Conducting Realistic Experiments in Software Engineering

Dag I.K. Sjøberg, Bente Anda, Erik Arisholm, Tore Dybå, Magne Jørgensen,
Amela Karahasanovic, Espen F. Koren and Marek Vokác

Simula Research Laboratory, P.O. Box 134, NO-1325 Lysaker, Norway
Email: Dag.Sjoberg@simula.no, Telephone: +47 67 82 83 00

Abstract

An important goal of most empirical software engineering research is the transfer of research results to industrial applications. Two important obstacles for this transfer are the lack of control of variables of case studies, i.e., the lack of explanatory power, and the lack of realism of controlled experiments. While it may be difficult to increase the explanatory power of case studies, there is a large potential for increasing the realism of controlled software engineering experiments. To convince industry about the validity and applicability of the experimental results, the tasks, subjects and the environments of the experiments should be as realistic as practically possible. Such experiments are, however, more expensive than experiments involving students, small tasks and pen-and-paper environments. Consequently, a change towards more realistic experiments requires a change in the amount of resources spent on software engineering experiments.

This paper argues that software engineering researchers should apply for resources enabling expensive and realistic software engineering experiments similar to how other researchers apply for resources for expensive software and hardware that are necessary for their research. The paper describes experiences from recent experiments that varied in size from involving one software professional for 5 days to 130 software professionals, from 9 consultancy companies, for one day each.

Keywords: Empirical software engineering, technology transfer, experiments, professionals

1. Introduction

There is an increasing understanding in the software engineering (SE) community that empirical studies are needed to develop or improve processes, methods and tools for software development and maintenance (Basili *et al.* 1986, Basili *et al.* 1993, Rombach *et al.* 1993, Basili 1996, Tichy 1998, Zelkowitz & Wallace 1998). The classical method for identifying cause-effect relationships is to conduct *controlled* experiments where only a few variables vary. Controlled experiments in software engineering often involve students solving small pen and paper tasks in a classroom setting. A major criticism of such experiments is their lack of realism (Potts 1993, Glass 1994), which may deter technology transfer from the research community to industry. The experiments would be more realistic if they are run on real tasks on real systems with professionals using their usual development technology in their usual working environment. Note, however, keeping control *is* a challenge when the realism is increased.

A prerequisite for the discussion on realism is that we are conscious about the population we wish to make claims about. Implicit in our discussion is that the interesting population is "representative" software builders doing "representative" tasks in "representative" industrial environments. However, it is far from trivial what "representative" means. Generally, a weakness of most software engineering research is that one is rarely explicit about the target population regarding tasks, subjects and environments.

The purpose of this paper is to describe (1) why we need more realistic experiments and (2) some experiences on how to run more realistic experiments. During the last couple of years the authors of this paper have conducted 12 controlled experiments with a total of 750 students and 300 professionals as subjects.[1] Additionally, the authors of this paper have during the last 6 years conducted about 30 case studies as part of two major Norwegian Software Process Improvement projects (SPIQ, PROFIT).

The remainder of this paper is organised as follows. Section 2 discusses the notion of realism in SE experiments. Partly based on our experiences on conducting controlled SE experiments, Sections 3, 4 and 5 discuss in more details the realism of respectively tasks, subjects and environment. Section 6 concludes.

[1] Information about most of these experiments can be found at http://www.ifi.uio.no/forskning/grupper/isu/forskerbasen.

2. Realism in Software Engineering Experiments

The ultimate criterion for success in an applied discipline such as software engineering research is the widespread adoption of research results into everyday industrial practice. To achieve this, diffusion of innovation models emphasizes the importance of *homophily*, which Rogers (1995) defines as the degree to which the innovator and the potential adopter are similar in certain attributes such as objectives, beliefs, norms, experience and culture. *Heterophily* is the opposite of homophily, and, according to Rogers (1995), "*one of the most distinctive problems in the diffusion of innovations is that the participants are usually quite heterophilous*" (*ibid*., p. 19, italics in original). This means that evidential credibility of software experiments depends on both the producer and the receiver of the results. Hence, without a close tie between the experimental situation and the "real", industrial situation, practitioners will tend to perceive the experiment as irrelevant and ignore the results.

According to the Merriam-Webster dictionary, *realism* can be defined as a "concern for fact or reality and rejection of the impractical and visionary" and to "accurate representation without idealization." Similarly, the Oxford dictionary defines realism as a "close resemblance to what is real; fidelity of representation, rendering the precise details of the real thing or scene."

