An experiential report on the limitations of experimentation as a means of empirically investigating software practitioners

Chris Exton, Gabriela Avram, Jim Buckley, Andrew LeGear

{Chris.Exton, Gabriela.Avram, Jim.Buckley, Andrew.LeGear, @ul.ie},
University of Limerick, Computer Science Systems Department, Limerick Ireland and
Lero, University of Limerick, Ireland

Abstract. This paper outlines the needs for careful empirical-design choices during the study of software practitioners. It does this by presenting a documented, but unpublished, in-vivo, empirical, group study. The study was initially conceived as an experiment but was subsequently overwhelmed by human and other factors. As a consequence, only more observational comments could be derived from the study. In this paper, the study is analyzed and discussed, as a means of illustrating the conflict that often exists between in-vivo empirical studies and the experimental paradigm.

1 Introduction

All understanding and efforts to gain understanding exist in a tradition: a set of related and harmonious assumptions about what exists and the way to gain knowledge about it. Many researchers studying Software Engineering (SE) come from a computer science tradition. This tradition has a strong affiliation with cognitive psychology through Turing's [21] central influence. The cognitive psychology perspective, with its use of information processing models and scientific idioms, provides an island of familiarity for computer science researchers when they branch into empirical studies. However, closely associated with this affiliation, comes the acceptance of experimentation as a valid means of gaining insight into what participants are doing at a cognitive level.

One of the major reasons for empirical studies in SE is to identify and inform on the practice of SE [16]. While accepting the essential role of evidence in deepening our understanding of such SE practice, several studies show that there is a tendency in empirical software engineering to value quantitative evidence over qualitative [8,19,16]. The assumption that all constructs of interest "must have observable features that we can measure, although imperfectly" [8] often serves to ignore psychological factors such as social dynamics and organizational environments that have a strong influence on the software engineering practice. Thus they are inappropriate and unrealistic [15].

Moving away from strictly controlled laboratory experiments, several researchers are now focusing on real work environments and observing practitioners at work [10,7,22]. They try to understand the software engineers' day-to-day problems and to devise methods and tools that could help them solve these problems. Another trend coming to the fore in empirical software engineering research is to pay more and more attention to "the central role of human behaviour in software development" [19].

In this paper, we review the findings of an unpublished experiment that took place in a large computer company in Dublin in 2006. The original intent of this study was to use an architecture recovery technique called Reflexion Modelling as a collaborative learning support for a group of "software immigrants"[17] who were beginning to familiarize themselves with one of the company's large, commercial, proprietary software systems. Reflexion modelling had been used successfully in the organization previously as an aid to individual software engineers and thus, the company was interested in broadening their adoption of the technique. However this was the first time the tool would have been used in a collaborative context.

In this paper, we use our experiences of this experimental study to highlight the difficulties associated with a purely experimental approach in the empirical study of collaborative software engineering. These include subtle social and organizational dynamics that could easily be overlooked in a purely quantitative analysis. They also include other in-vivo considerations that impacted on the potential validity of our findings.

1.1 Use of Experiments in Software Engineering research

One of the most challenging aspects of research in Software Engineering is the creation, adaptation and use of suitable empirical research methods and sources of information that facilitate the comparison and evaluation of different Software Engineering practices, tools and techniques. Software Engineering research is still very much in its infancy and, as such, still has very much to learn in terms of building a wide selection of methods from which sound theories can be developed. Although it would make our life as researchers straightforward, it is improbable that any mono-dimensional attempts, could hope to accommodate the complexity of the multiple interacting influences between the software, the software engineering processes, software engineers and the wider environment. Given this complexity, the choice of research methods with which to study these interactions needs to vary according to the circumstances of each study. However computer scientists often seem to have a disposition to default to experimentation as the predominant empirical research method[1]. For example Sjoberg et al [18], in their review of the pervasiveness of experimentation in empirical SE, found 103 experiments in software engineering published in journals and conference proceedings between 1993-2002. The experiments presented involved 207 scholars

[1] Fenton et al [3] define a formal experiment as "a rigorous, controlled investigation of an activity, where key factors are identified and manipulated to document their effects on the outcome." In their opinion, formal experiments, together with case studies and surveys, are among the key components of empirical investigation in software engineering [3].

from 109 institutions in 19 countries, illustrating how popular this type of empirical study is in software engineering. The choice of a research method should instead be guided by the actual research question, the type of evidence needed, and the intended audience for the evidence.

