.001	p = 0.006
NDWP (Total number of dependencies of the WorkProducts with the Activities)	**0.893**	**0.707**
	p = 0.001	p = 0.022
NDA (Number of precedence dependences among activities)	**0.821**	0.599
	p = 0.003	p = 0.067
NCA (Activity Coupling) Level)	**−0.752**	−0.44
	p = 0.012	p = 0.203
RDWPIn (Ratio Input Work Products Dependences)	0.79	0.115
	p = 0.828	p = 0.751
RDWPOut (Ratio Output Work Product Dependences)	−0.79	−0.115
	p = 0.828	p = 0.751
RWPA (Total Ratio Work Product-Activity Dependences)	−0.116	−0.30
	p = 0.751	p = 0.934
RRPA (Ratio Process Roles and Activities)	−0.560	−0.141
	p = 0.092	p = 0.697

values), thus demonstrating that some had influence on the maintainability of the models. These metrics could be used as indicators of the time necessary to understand and modify a SPM. They could also be useful in comparing and evaluating the best model among those which are semantically equivalent but with varying complexity.

PEDAGOGICAL ANALYSIS OF THE EXPERIMENT DEALING WITH SPMS

From the students' perspective, the run of this experiment in the third year of a software engineering course allowed us first of all to teach a special lesson about software process modelling and technology. In this lesson, the students studied the definition of software processes, their most significant elements and the importance of the quality management of the processes and learned how to model software processes in detail. This helped them to have a broader vision of software engineering. The topic of software processes is only touched upon in normal lectures when software lifecycles are tackled.

Moreover, the students, who are only used to applying software product modelling techniques during the course, also learnt about process modelling. Most aspects of this topic belonged to state-of-the-art research, so the students were aware of the problems being addressed currently in process modelling.

On the other hand, the students worked with software process models which had different complexities as regards their ease of maintenance; as a consequence they perceived the importance of providing more maintainable process models to improve their quality and, ultimately, management.

The pedagogical benefits that students obtained when the results were communicated must also be considered. For this reason, another lecture was given where, before they did anything else, they were made aware of the importance of carrying out empirical studies. We explained to them that more maintainable software process models can benefit from the management of software processes. They become able to guarantee the understanding and diffusion of the processes as they evolve, without being stopped from bringing a successful execution to completion. They can reduce the effort necessary to change the models, thereby reducing maintenance costs. This is due to their influence on the software lifecycle costs [20]. They were also encouraged by having their results compared with those obtained in the same experiment by qualified engineers working in a software company and with students from two Italian universities and finding that results were similar (see Figs 3 and 4), which demonstrates that, under some circumstances, there are no great differences between students of final-year courses and practitioners. As we can see, this experiment was very profitable for these students, who are the qualified professionals of tomorrow.

Finally, it is important to emphasize that the training of students in the topics tackled in the current experiment was more profitable for being included in the context of empirical research. By means of conventional teaching the training of subjects could be only partially attained. The use of an empirical study allowed students to demonstrate the usefulness of the metrics not only based on subjective criteria, but also on quantitative and