Most of the studies in software engineering that have emphasized realism are so far case studies. However, a major deficiency of case studies is that many variables vary from one case study to another so that comparing the results to detect cause-effect relationships is difficult. Therefore, controlled experiments should be conducted to *complement* case studies in empirical software engineering. For example, we know that evolving software is likely to decay regarding, e.g., changeability (Arisholm & Sjøberg 2000, Arisholm *et al*. 2001). The reasons for this decay, however, cannot always be analyzed in a case study. Instead, we need to isolate and analyze a few variables at a time in controlled experiments. For example, if the purpose of a study is to determine how design principles affect changeability, a case study would be unable to separate the effect of developer skill and design principles. A controlled experiment, on the other hand, would permit the same changes to be implemented by similarly skilled developers applying alternative design principles.

So, while the *raison d'être* for experimental research is to establish evidence for causality through *internal* logical rigor and control, this is not enough. It is also important to ensure *external* validity. If an experiment lacks external validity, its findings hold true only in experimental situations, making them useless to both basic and applied research. An important issue, therefore, is whether the particular features of formal SE experiments are realistic.

There are two types of realism to consider (Aronson & Carlsmith 1968). One is *experimental realism*, which refers to the impact of an experimental treatment on subjects. It occurs when the experiment appears real and meaningful to the subjects and when the experiences encountered in the experiment will occur in the real world. In other words, if an experiment has impact upon a participant, forces the participant to take the matter seriously and involves the participant in the procedures, then we say the procedure is high on experimental realism.

The other type of realism, which is of most concern to us in this paper, is *mundane realism*. Mundane realism refers to the resemblance of an experiment with real world situations and, therefore, with our ability to generalize the results of the experiment to industrial practice. In this sense, an experiment is realistic if the situation presented to the subjects is realistic and the subjects react to the situation in the same way as they would do in their usual work environment. In particular, it is a challenge to achieve realism regarding experimental tasks, subjects and environment (Harrison 2000):

- *Realistic tasks*. This challenge is concerned with the size, complexity and duration of the involved tasks. Most experiments in software engineering seem simplified and short-term in which "the experimental variable must yield an observable effect in a matter of hours rather than six months or a year" (Harrison 2000). Such experiments are hardly realistic given the tasks of building and maintaining real, industrial software, particularly since many of the factors we wish to study require significant time before we can obtain meaningful results.
- *Realistic subjects*. This challenge is concerned with the selection of subjects to perform the experimental tasks, that is, to what extent do the selected subjects represent the population that we wish to make claims about? Even though there are some preliminary indications that students can be used for certain tasks instead of professionals under certain conditions (Höst *et al.* 2000), it is still unclear how well results from student-based experiments generalize to professional software engineers (Harrison 2000). It is worrying, therefore, that most of these studies attempt to generalize their results to an industrial environment.
- *Realistic environment*. Even when realistic subjects perform realistic tasks, the tasks may be carried out in an unrealistic manner. The challenge is to configure the experimental environment with an infrastructure of supporting technology (processes, methods, tools, etc.) that resembles an industrial development environment. Traditional pen and paper based exercises used in a

classroom setting are hardly realistic for dealing with relevant problems of the size and complexity of most contemporary software systems.

While our focus is on controlled experiments, this does not mean that we are only concerned with laboratory, or *in vitro*, experiments. Controlled experiments can also be conducted *in vivo*, in a more realistic environment than is possible in the artificial, sanitized laboratory situation (Basili 1996). However, the realistic environment can also be a weakness, because it may be too costly or impossible to manipulate an independent variable or to randomize treatments in real life. Thus, the amount of control varies on a continuum, and prioritizing between the validity types is an optimization problem, given the purpose of the experiment. Nevertheless, external validity is always of extreme importance whenever we wish to generalize from behaviour observed in the laboratory to behaviour outside the laboratory, or when we wish to generalize from one non-laboratory situation to another non-laboratory situation.

In the remainder of this paper, we will show how we have dealt with these challenges of conducting realistic experiments in SE, by substituting students with professionals, toy tasks with relevant problems, pen and paper with commercial tools, and the classroom setting with a real industrial environment.

3. Realistic Tasks

When conducting controlled experiments in software engineering, one should consider the realism and representativeness of the tasks regarding the size, complexity and duration of the involved tasks. Of course, the world is diverse, so the realism of a task must be considered relative to a certain part or aspect of the world. Some experimental tasks bear in our opinion little resemblance to actual tasks in software engineering; others are very similar to actual tasks. In between there is a continuum. Larger development tasks may take months, while many maintenance tasks may take only a couple of hours.

A systematic way to define realistic tasks according to a given application area in a given context, is to collect information about the kinds and frequencies of tasks in the actual environment and then create “benchmark tasks”, i.e., a set of tasks that is a representative sample of tasks from the population of all tasks. An example use of such benchmark tasks is described in (Jørgensen & Bygdås 1999). In that study, the maintenance benchmark tasks were derived from another study of 109 randomly sampled maintenance tasks (Jørgensen 1995).