Observation of collaborative work practices in SE can and should successfully complement quantitative methods like experiments, according to O'Brien et al. [15]. Observation focuses mainly on people's behaviour and discourse, paying attention to all details. The role of the researcher can range from complete observer (having no contact with those he is observing) to participant-observer (when the researcher actually gets involved and plays a role in the event). Observation might involve note taking, audio and/or video-recording and even interviews in order to produce a rich description of what happened in the setting. The purpose is "to gain an understanding about what people do, why they do it and what meaning they assign to activities" [14].

2 The Empirical Study

Over the course of 2005 and 2006 researchers from the University of Limerick, in collaboration with a large Dublin-based, computer company carried out a number of in-vivo empirical studies of software engineering. These studies evaluated Reflexion Modelling as a tool to support architecture recovery and architecture conformance in large commercial software systems. (Reflexion Modelling is described in detail in the paper: "ESCAPE Meta Modelling in Software Engineering: When Premature Commitment is Useful in Representations", which is also to be presented at this workshop).

In the empirical study described here, software developers were tasked with familiarizing themselves with the software system and were already working, semi-independently, towards this goal, when the idea of this study arose. The architects responsible for the system were committed to having the software developers perform Reflexion Modelling as soon as possible to allow them leverage the greatest possible (architectural) understanding from the approach, before they had developed it independently. Hence the experiment had to be designed and performed with only one weeks notice.

However, despite the short preparation time, the potential research value appeared significant. Firstly, it would advance our case-study work, allowing us to apply Reflexion Modelling to a population, and thus introduced the possibility of generalizing the results using statistics. Secondly, Reflexion Modelling had never before been considered as a collaborative tool. This study allowed us the opportunity to study it in this new and potentially interesting context.

The experiment was designed immediately, with one constraint specified by the company: that the context of the experiment was in-vivo with respect to the subject system and with respect to the participants participating in situ. This was seen by the researchers involved as beneficial, heightening the ecological validity of the study. The design of the proposed experiment is now discussed in more detail.

2.2 Design of the Experiment

As described above, an experimental procedure was designed to try and gain an insight into how a group of software engineers could collaboratively use Reflexion Modelling as a tool to further their understanding of a specific software system. Specifically, the aim of the study was to address several research questions, most of which can be subsumed by the question: *Are there advantages to group design recovery mediated through Reflexion Modelling?*

The component hypotheses were:

1. Do participants aggregate to a more consistent understanding of the system through group design recovery (mediated through Reflexion Modelling)?
2. Do participants become more aware of other participants' perspectives on the system through group design recovery (mediated through Reflexion Modelling)?
3. Does participant role (architect, software immigrant) impact on the amount of change a participant's architectural model undergoes as a result of group design recovery (mediated through Reflexion Modelling)? Specifically, do the models generated by participants with more senior, architect-type roles change less as a result of the group's design recovery exercise.

If the answer to all three were yes, then we would have evidence that, during this form of group design recovery, more senior members of the group generally influence the architectural model formed by more junior members of the team (although we anticipated that it may also work in reverse occasionally, especially if a more junior member of the team is focused on the specific area being modelled.)

Thus the experiment was designed in 3 phases. In the first phase the participants (new developers and existing architects of the system) were to independently create their own Reflexion Models of part of the system. These models would be gathered by the researchers. Then the participants would collaborate together in three teams to form team-models of the system. These collaboration meetings would be observed by the researchers and the resultant Reflexion Models gathered. Finally, the participants would revisit their individual Reflexion models in the light of their collaboration and refine them. By comparing the various Reflexion Models produced, the researchers could address the research questions.