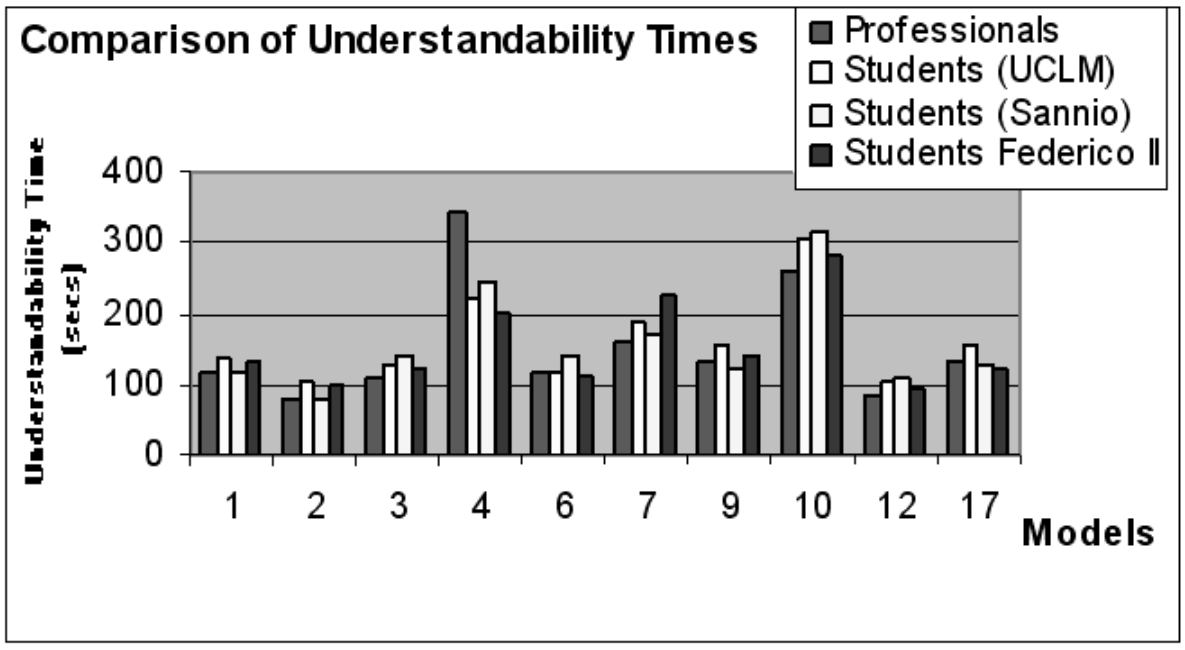

Fig. 3. Comparison of SPM Understandability Times.

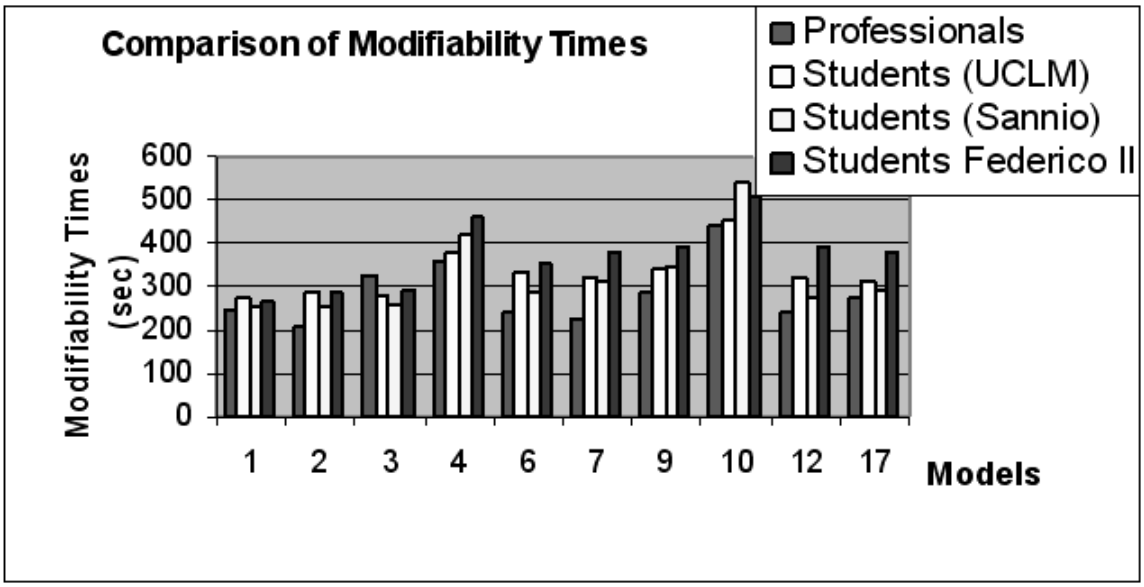

Fig. 4. Comparison of SPM Modifiability Times.

objective data. Such objectivity can only be attained by applying empirical studies. Furthermore, students worked as experimental subjects, regarded as basic training for future experimenters who need to know how a subject might behave in order to learn how to properly design experiments and to avoid possible threats to validity.

Experiment for evaluating the effect of composite states on the understandability of UML statechart diagrams

UML statechart diagrams have become an important technique for describing the dynamic aspects of a software system and are also a relevant element of OO design documents [21]. In a previous work [22], we have already defined a set of metrics for evaluating the structural properties (size and structural complexity) of UML statechart diagrams and partially validated them as early indicators of UML statechart diagrams understandability, through a family of experiments.