In yet another study, we collected information about all the maintenance tasks in a tool vendor company through a Web interface during a period of six months (Arisholm & Sjøberg 2000).

Generally, to increase the realism of SE experiments, the duration of the studied tasks should be increased. As far as we have observed, the tasks carried out in most student experiments take only up to one hour (a few of them two hours) – to fit with the time schedule of a university class. In an experiment on object-oriented (OO) design principles, however, 60 students and 130 professionals (junior, intermediate and senior consultants from the companies Accenture, Cap Gemini Ernst & Young, TietoEnator, Ementor, Software Innovation, Genera, Ementa and ObjectNet) spent one day each on the five experimental tasks.

In an experiment on design patterns that took place in May 2002, 44 professionals from 12 companies spent 3 days (including a course on design patterns the second day). This experiment was a replication of a former experiment in which only pen and paper were used (Prechelt *et al.* 2000); this time the programming tasks were actually carried out using real development tools (see also Section 5).

We have conducted one longer-term, one-subject explorative study (35 hours), that is, an “N=1 Experiment” (Harrison 2000), which is not a controlled experiment since only one subject takes part (Karahasanovic 2002). However, the longer duration of this study allowed a wider spectrum of tasks to be carried out. The system being subject of the tasks was also larger in size and complexity than usual in most experiments. Another positive effect of the longer duration was that the pressure put on the subject was less, that is, more realistic, than what we have experienced in the controlled experiments we have run. In the student experiments, most students felt as if they were in an exam situation. “How did it go?”, they asked after the experiment.

Another example of an experiment with high realism of tasks is our ongoing study on uncertainty in the estimation of development effort. In that experiment we pay an organization to evaluate three estimation processes. The organization now estimates one third of their incoming projects respectively according to the first, second and third estimation process.

Increasing the duration of the experiments enables more realistic tasks to be carried out. We have tried several means to achieve longer experiments, some of them with success (see Section 4.2). Of course, our tasks may still be small compared with many actual tasks. We are therefore considering an experiment where an application system that is actually requested by Simula Research Laboratory, will be developed by 4 different project teams (each consisting of 4 persons) from 4 different consultancy companies. It should be possible to develop the system in approximately 1 month. This would then be a very realistic development task. Of course, the number of subjects (here teams) is too small to conduct hypothesis testing, but we still have some control. Nevertheless, there are many challenges to such an experiment; we are only in the brainstorming phase.

4. Realistic Subjects

The similarity of the subjects of an experiment to the people who will use the technology impacts the ease of the technology transfer (Rogers 1995). A common criticism of experiments in software engineering is that most of the subjects are students, which might make it difficult to generalise the results to settings with professionals. We will emphasise, though, that there are sub-areas of software engineering where the proportion of professionals versus students is relatively high. For example, experiments on inspections have traditions for using professionals, e.g., (Seaman & Basili 1997, Zhang *et al*. 1998, Porter & Votta 1998, Laitenberger *et al*. 2000), and about 50% of the reported software effort estimation experiments use software professionals as subjects (Jørgensen 2002).[2] Section 4.1 discusses differences between students and professionals. Section 4.2 discusses how to get professionals to take part in experiments.

4.1. Students versus Professionals

Most subjects in software engineering experiments are students. The main reason is that they are more accessible and easier to organise, and hiring them is generally inexpensive. Consequently, it is easier to run an experiment with students than with professionals and the risks are lower. Nevertheless, one should also use professionals in experiments because there may be many differences between students and professionals, for example regarding:

- experience and skill level,
- use of professional methods and tools, and
- team work versus individual work.

However, the discussion in the SE community about students versus professionals seems over-simplified. What do we mean by "student" and "professional"? For example, in two of our experiments, 40% of the students were either working part time during their study or had previously been working full time in industry before they started the current study (Anda & Sjøberg 2002). Should these students be classified as "semi-professionals"? More importantly, the variations among students and variations among professionals may be so large that whether the person is a student or a professional, may just be one of many characteristics of a software engineer. This is also indicated by most software effort estimation models, e.g. COCOMO, the members of a software development project are categorised according to several parameters.

The experiments we have conducted indicate that the variations among professionals are higher than the variations among students. For example, many professionals were employed in times were there was an extreme shortage of IT personnel. Consequently, some of them may have less formal qualifications than what is normally expected for software engineers. However, this indication of larger variations among professionals than among students may simply be a consequence of the fact that our student subjects are taken from a very homogenous group of students, that is, they belong to the same class or a couple of classes, which means that the topic studied and the year they study are basically the same for all the students. In the case of postgraduate students, they are at least from the same university. Generally, we expect more variation among professionals than among students due to more varied educational background, working experience, etc.