3 Realization of the Experiment

3.1 Phase 1: using Reflexion Modelling individually

Initially 18 developers and 3 architects had signed up as participants. A standard tutorial was given to each of them individually, at their desk, to familiarize them with both the Reflexion Modelling approach and the associated software. The participants then had 2 weeks to develop their individual Reflexion Model of the system under study, focusing on its ‘search’ utility. They were asked to do this in their spare time but they were made aware that they were doing this at the bequest of their team leaders and that they would be required to present their architectural model to a system’s architect in a group meeting after this time. Participants were asked to develop their model in isolation. However, given the open-plan work environment of the company and the presence of electronic communication, we cannot guarantee that this was entirely their mode of operation.

Participants were offered email support for their efforts but only one participant chose to email us over the following 2 weeks. Additionally, one of the research team went up to the company after the first week, and visited all the participants to ensure that they were progressing and to help them circumvent any bottleneck or problems they were having with the process. During this visit, the researcher fixed 2 problems encountered by different participants.

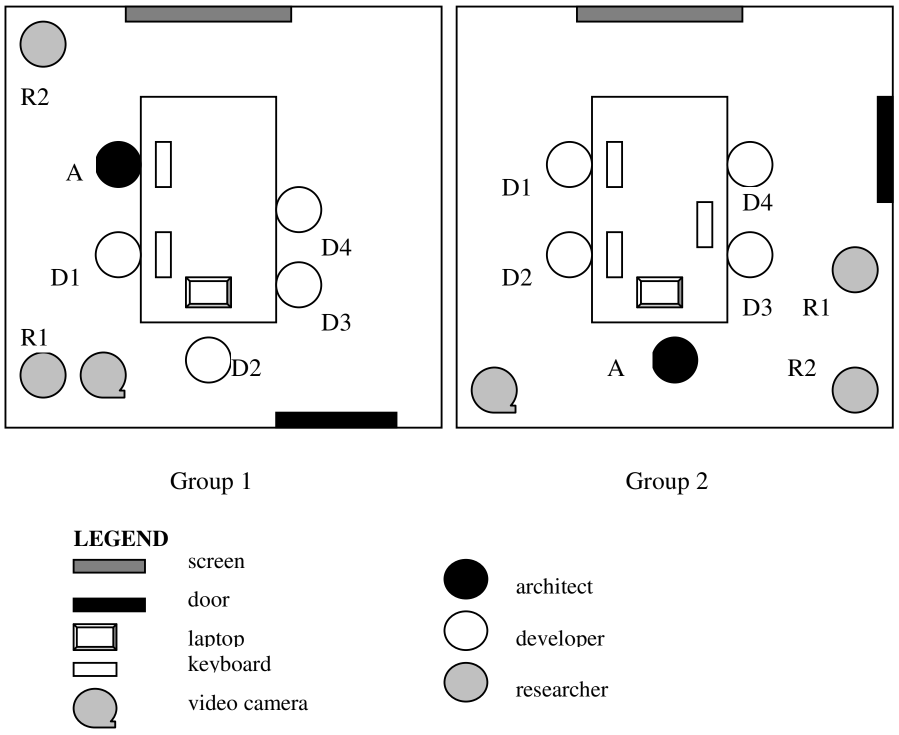
3.2 Phase 2: collaborative Reflexion Modelling

On the morning of the experiment, only 8 developers and two architects were still available. Unfortunately, it was impossible to control who would be involved in the experiment: the participants volunteered to do so, but some were subsequently prevented from participation by various other work commitments that arose over the initial 2 weeks.