These previous experiments have also revealed that some UML constructs such as the fact that the number of simple states and the number of events, etc. seem to have a close relationship with the understandability of UML statechart diagrams.

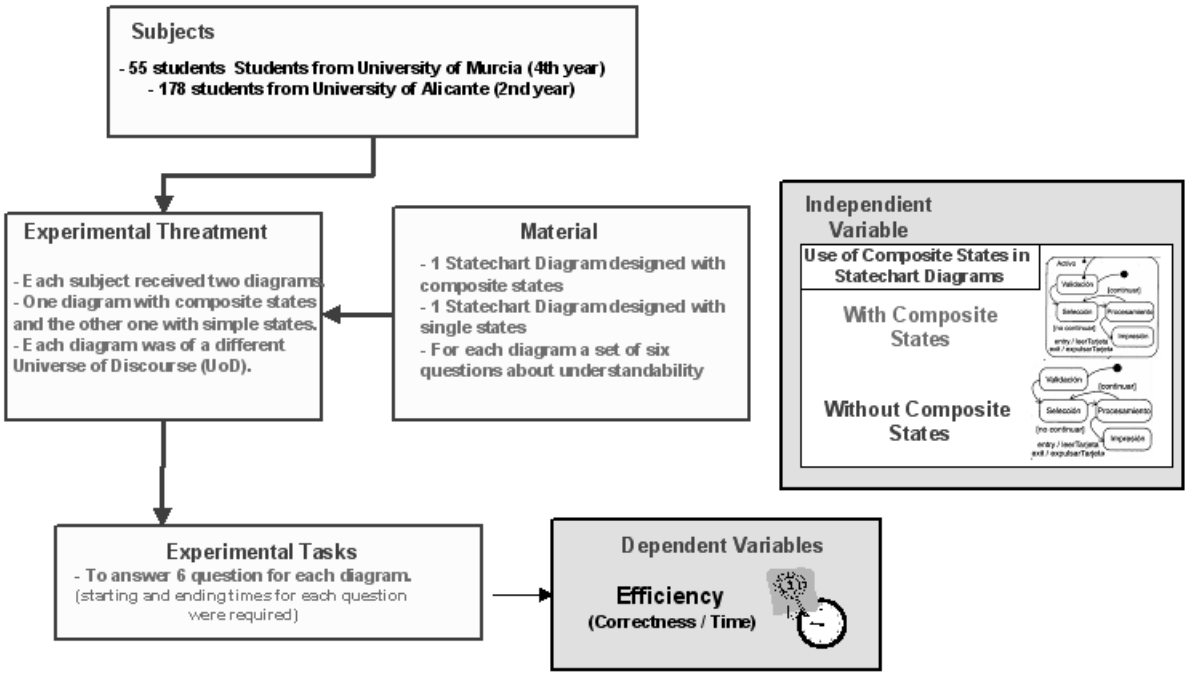

Fig. 5. UML Statecharts: Overview of Experimental Design.

But in these studies composite states apparently did not influence the diagrams' understandability. Having read what was reported about this issue in the literature, this fact seemed quite suspicious to us [23], so we decided to go a step further and perform a controlled experiment (see Fig. 5), along with a replication of it, to investigate the effect that composite states have on the understandability of UML statechart diagrams [22].

Design and execution: planning

The experiments and the replication were performed by computer science students in Spanish Universities: in the original experiment 55 students (enrolled in the fourth year), who are taking a second software engineering course at the University of Murcia participated, and in the replication 178 students (enrolled in the second year), who are taking a first software engineering course, at the University of Alicante participated. So, the subjects in the original experiment were more experienced in working with UML diagrams than the subjects of the replication.

We selected a factorial with interaction confounded experimental design. The dependent variable was the understandability of UML statechart diagrams. The independent variables were the Universe of Discourse (UoD) to which the diagrams were related and the Use of Composite States (CS) in statechart diagrams. We used two UoDs: an ATM machine and a phone call. For each of them, we presented two different diagrams, which were conceptually identical. One of the diagrams included composite state(s) and the other did not.