Generally, SE experiments that involve professionals seldom characterise the professionals' competence, experience and educational background, and the authors seldom justify to what extent their subjects are representative of the software engineers who usually perform such tasks. This leads to several problems:

- The results may not be trustworthy, that is, the professionals may not be realistic for the actual experimental tasks. The sample recruited may be biased in some way, for example, a company may only be willing to let their least experienced or least demanded software engineers take part in an experiment.
- Comparing results from the original with replicated studies is difficult.
- Successful transfer of the results into industrial practice is less likely.

To generalise from experiments with a given group of subjects, we would need information about the ability and the variations among the subjects and the group of people to which the results will be generalised. For professionals, depending on what we wish to study, it would be relevant to know the variations regarding competence, education, experience, age, nationality (?), etc. (Some of this information may be highly controversial.) In the OO design experiment we conducted, we collected detailed information about:

[2] Interestingly, studies on estimation in forecasting and human judgement research journals seem to have a *lower* proportion of experiments with professionals as subjects than the software development effort research studies (Jørgensen 2002). Increasing the realism of experiments is, therefore, probably not only an important issue in software engineering.

- age,
- education (number of credits in general, number of credits in computer science),
- general work experience,
- programming experience (OO in general, particular programming languages (Java, C++, etc.),
- knowledge of systems developments methods and tools, and
- subjective description of their own programming skills.

This background information can be used in several ways, for example, to determine

- the target population for which the results are valid, and
- to what extent the results of the treatments depend on the collected background information, e.g., that certain design principles might be easier to understand for experienced professionals than for novices.

In the OO design experiment, a representative sample of the population of programmers in the Norwegian IT industry (130 professionals from 9 consultancy companies) carried out exactly the same tasks. The background information collected, combined with the results of the experiment, will thus indicate the distribution of these variables for Norwegian programmers. This distribution can be used to create smaller, but still representative, samples in future experiments. Given that we know the proportion of programmers in various categories, we can stratify the sampling process. One of the main advantages of hiring consultants as subjects is that we then can specify the kinds of subjects we want to ensure representativeness (see Section 4.2).

It will be interesting to compare the individual variations in programming skills in our data with similar data collected by others. It would also be interesting to identify the variations within the same company versus between companies, variations between in-house professionals versus consultants, etc. For example, in-house software development in Nokia, Ericsson, Bosch, etc. may differ from development projects run by consultancy companies. Nevertheless, knowledge about the effect of a certain technology among consultants or even students may still be useful in the lack of knowledge of the effect of the technology in a company's own environment.

Regarding background information of students, it would be relevant to know the variations regarding level (undergraduate, postgraduate), subject, age, nationality (?), etc. It would also be interesting to identify the variations within the same university versus between universities. (Some of this information may also be highly controversial.)

Related to the student-professional discussion, is the degree of competence, i.e., novice versus expert, in the studied topic. For example, in a study on risk assessments, we found that if there is a lack of feedback on such assessments, even experienced software professionals may be at a novice level (Jørgensen 1995). Consequently, a naïve study of software professionals, without knowing about their competence level, makes it hard to generalize to other organizations.

A non-controversial use of student experiments is to use them to test experimental design and initial hypotheses, before conducting experiments with professionals, as recommended in (Tichy 2000). However, to get more knowledge about under what circumstances and to what degree students can be expected to perform similarly to software professionals, we plan to compare the performance of students with professionals in most of our experiments.

In particular, it would be interesting to compare students with professionals in both *absolute* and *relative* terms. The applicability of a technology may depend one the absolute performance, e.g., the size of the effect may be unsatisfactory among students but satisfactory among professionals. However, in some cases (when the size effect is considered unimportant) it may be sufficient to compare technologies on a relative basis, that is, one can use student experiments if the relative difference between two or more technologies will be the same independently of whether the subjects are students or professionals. In summary, more research is needed to find the appropriate balance between using students and professionals in experiments.

4.2. How to get Subjects?

Most university researchers also teach. They can then relatively easy use students as subjects in experiments. One can organise an experiment as follows:

1. The experiment is considered a compulsory part of a course, either as part of the teaching or as an exercise (Anda *et al*. 2001, Jørgensen *et al*. 2002).
2. The experiment is not compulsory; it is voluntary, but is still regarded relevant for the exam (Jørgensen & Sjøberg 2001). (In practice, students may feel obliged to take part to show their teacher that they are enthusiastic students.)
3. The students are paid, that is, the experiment is not considered as part of the course (but it may still be relevant) (Anda & Sjøberg 2001, Arisholm *et al*. 2001, Karahasanovic & Sjøberg 2001).