Given the fall-off in numbers, the researchers decided to re-organize the collaborative meetings into two groups, each made of one software architect and four developers (the original plan was to have 3 teams with six developers each). In each room, there was one researcher who lead the experiment and a second one observing its development. Each group made use of a laptop and two wireless keyboards; the image on the screen (the Reflexion Model) was shared on a big screen with the help of a projector.

Group 1 was dominated by the software architect, who took the opportunity to teach the developers what he knew about the application under scrutiny. The interventions of researchers were minimal (as intended) while the developers’ interventions were few, and mainly aimed at clarifying issues related to the use of the supportive tool provided.

Group 2 worked more collaboratively and the architect tried to act as a coach, refraining from imposing his own vision. There were extensive discussions between the participants and the researchers about the issues encountered when using the tool, as the tool was a prototype only and required large processing power when applied to a commercial software system, often causing their machine to pause for several minutes



The layout of the room for each of the 2 groups is presented in the figure above. We will now highlight some of the participant-based differences between the 2 groups that suggested the limited utility of a controlled experiment in this case. We will start with the architect.

The architect in each team was supposed to contribute an initial opinion on the individual Reflexion Models produced by the developers, and then to become one of the participants, asking relevant questions, commenting and giving hints when necessary.

In **Group 1**, the architect dominated the session, with the result that a sort of 'classroom' atmosphere was created, where he 'taught' the others. When asked to comment on the individual models, he emphasized where they had deviated from the actuality of the system. He was the controlling party, and did all the changes to the model while commenting on them. In the end, he commented:

"As kind of senior here, I'd like to come up with some ideas about where the misfits were"

These words illustrated the role he felt he should play. Even if he didn't take a central seat, and used a keyboard instead of the laptop itself, he took over the meeting, speaking for most of the time, initiating and creating his own changes on the model.

From a spatial perspective, he shared his attention between the keyboard and the screen, seldom making visual contact with the others.

In **Group 2,** the architect played more of a mediating role. He stood in the central seat and operated the changes on the laptop, but most of the time he waited for the changes to be suggested by the others. He gave his opinion whenever asked to, but he asked the others not to take his point of view as necessarily correct, emphasizing that he was no expert on that part of the system and he might be wrong. When commenting on the initial individual models, he found good parts in them all, and similarities with his own model. He did much less talking that the architect in group 1 but seemed particularly concerned about the influence he had on the final model:

> "It's so peculiar, really – it looks very much like my own model! But I assume I'm influencing things too much here!"

He empathized with the others about his acquired experience in working with the prototype tool:

> "See? That's exactly what happened to me…and the only solution is to take every class and map it…"

He also encouraged a relaxed atmosphere, by making remarks like:

> "It reminds me of the map of a ski resort in Switzerland I go to every year!"

As regards **the software developers**, they were supposed to take an active role, and to suggest and operate changes to the model themselves. In addition, they were expected to comment on the changes suggested by the others. The final model was supposed to be a result of the iterative interactions within each group.

In **Group 1**, the developers' interventions were minimal. In the beginning, each immigrant had to comment on his own model, but after that, their interventions were limited to asking questions of the architect, or responding to the researchers' questions:

"What are we focusing on? On this model or on the search function in general?"

("Did you actually find it (the tool) very slow?) It's terrible. Not only that it's slow, but it also blocks everything else that's going on on your machine")

In the first 30 minutes, there were a few attempts to suggest various alternatives. None of them was accepted by the architect. In the second half of the experiment, all of them became passive with respect to modelling the system.

In **Group 2**, the developers' interventions were received with enthusiasm, discussed by the architect and by the other developers and put into practice if accepted by the group. Because of the long time required by model recalculation, a batch of changes was suggested, discussed and sometimes put into practice before every model recalculation. This inconvenience of the tool made it difficult to see the direct effects of every change made to the model. Developer1 and Developer3 tried to use the keyboards for making changes, and for a while, Developer1 was the one operating the changes instead of the architect. Developer4 made a number of pertinent suggestions and commented extensively on their rationale. Developer2 played a less active role, having to attend another meeting for 30 minutes in the middle of the experiment.