Table 3. Material Assignment

	Universe of Discourse	
	ATM machine	Phone call
Without composite states	Group A	Group B
With composite states	Group B	Group A

As each subject would receive two diagrams, one with and the other without composite states, and each of them related to a different UoD, we obtained two different groups as shown in Table 3. The diagrams of each group were given to the subjects in different orders. For instance, in group A, the subjects first had to solve the tasks related to an ATM machine without composite states and, after that, those related to a phone call with composite states or exactly the same tasks for the same diagrams but in a different order (the phone call with composite states and then the ATM machine without composite states).

Group A had 28 subjects and group B 27 subjects in the original experiment and 92 and 86 subjects respectively, in the replication.

Each diagram had a test which contained six questions which were conceptually similar and set out in the same order. In fact, in both diagrams of each UoD, the questions were the same. An example of the experimental material is shown in Fig. 6.

The subjects had to write down the times at which they started and finished answering the questions, as well as providing the answers to the questions themselves. From this, we obtained their

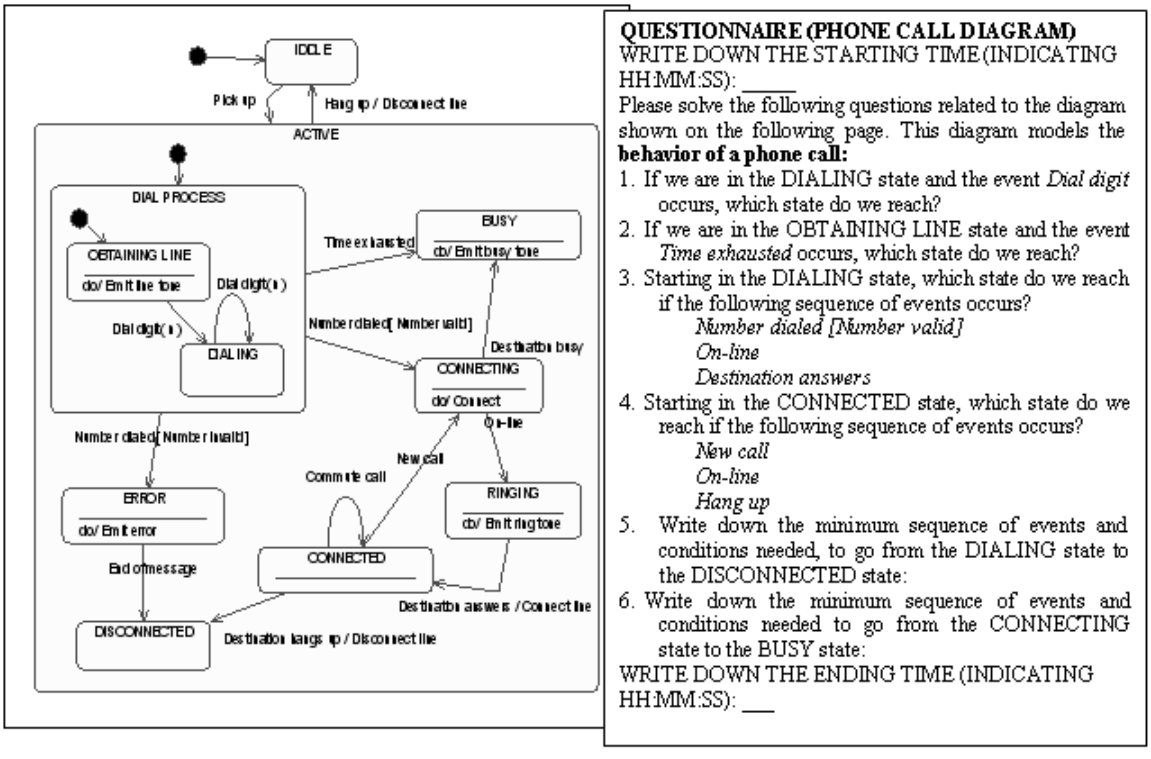

Fig. 6. Example of UML statechart diagram with composite states (phone call).

understandability efficiency, defined as the number of correct answers divided by the time the subjects spent answering them. We therefore need to test whether the use of composite states improves the efficiency of the subjects when carrying out understanding tasks.

Design and execution: operation

The experiment started with a twenty-five minute introductory session in which the instructor briefly explained what the experiment had been designed to achieve as well as the constructs of UML statechart diagrams. After that, the materials for the experiment were randomly distributed to the subjects.

At this point two examples in shortened versions were performed by the instructor, who explained the correct answer to each question and how to correctly note down the starting and finishing times.