In our research group, we have experienced that alternative (3) is easiest to organise. One then does not have the time constraint of the ordinary classes, the students are motivated and there are no ethical problems regarding the difference in the technology being exposed to the students. (One might argue that it is unethical if some students have been using a technology that proved better than the technologies being used by others students, that is, some students have learned better technologies than other students.) We usually pay the students a fixed rate per hour (15 US$). We then get about 50% of the students of a class or group of students (e.g., MSc students). In the cases where we want a higher proportion or we need subjects on a short notice, we increase the honorarium.

One should be aware that it may be a methodological problem that the teacher is also the researcher, that is, the technology being subject of an experiment run by a given researcher is also being taught by the same researcher. Consequently, the students might be biased. This is avoided if professionals are used as subjects (unless the professionals happen to be the former students of the actual researcher).

The lack of professionals in software engineering experiments is due to the conception of high costs and large organisational effort. Warren Harrison (2000) puts it this way:

> Professional programmers are hard to come by and are very expensive. Thus, any study that uses more than a few professional programmers must be very well funded. Further, it is difficult to come by an adequate pool of professional developers in locations that do not have a significant software development industrial base. Even if we can somehow gather a sufficiently large group of professionals, the logistics of organizing the group into a set of experimental subjects can be daunting due to schedule and location issues.

To alleviate these problems, we have applied alternative incentives to conduct experiments with professionals:

- Offer the organisation tailored, internal courses and, for example, use the course exercises as experiments.
- Have a part time job in the company and advocate the experiment as useful for the company (Anda 2002).
- Involve some of the employees in the research and offer them co-authorship of the resulting research paper.
- Offer the organisation a network of people from other organisations with relevant experience.
- Pay the company directly for the hours spent on the experiment (Jørgensen & Sjøberg 2002).

The first and the last alternative have proved most successful. Regarding the last alternative, we thought that it would be most effective to use our personal network to get people to take part in our experiments on their spare time and pay them individually. However, it turned out that a much better approach is to ring the switchboard of a major consultancy company and request a specific service. For example, we contacted Accenture, Cap Gemini Ernst & Young, TietoEnator and six other consultancy companies in Norway and Sweden, and asked for 8-30 Java programmers for one day and one internal project leader to take part in our OO design experiment. We offered payment in the lower range of the current market price. (The fee per hour varied depending on whether the consultant was junior, intermediate or senior.) All the companies were positive. Some of them requested that as part of the deal, we should give an in-house seminar on the results after the experiment, which showed that they were also interested in the experiment itself, not only the money.

Note that this experiment was carried out in November and December 2002, which generally was a difficult time for IT consultancy companies. Our future experiments will show whether they will be as positive in periods with higher demand on their services. One should also note that it was considerably more difficult to get subjects to take part in the design pattern experiment (held in May 2002, which was also a difficult time for the software industry) because it was held for a period of three given days, whereas the length of the OO design experiment was only one day, and the day was chosen by the actual company itself (within a certain interval).

When we hire people from consultancy companies to take part in our experiments, we are treated much more professionally than when we work as Software Process Improvement researchers. We are like any ordinary customer (although several consultants have said that they find our projects more exciting than most other projects). We agree on a contract and they internally define a project with project leader, budget, etc. Of course, one must have the resources to do research this way. We therefore believe that empirical SE research departments should have particular budgets for paying students and software professionals for taking part in experiments.

5. Realistic Environment

A challenge when configuring an experimental environment is to provide an infrastructure of supporting technology (processes, methods, tools, etc.) that resembles an industrial development environment.

5.1. Experimental Procedure in a Realistic Environment

Many threats to external validity are caused by an artificial setting of the experiment. For example, because the logistics is simpler, a classroom is used instead of a usual work place. Conducting an experiment on the usual work site with professional development tools implies less control of the experiment than we would have in a classroom setting with pen and paper. Nevertheless, there are many challenges when conducting experiments with professionals in industry. From the OO design experiment, we learned the following lessons:

- Ask for a local project manager of the company who should select subjects according to the specification of the researchers, ensure that the subjects actually turn up, ensure that the necessary tools are installed on the PCs, and carry out all other logistics, accounting, etc.
- Motivate the experiment up-front: inform the subjects about the purpose of the experiment (at a general level) and the procedure (when to take lunch or breaks, that phone calls and other interruptions should be avoided, etc.).
- Ensure that the subjects do not talk with one another in breaks, lunch, etc.
- Ensure the subjects that the information about their performance is kept confidential (both within company and outside).
- Ensure the company that its general performance is kept confidential.
- Monitor the experiment, that is, be visible and accessible for questions.
- Give all the subjects a small training exercise to ensure that the PC and tool environment are working properly.
- Ensure the company and subjects that they will be informed about the results of the experiment (and do it).
- Provide a proper experiment support environment to help set up and monitor the experiment, and collect and manage the experimental data (see Section 5.2).