3.3 Phase 3: the new individual models

Initially, our intention was to ask the individual developers to review their architectural Reflexion Model in the light of their group session. Consequently, participants were asked 'if they could review their individual model' over the following 2 weeks. Unfortunately, this weak phrasing of the request, coupled with work pressures meant that none of the programmers did so. Indeed, given the divergence between the 2 group sessions, the research team decided that the data would be of limited value in evaluating their research questions and so this data was not collected.

4 The Postmortem

It became apparent that we had to examine the study on a number of fronts. Firstly there were several operation issues which had occurred that could have been avoided given proper preparation. For example, some of the keyboards didn't work and the researcher had to leave one of the rooms to ask there colleagues about specific characteristics of the tool (stopping proceedings). Also, the long waiting times for building the model was a cause of frustration to the participants, although this was less avoidable without significant overhead.

We felt however that, apart from these operational issues, there were more fundamental and interesting observation as to what actually had occurred in relation to the dependent and independent variables we had hoped to measure between the two groups. From an experimental perspective, the two groups were similar in nature (similar makeup) and the dependent variables were the same. Consequently, we could have expected to see broadly similar characteristics in terms of the groups' interactions and outcomes. This was not the case.

Instead, it was obvious that unconsidered human factors were at play and that these had had an extremely large and differentiating effect on the two experimental groups we had observed. The interaction between the architect and developers was significantly different in both groups. One was collaborative whereas the other was a largely passive experience for the developers.

If we really wanted to understand why the two groups had functioned so differently it would have been more appropriate to focus on what the participants had said, how they had said it and how this discourse had effected the roles each participant played within there group. For the dedicated experimenter it may seem at first that the obvious solution was to tighten the level of control by scripting the architect's discourse -

but the effect of this course of action would have been to significantly decrease the ecological validity or real-world similarity we hoped to maintain.

The next action for the dedicated experimenter might have been to simply introduce these factors as independent variables. But again, we had to question to what extent we could quantitatively measure these human characteristics (variables such as the 'use of authority' and 'use of humour' and how they influenced the junior participants by empowering or dis-empowering them.). Yet these characteristics of the sessions obviously impacted our study to a great extent.

A third alternative would have been to use the same architect in both sessions. However, it is likely in this instance that such a strategy would have also resulted in 2 different sessions for the developers as the experience of the architect in his first session would have introduced learning factors into the second session.

Given our strategy, it became apparent that the interaction we had observed, and resultant output, was fundamentally different between the two groups. Due to the large magnitude of difference in behaviour we observed between the two groups, we felt that any of the quantitative measurements that we initially proposed would have little value in terms of the initial research questions, and any statistical significance that may have been reported by us would be ambiguous at best - as it was unclear if the variables we were measuring were the core causal factors.

Another issue was the lack of control we researchers had in studying realistic in-vivo behaviour. For example, we had no way of insuring that the participants did create their initial Reflexion model in isolation (although, given the diversity in their original models, individual work was likely for the most part). Likewise we could not stop a participant from leaving the collaborative meeting in phase 2, for another meeting of higher priority. Yet critically, these are factors that come into play in real Software Engineering practice.

Finally, while we started off with a relatively large cohort, the in-vivo nature of the study resulted in a large fall-off in the number or participants willing to proceed to phase 2. Even with this fall-off we still retained 8 participants, a number suitable for some non-parametric, repeat-measure statistical techniques. However, in the majority of in-vivo empirical studies, it is likely that the number of participants would be smaller still and unsuited to statistical analysis.

On a positive note it was felt that the experience we obtained in this study should be documented as it may provide insights into some of the practical pitfalls in performing experimentation in a real-world industrial setting. In addition, our documented and pragmatic experience may feed into a number of current debates that relate to the use and values of qualitative and quantitative research in the domain of software engineering.