Design and execution: analysis and interpretation

The results obtained through an ANOVA reveal that:

- The use of composite states noticeably improved the understandability of the diagrams in the original experiment, i.e. the level of efficiency of the subjects was significantly better.
- In the replication, there seemed to be no difference in the understandability efficiency of the subjects when they worked with or without composites states.

The difference between both populations could be caused by the fact that the subjects of the replication were not sufficiently skilled in the use of UML statechart diagrams, so they did not appreciate the advantages of composite states. Therefore, the experimentation showed that the use of composite states could improve the understandability efficiency of UML statechart diagrams if the subjects have a certain level of experience in working with this material.

PEDAGOGICAL ANALYSIS OF AN EXPERIMENT WITH COMPOSITE STATES IN UML STATECHART DIAGRAMS

Through experimentation we have obtained the following principle benefits:

- The students gather their own empirical evidence about the influence of composites states on the understanding of UML statechart diagrams. Thus, they have confirmed the intuition about composites states.
- The students have appreciated the need to corroborate through experimentation certain claims that are sometimes only based on experience or intuition.
- The experiment was very profitable for students in their training, as UML Statechart Diagrams is part of the knowledge they have to acquire on the software engineering course.
- The students demonstrated a very positive attitude when carrying out the experiment and in later debate they fully agreed upon the relevance of this type of exercises in software engineering courses.
- Another very interesting finding is the fact that when someone faces the task of understanding a diagram of this kind, the presence of composite states can help to carry it out in a more efficient way, always supposing that the students have certain previous knowledge and experience of UML statechart diagrams. This is why we consider that it is highly advisable for teachers to pay special attention to the use of composite states in UML statechart diagrams in software engineering courses.

CONCLUSIONS

In this paper, we have tackled the pedagogical view of running empirical studies in software engineering courses by means of two experiences in which the potential benefits of empirical studies for students were considered in the planning phase.

As a result diverse pedagogical benefits were obtained. Among those previously mentioned, one of the outcomes of the planning, execution and communication of results, is the importance of early training of students in the empirical method. It is what makes software engineering a 'science'. This point of view is not restricted to software engineering: good science in general is characterized by its emphasis on empiricism [24]. This must be instilled into students whenever possible in software engineering courses. The running of experiments in academic environments is a good way to achieve success in this respect. Along with training in the empirical method, we must not forget that carrying out experiments with students educates them in topics that are near to state-of-the-art research. Although most of the benefits that may be achieved by running experiments can be acquired by other teaching techniques, both these benefits are almost exclusively down to experiments.

Furthermore, as a measurement of the benefits that this experience gave our students, we carried out a post-experiment session where we provided them with all the experimental material and the results. After which we discussed the development and outcome of the experiments with them. These sessions were very positive from the point of view of both the instructor and the students. The students learnt how to perform experiments, how to analyse and interpret the empirical data, and the experiments also allowed them to doubt prejudices and statements that were widely-accepted but which had not previously been validated, when the results were not what they expected [25]. The enthusiasm shown by the majority of students in

participating in and performing the experiments, as well as their interest in knowing the final results, is a clear indicator that these studies are beneficial and should be carried out in all universities, as indeed is happening in many international academic institutions [26], [27].

The results demonstrate, we think, that when planning empirical studies in software engineering courses the pedagogical benefits they provide must be kept in mind. A careful experimental plan conducted in the right teaching period and right context (skills and capabilities of the students of the course, topics, etc.) can benefit all the stakeholders involved. Furthermore, diverse ethical considerations [28] must be considered in order to avoid any lack of motivation or dissatisfaction on the part of the students. In the end, the benefits should be greater than the disadvantages of performing experiments; if this is not the case, we should not make that empirical validation.

Our experience convinced us that the participation of students in empirical studies is of great relevance to them and does not only provide benefits to research. Therefore, we plan to continue performing experiments in our software engineering courses.