5.2. Experiment Supporting Technology

Our experience from the experiments we have run with both students and professionals is that the all the logistics around the experiments is work intensive and error prone: General information and specific task documents must be printed and distributed, personal information (bank account, etc.) and background information must be collected, all solution documents must be collected and then punched into an electronic form, etc. This may in turn lead to typing errors, lost data (Briand *et al*. 2001), etc.

We realised that if we were to scale up our experiments and particularly run experiments with professionals in industry using professional development tools, that is, make our experiments more realistic, we would need electronic tool support. Hence, we searched for suitable tools and found several Web tools developed to support surveys, most of them designed by psychologists (e-Experiment[3], PsychExperiments[4], Survey Pro 3[5], S-Ware WWW Survey Assistant[6], Wextor[7]). Those tools basically distribute questionnaires to the respondents who fill them in online. Then the results are stored in a local database or sent via emails to the researchers. However, to conduct the kind of experiments that we were interested in, we needed a more sophisticated tool. Therefore, in collaboration with a software company that develops solutions for Human Resource Management, we developed (and are still extending and improving) the Web-based Simula Experiment Support Environment (SESE). SESE is built on top of the company's standard commercial human resource management system. Figure 1 illustrates the way SESE supports an experiment:

Step 1: The researcher defines a new experiment (SESE can manage an arbitrary number of experiments simultaneously) with the required questionnaires, task descriptions, files to be down-loaded etc.

Step 2: The administrator creates a user-id and password for each person that will take part in the experiment, and emails that information to the person.

Step 3: The user (subject) fills in questionnaires (personal and background information) and downloads task descriptions and other required documents (design models, source code, etc.).

Step 4: The user carries out the tasks, answers questions along the way and uploads the finished documents. Timestamping is done continuously (when were the task descriptions downloaded and task solutions uploaded, when did a break start and stop, etc.).

Step 5: When a subject has finished the tasks, his or her results are stored in the (relational) database of SESE. When all the subjects have finished, the researcher can start analyse the data.

The OO design experiment was run at 10 different sites using SESE via Web. The experiences from using SESE are positive. SESE enables us to run distributed experiments – both in location and time – instead of only "big-bang" experiments. If acceptable from a methodological point of view, one should avoid "big-bang" experiments to reduce risks. For example, in our design pattern experiment, a fibre cable breakdown far beyond our control forced us to send 44 consultants home and defer the experiment to start on the next day. This accident caused a lot of frustration and a direct loss of 20 000 US$.

Moreover, the initial subjects participating in an experiment may be used to improve the formulation of the

[3] http://www-personal.umich.edu/~ederosia/e-exp/

[4] http://www.olemiss.edu/PsychExps/

[5] http://apian.com/survey/spspec.htm

[6] http://or.psychology.dal.ca/~wcs/hidden/home.html

[7] http://www.genpsylab.unizh.ch/wextor/index.html

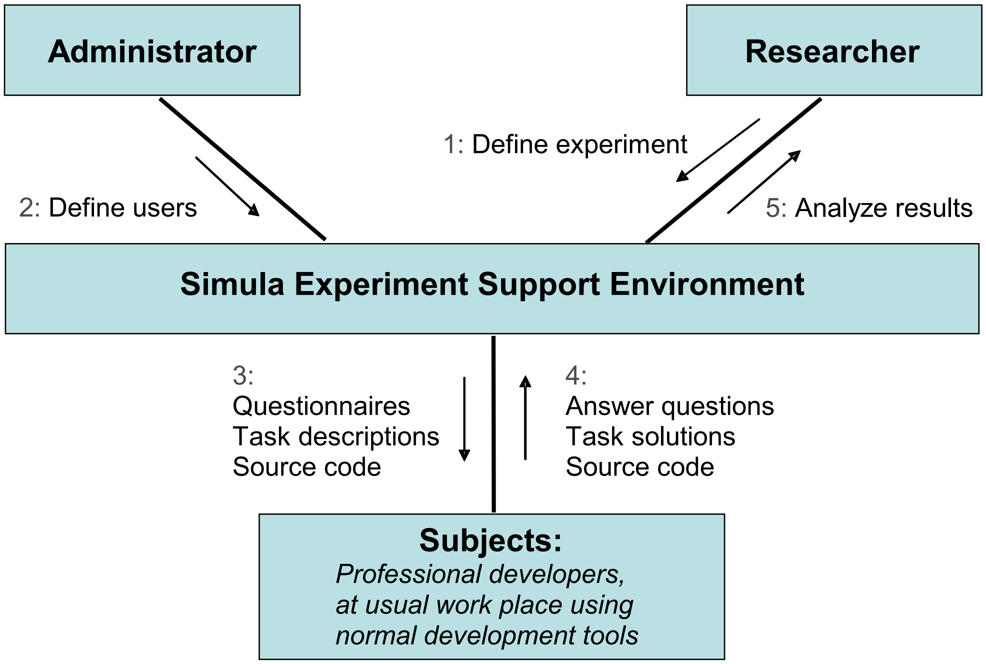

Figure 1. Web-based experiment support environment

hypotheses and decide the number of subjects needed. As such, the initial phase of the experiment may serve as a flexible pilot study – its extent and kind may be decided on the fly.