5 Discussion

Given the original experimental design, it is fair to state that our own perspective was firmly rooted in the experimental tradition and that we were primarily concerned about presenting our findings in a quantitative manner. Our intent was to produce credible and authoritative knowledge about the use of Reflexion Modelling as a means of gaining understanding of software in the social setting of a design team. It was an experiment designed to take place in a real work setting, with real software engineers trying to solve a real software-familiarization problem.

Our use of the experimental method provided a well-recognized approach that would have enabled us to present our findings as credible. The experimental paradigm, as the primary basis of gaining knowledge, needs a controlled environment with identifiable variables. Although quantifiable variables can be identified for social interactions, we should ask ourselves the true value of these variables. Given the individuality of participants[23], are these truly useful or merely a fudge that enables our use of experimentation in an otherwise impossible situation?

In many cases it may be more appropriate if we simply accepted that too many factors are at play and defy quantification, and that other augmenting methods should be considered. In addition, the very creation of a controlled environment may destroy telling interactions that would serve to inform on the practice of software engineering.

Thus, in many cases, the knowledge we gain from controlled experiments gives us insights into how programmers behave in experimental situations, but cannot be used with any validity outside that particular context. It is important that, like researchers from the qualitative social psychology domain, we recognize that issues such as "situational knowledge" [24] and "power relations" do exist and they directly affect all aspects of software engineering from analysis through design to debugging. (Where situational knowledge can be defined as knowledge that is embedded in language, culture, or traditions, and power relations consider the relationships between individuals within the study and how they are used.)

It is essential that we accept that experimentation can be stretched beyond its useful capacity in some situations and that it should augmented by confidence-building qualitative methods, or even displaced by such methods. This is of course not without loss, as experimentation has and still does provide a well accepted, tested and understood means of knowledge creation. Although we can argue that there must be a more appropriate means of gaining knowledge than experimentation for many in situ industrial based studies, it is still unclear as to what qualitative methods are the best approach for investigations of social settings in software engineering.

6 Conclusion

Statistics are based on the quantitative findings of a study. They depend on the need to reduce the scenario to a number of quantitative observations. This reductionist approach is a dual edged sword as, on one hand, it is intended to ascertain the relationship between the core elements or variables. However if it simultaneously jettisons elements that may directly affect the outcome due to their inability to be measured in a quantitative manner, it renders the statistics meaningless. From our experienced viewpoint the intrinsic properties of social interaction, which constituted a key element of this study were not quantifiable in any meaningful manner.

Hence, it is important that we continuously question if the necessary assumptions for the experimental method to operate undermine the validity of the knowledge discovered. We should not simply assume that the sacrifice of considerations such as "situational knowledge" and "power relations" are acceptable, as they are real components of many software engineering situations and the studies that observe these practices. Hence, they often affect what we are trying to observe.

Our analysis of this unsuccessful experiment is meant to shine a light on the need to carefully choose the most appropriated research methods for each situation. Even if our experiment did not succeed, the reflection triggered by this case can be considered worthwhile in itself, serving to improve the quality of future studies.

7 Acknowledgments

The authors would like to acknowledge the contributions made by research assistant Jacek Rosik and Professor Liam Bannon to this work. Lero is supported by Science Foundation Ireland (under grant no. 03/CE2/I303_1).