REFERENCES

1. I. Deligiannis, Shepperd M., Webster S. and Roumeliotis M. A Review of Experimental into Investigations into Object-Oriented Technology, *Empirical Software Engineering*, **7**, 2002, pp. 193–231.
2. T. Dyba, B. Kitchenham and M. Jorgensen, Evidence-Based Software Engineering for Practitioners, *IEEE Software*, **22(1)**, 2005, pp. 58–65.
3 M. Höst, Introducing Empirical Software Engineering Methods in Education, *Proc. 15th Conference in Software Education and Training (CSEET'02)*, 2002, pp. 170–179.
4. L. Baresi, S. Morasca and P. Paolini, Estimating the Design Effort of web Applications, *Proc. 9th International Software Metrics Symposium (METRICS'03)*, 2003, pp. 62–71.
5. J. Carver, L. Jaccheri, S. Morasca and F. Shull, Issues in Using Students in Empirical Studies in Software Engineering Education, *Proc. 9th International Software Metrics Symposium (METRICS'03),* 2003, pp. 239–251.
6. D. Sjøberg, Hannay, J., Hansen, O., By Kampenes, V., Karahasanovic, M., Liborg, N and Rekdal, A. A Survey of Controlled Experiments in Software Engineering. *IEEE Transactions on Software Engineering*, **31(9),** 2005, pp. 733–753.
7. M. Ciolkowski, Muthig, D. and Rech, J. Using academic courses for empirical validation of software development processes. 30th Euromicro conference (EUROMICRO'04). 1089–6503/04. *IEEE Computer Society*, 2004, pp. 354–361.
8. D. Port and Klappholz, D. Empirical Research in the Software engineering classroom. 17th Conference on software engineering education and training (CSEET'04). 1093–0175/04. *IEEE Computer Society*, 2004, pp.
9. N. Juristo and A. Moreno, *Basics of Software Engineering Experimentation*, Kluwer Academic Publishers, (2001).
10. C. Wohlin, P. Runeson, M. Höst, M. Ohlson, B. Regnell and A. Wesslén, *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, (2000).
11. D. Perry, A. Porter and L. Votta, Empirical Studies of Software Engineering: A Roadmap, *Future of Software Engineering,* ACM, Anthony Finkelstein Ed., pp. 345–355, (2000).
12. V. Basili, F. Shull and F. Lanubile, Building Knowledge Through Families of Experiments, *IEEE Trans. on Software Engineering*, **25(4),** 1999, pp. 435–437.
13. B. Kitchenham, S. Pfleeger, L. Pickard, P. Jones, D. Hoaglin, K. El Emam and J. Rosenberg, "Preliminary Guidelines for Empirical Research in Software Engineering", *IEEE Trans. on Software Engineering*, vol. 28(8), 2002, pp. 721–734.
14. F. García, F. Ruiz, and M. Piattini, Definition and Empirical Validation of Metrics for Software Process Models, *Proc. 5th International Conference Product Focused Software Process Improvement (PROFES'2004)*, LNCS, 3009, Kansai Science City (Japan), pp. 146–158, (2004).
15. F. García, F. Ruiz, F. and M. Piattini, An Experimental Replica to Validate a set of Metrics for Software Process Models, *Proc. European Software Process Improvement Conference, LNCS,* 3281, pp. 146–158, (2004).
16. Software Process Engineering Metamodel Specification; adopted specification, version 1.0. Object Management Group. November 2002. Available http://cgi.omg.org/cgi-bin/doc?ptc/02–05–03
17. Project Management Institute, PMBOK: A Guide to the Project Management Body of Knowledge. Project Management Institute Communications, United States, (2000).
18. I. Jacobson, G. Booch and J. Rumbaugh, *The Unified Software Development Process*, Addison Wesley, (1999).
19. METRICA 3: Methodology for the Planning, Development and Maintenance of Information Systems, Ministry of Public Administrations, Spain, available on http://www.csi.map.es/csi/metrica3/ (in Spanish)
20. T.M Pigoski, *Practical Software Maintenance. Best Practices for Managing your Investment*, John Wiley & Sons, (1997).
21. J. A. Cruz-Lemus, M. Genero and M. Piattini, Metrics for UML Statechart Diagrams, in *Metrics for Software Conceptual Models*. M. Genero, M. Piattini and C. Calero Eds., Imperial College Press, UK, (2005).
22. J. A. Cruz-Lemus, A., M. Genero, M[a]. E. Manso and M. Piattini, Evaluating the Effect of Composites States on the Understandability of UML Statechart Diagrams, MODELS/UML 2005, *LNCS 3713*, pp 113–125, (2005).