Future extensions of SESE may include detailed logging of the way a task is performed or a technology is used. This may include window operations, keystrokes, mouse operations and movements logged with timestamps. SESE and the experiences from using it are more fully described in (Arisholm *et al.* 2002).

6. Conclusions

This paper focused on the need for conducting more realistic experiments in software engineering. Using a large experiment on OO design alternatives and other experiments conducted by our research group as examples, we described how increased realism can be achieved, particularly along the dimensions tasks, subjects and environment. A Web-based experiment supporting tool was also described.

Increasing the realism of SE experiments also requires an increase in the resources needed to conduct such experiments. Using professionals as subjects usually means that they must be paid. Development of necessary supporting tools is costly. Attracting experts to take part in the design, management and data analysis of realistic experiments also requires resources.

However, our experience is that compared with personnel costs a relatively small amount of money may fund relatively large experiments. Even our OO design experiment with 130 professionals did not cost more than 70 000 US$, which is less than the cost of one post-doc for one year (including overhead). In the SE department of Simula Research Laboratory we are to a major extent allowed ourselves to prioritise how we spend our money. We have argued that we would rather spend money on experiments than employing the 10th person in the group. This may be more difficult in a university setting.

Nevertheless, we believe that few university research groups in empirical SE actually apply research funding bodies for money for carrying out experiments. (Neither did we when we were at the university.) Why should not empirical SE groups get funding to hire professionals to take part in experiments like other research groups get funding for buying super-computers, Linux clusters, etc.? Given the importance and challenges of the software industry (PITAC 1999), the empirical SE community should apply national and multi-national (e.g. EC) research bodies to fund realistic experiments.

Acknowledgements

We thank the anonymous referees for their constructive comments.

References

Anda, B. & Sjøberg, D.I.K. Towards an Inspection Technique for Use Case Models, SEKE'2002 (Fourteenth International Conference on Software Engineering and Knowledge Engineering), Ischia, Italy, July 15-19, 2002.

Anda, B. Comparing Effort Estimates Based on Use Case Points with Expert Estimates. In Empirical Assessment in Software Engineering (EASE 2002), Keele, UK, April 8-10, 2002.

Anda, B., Sjøberg, D.I.K. & Jørgensen, M. Quality and Understandability in Use Case Models. In J. Lindskov Knudsen (Ed.): ECOOP'2001 (Object-Oriented Programming, 15th European Conference), Budapest, Hungary, June 18-22, 2001, LNCS 2072 Springer-Verlag, pp. 402-428.

Arisholm, E. & Sjøberg, D. Towards a Framework for Empirical Assessment of Changeability Decay. *Journal of Systems and Software*, Vol. 53, pp. 3–14, Sep. 2000.

Arisholm, Erik, Sjøberg, Dag I.K., Carelius, Gunnar J. and Lindsjørn, Yngve. SESE – an Experiment Support Environment for Evaluating Software Engineering Technologies, NWPER'2000 (Tenth Nordic Workshop on Programming and Software Development Tools and Techniques), Copenhagen, Denmark, 18-20 August, 2002.

Arisholm, E., Sjøberg, D.I.K. & Jørgensen, M. Assessing the Changeability of two Object-Oriented Design Alternatives – a Controlled Experiment. *Empirical Software Engineering,* (6):231–277, Sep. 2001.

Aronson, E. & Carlsmith, J.M. Experimentation in Social Psychology. In G. Lindzey & E. Aronson (eds.), The Handbook of Social Psychology, Third Edition, Vol. 2, Reading Mass.: Addison-Wesley, pp. 1-79, 1968.

Basili, V.R. The Role of Experimentation in Software Engineering: Past, Current, and Future, Proceedings of the 18th International Conference on Software Engineering, Berlin, Germany, March 25-29, pp. 442-449, 1996.

Basili, Victor R., Rombach, Dieter & Selby, Richard. The Experimental Paradigm in Software Engineering. Experimental Engineering Issues: Critical Assessment and Future Directions, International Workshop, Dagstuhl, Germany, 1992, Springer Verlag, LNCS, No. 706, 1993.

Basili, Victor R., Selby, Richard & Hutchens, David. Experimentation in Software Engineering. *IEEE Transactions on Software Engineering* (invited paper), July 1986.