8 References

1. Basili, V. (1996) The Role of Experimentation in Software Engineering - http://ieeexplore.ieee.org/iel2/3540/10631/00493439.pdf?tp=&arnumber=493439&isnumber=10631
2. Christl A, Koschke R, Storey M-AD. (2005) Equipping the Reflexion Method with Automated Clustering. Working Conference on Reverse Engineering 2005; pages 89-98
3. Fenton, Pfleeger, Glass (1994) - Science and Substance: a challenge to software engineers- http://ieeexplore.ieee.org/iel1/52/7423/00300094.pdf?tp=&arnumber=300094&isnumber=7423
4. Hassan A. and Holt R. (2004) Using Development History Sticky Notes to Understand Software Architecture. Proceedings of the 12th International Workshop on Program Comprehension. pp 183-193.
5. Johnson, G. (1998) Collaborative Visualization 101, ACM SIGGRAPH – Computer Graphics, pages 8-11, vol 32 number 2 May 1998
6. Koschke R, Simon D. (2003) Hierarchical Reflexion Models. Working Conference on Reverse Engineering.
7. Ko A.J., DeLine R. Venolia G. (2007) Information Needs of Co-located Software Development Teams. International Conference on Software Engineering. May 2007 (To appear).
8. Lanubile,F. (1997) 'Empirical evaluation of software maintenance technologies', Empirical Software Engineering, 2, 1997, pp 97-108.
9. Le Gear A., Buckley J. (2005) Reengineering Towards Components with "Reconn-exion." ESEC/FSE Doctoral Symposium.
10. Lethbridge T. and Singer J. (2003). How Software Engineers use Documentation: The State of the Practice". IEEE Software. Vol 20. No 6. pp 35-39.
11. Murphy, G. (2003), 'jRMTool Reflexion Modelling eclipse plug-in.', http://www.cs.ubc.ca/murphy/jRMTool/doc/ [Accessed December 2003].

12. Murphy GC, Notkin D, Sullivan K. (1995) Software Reflexion Models: bridging the gap between source and high-level models. Symposium on the Foundations of Software Engineering. pages 18-28
13. Murphy GC,Notkin D. (1997) Reengineering with Reflexion Models: a case study IEEE Computer. 2(17):29-36
14. Oates, B.J. (2006) 'Researching Information Systems and Computing' Sage Publications: London; Thousand Oaks (see www.sagepub.com)
15. O'Brien, M. P., Buckley, J., and Exton, C. (2005). Empirically Studying Software Practitioners " Bridging the Gap between Theory and Practice. In *Proceedings of the 21st IEEE international Conference on Software Maintenance (Icsm'05) - Volume 00* (September 25 -30). ICSM. IEEE Computer Society, Washington.
16. Segal, Judith (2003) *The Nature of evidence in empirical software engineering.* In: 11th Annual International Workshop on Software Technology and Engineering Practice, 19-21 Sep 2003, Amsterdam, The Netherlands.
17. Sim, S.E., Holt, R. C.(1998*) The Ramp-Up Problem in Software Projects: A Case Study of How Software Immigrants Naturalize*, Proceedings of the Twentieth International Conference on Software Engineering, pp. 361-370, Kyoto, Japan, 19-25 April, 1998
18. Sjoberg et al (2005) - A Survey of Controlled Experiments in SE-http://csdl2.computer.org/persagen/DLAbsToc.jsp?resourcePath=/dl/trans/ts/&toc=comp/trans/ts/2005/09/e9toc.xml&DOI=10.1109/TSE.2005.97
19. Seaman, C(1999) "Methods in empirical studies of software engineering', *IEEE Transactions on Software Engineering*, 25(4), 1999, pp.557-572.
20. Shadish, W.R., Cook T.D., Campbell D.T. (2002) *Experimental and Quasi-Experimental Design for Generalized Causal Inference,* Boston:Hought-on-Mifflin
21. Turing, A. M. (1950) ‘Computing Machinery and Intelligence’, Mind 59:pp. 433-460.
22. Von Mayrhauser A. Vans A.M. (1995) Industrial Experience with an Integrated Code Comprehension Model” IEEE Software Engineering. Vol 10. No. 5.
23. Prechelt L. (1999). "The 28:1 Grant/Sackman legend is misleading" Technical Report 1999-18, 25 pages, Universität Karlsruhe, Fakultät für Informatik, Germany.
24. Brown, J. S., Collins. A., and Duguid, P., (1989), “Situated Cognition and the Culture of Learning” Educational Researcher; v18 n1, pp. 32-42, Jan-Feb 1989.