23. G. Booch, J. Rumbaugh and I. Jacobson. The Unified Modeling Language User Guide, 2nd edition, Addison-Wesley Professional, (2005).
24. S. L. Pfleeger, Soup or Art? The Role of Evidential Force in Empirical Software Engineering, *IEEE Software*, 2005, pp. 66–73.
25. A. Endres and D. Rombach, *A Handbook of Software and Systems Engineering: Empirical Observations, Laws, and Theories*, Addison-Wesley, (2003).
26. J. Carver, L. Jaccheri, S. Morasca and F. Shull, Using Empirical Studies during Software Courses, *Experimental Software Engineering Research Network 2001-2003, LNCS,* ***2765*****,** 2003, pp. 81–103.
27. M. Höst, B. Regnell and C. Wholin, Using Students as Subjects—A comparative Study of Students & Professionals in Lead-Time Impact Assessment, *Proc. 4th Conference on Empirical Assessment & Evaluation in Software Engineering (EASE)*, Keele University, UK, 2000, pp. 201–214.
28. J. Singer and N. Vinson, Ethical Issues in Empirical Studies of Software Engineering, *IEEE Trans. Software Engineering*, **28 (12)**, 2002, pp. 1171–1180.
29. G. Mitchel and J. Declan Delaney. An assessment strategy to determine learning outcomes in a Software Engineering Problem-based Learning Course, *International Journal on Engineering Education*, **20 (3),** 2004, pp. 494–502.
[11] Denger, C. and Ciolkowski, M.: High Quality Statecharts through Tailored. Perspective-Based Inspections. Proc. of 29th EUROMICRO Conference ''New Waves in System Architecture''. Belek, Turkey. (2003) 316–325
[4] L. Briand, S. Arisholm, F. Counsell, F. Houdek, and P. Thévenod-Fosse, "Empirical Studies of Object-Oriented Artefacts, Methods, and Processes: State of the Art and Future Directions", *Empirical Software Engineering*, vol. 4(4), pp. 387–404, 2000.

Félix García is M.Sc. and Ph.D. in computer science from the University of Castilla-La Mancha (UCLM) in Ciudad Real, Spain. He is Assistant Professor in the Department of Information Technologies and Systems at UCLM and member of Alarcos Research Group. His research interests include business process management, software processes, software measurement, empirical software engineering and agile methods. About these topics he has published two books and several book chapters, articles in journals and papers at national and international conferences.

Manuel Serrano is M.Sc. and Ph.D. in computer science from the University of Castilla-La Mancha in Ciudad Real, Spain. He is Assistant Professor at the Escuela Superior de Informática and member of the Alarcos Research Group, in the same University, specializing in information systems, databases and software engineering. His research interests are: data warehouses quality and metrics, software quality.

José A. Cruz-Lemus is an Assistant Professor at the Department of Technologies and Information Systems at the University of Castilla-La Mancha in Ciudad Real, Spain. He received his M.Sc. degree in computer science there in 2003. He will receive his Ph.D. degree at the University of Castilla-La Mancha by the summer of 2007. His main interests in research are: empirical software engineering, software metrics and the quality of conceptual data models. His works has been published at several conferences, such as E/R, MoDELS/ UML, ISESE, SEKE, etc.

Marcela Genero is Associate Professor at the Department of Information Systems and Technologies at the University of Castilla-La Mancha, Ciudad Real, Spain. She received her M.Sc. degree in computer science in the Department of Computer Science of the University of South, Argentine in 1989, and her Ph.D. at the University of Castilla-La Mancha, Ciudad Real, Spain in 2002. Her research interests are: empirical software engineering, software metrics, conceptual data models quality, database quality, quality in product lines, quality in MDD, etc. She has edited two books and published in prestigious journals and conferences. She has served as reviewer of many workshops, conferences and journals. She is member of the International Network of Empirical Software Engineering (ISERN).

Coral Calero is M.Sc. and Ph.D. in computer science. She is Associate Professor at the Escuela Superior de Informática of the Castilla-La Mancha University in Ciudad Real. She is a member of the Alarcos Research Group. Her research interests are: advanced databases design, database/datawarehouse quality, web/portal quality, software metrics and empirical software engineering. She is the author of articles and papers in national and international conferences on these subjects.

Mario Piattini is full professor at UCLM. His research interests include software quality, metrics and maintenance. He has a Ph.D. 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 Centre. He is a member of ACM and IEEE Computer Society.