Briand, L.C., Bunse, C. & Daly, J.W. A Controlled Experiment for Evaluating Quality Guidelines on the Maintainability of Object-Oriented Designs, *IEEE Transactions on Software Engineering*, Vol. 27, No. 6, pp. 513–530, 2001.

Glass, R.L. The Software-Research Crisis, *IEEE Software*, Vol. 11, No. 6, pp. 42-47, 1994.

Harrison, W. N = 1: An Alternative for Software Engineering Research?, Beg, Borrow, or Steal: Using Multidisciplinary Approaches in Empirical Software Engineering Research, Workshop, 5 June, 2000 at 22nd International Conference on Software Engineering (ICSE), Limerick, Ireland, 2000.

Höst, M., Regnell, B., & Wohlin, C. Using Students as Subjects – A Comparative Study of Students and Professionals in Lead-Time Impact Assessment. *Empirical Software Engineering,* Vol. 5, No. 3, pp. 201-214, 2000.

Jørgensen, M. An Empirical Study of Software Maintenance Tasks. *Journal of Software Maintenance*, Vol. 7, pp. 27-48, 1995.

Jørgensen, M. and D. I. K. Sjøberg. Impact of software effort estimation on software work. *Journal of Information and Software Technology.* Vol. 43, pp. 939-948, 2001.

Jørgensen, M. and D. I. K. Sjøberg. The Impact of Customer Expectation on Software Development Effort Estimates. Submitted to *Journal of Empirical Software Engineering*. 2002.

Jørgensen, M. Cost and Effort Estimation of Software Development Work: A Review and an Annotated Bibliography, 2002 (submitted for publication).

Jørgensen, M., U. Indahl and D. Sjøberg. Software effort estimation and regression toward the mean. Accepted for publication in *Journal of Systems and Software*, 2002.

Jørgensen, M; Bygdås, S. An Empirical Study of the Correlation between Development Efficiency and Software Development Tools. *Technical Journal of Norwegian Telecom,* Vol. 11, pp. 54-62, 1999.

Karahasanovic, A. Supporting Application Consistency in Evolving Object-Oriented Systems by Impact Analysis and Visualisation, PhD Thesis, Department of Informatics, University of Oslo, 2002.

Karahasanovic, A., Sjøberg, D. Visualizing Impacts of Database Schema Changes – A Controlled Experiment, In 2001 IEEE Symposium on Visual/Multimedia Approaches to Programming and Software Engineering, Stresa, Italy, September 5-7, 2001, pp 358-365, IEEE Computer Society.

Laitenberger, O., Atkinson, C., Schlich, M. & El Emam, K. An experimental comparison of reading techniques for defect detection in UML design documents. *Journal of Systems and Software*, Vol. 53(2), pp. 183-204, 2000.

PITAC report, http://www.ccic.gov/ac/report/, 1999.

Porter, A. & Votta, L. Comparing Detection Methods for Software Requirements Inspections: A Replication Using Professional Subjects. *Empirical Software Engineering*, Vol. 3, No. 4, pp. 355-379. 1998.

Potts, C. Software-Engineering Research Revisited, *IEEE Software*, Vol. 10, No. 5, pp. 19-28, 1993.

Prechelt, L., Unger, B., Tichy, W. F., Brössler, P. &Votta, L. G. A Controlled Experiment in Maintenance Comparing Design Patterns to Simpler Solutions. Accepted for *IEEE Transactions on Software Engineering* in September 2000, to appear.

Rogers, E.M. *Diffusion of Innovations*, Fourth Edition, New York: The Free Press, 1995.

Rombach *et al. Experimental Software Engineering Issues: Critical Assessment and Future Directions*, Dagstuhl Workshop, Germany, September, 1992, LNCS 706, Springer Verlag, 1993.

Seaman, C. & Basili, V. An Empirical Study of Communication in Code Inspection, Proceedings of the 19th International Conference on Software Engineering (ICSE-19), May 17–24, 1997.

Tichy, W.F. Should Computer Scientists Experiment More? 16 Reasons to Avoid Experimentation, *IEEE Computer* Vol. 31, No. 5, pp. 32-40, May 1998.

Tichy, W.F. Hints for Reviewing Empirical Work in Software Engineering. *Empirical Software Engineering*, Vol. 5, No. 4, pp. 309–312, 2000.

Zelkowitz, M.V. & Wallace, D.R. Experimental Models for Validating Technology, *IEEE Computer*, Vol. 31, No. 5; pp. 23-31, May 1998.

Zhang, Z., Basili, V. & Shneiderman, B. An Empirical Study of Perspective-based Usability Inspection. Human Factors and Ergonomics Society Annual Meeting, Chicago, Oct. 1998.