A Systematic Mapping Study on Practical Approaches to Teaching Software Engineering

Maíra R. Marques
Computer Science Department
Universidad de Chile
Santiago, Chile
mmarques@dcc.uchile.cl

Alcides Quispe
Computer Science Department
Universidad de Chile
Santiago, Chile
aquispe@dcc.uchile.cl

Sergio F. Ochoa
Computer Science Department
Universidad de Chile
Santiago, Chile
sochoa@dcc.uchile.cl

***Abstract*— Background: Software engineering is a core subject in computing education. Today, there seems to be a consensus that teaching software engineering requires students to perform practical experiences that simulate the work in the software industry. This represents a challenge for universities and instructors, because these experiences are complex to setup and involve considerable time and effort. Although there are several experiences and proposals reported in the literature, there is no clear solution to address this challenge. Aim: Being knowledgeable about the several approaches reported in the literature for dealing with this challenge is the first step to proposing a new solution. Counting on this knowledge allows instructors to reuse lessons learned from other universities. In order to address this challenge, we conducted a systematic mapping study that intends to answer the following questions: What are the main approaches used to address the practical experiences in software engineering education? Is there an emerging tendency to address this challenge? Which software process models are used to support the practical experiences in software engineering courses? Have the universities changed the way of conducting these experiences over the years? What are the main forums to seek information on practical approaches for teaching software engineering? Method: We used a systematic mapping study to identify and classify available research papers that report the use of practical experiences in software engineering education.**

Results: There were 173 papers selected, analyzed and classified. The results indicate that universities have realized the value of including practical experiences as part of the software engineering teaching process. However, few proposals indicate how to address that challenge. The practical approaches identified in this study were game learning, case studies, simulation, inverted classrooms, maintenance projects, service learning, and open source development. Only one recent report on the use of traditional approaches (i.e., teaching using expositive lectures) was found. The use of a development process to support these practical experiences seems not to be a concern for software engineering instructors. Only 40% of these studies report the use of a development process to guide the process experience. The reported processes are mainly agile methods. Conferences are the most used forum to publish studies in this area (72%). One third of these studies have been published over the last five years.

Conclusion: There is a clear concern for teaching software engineering involving practical experiences, and there are several initiatives exploring how to do it. The map gives us an overview of the different proposals to address this challenge, and also allows us to make some preliminary conclusions about the preferred approaches.

Keywords—software engineering education; systematic mapping.

I. INTRODUCTION

Software engineering is a discipline designed to apply systematic, disciplined, quantifiable approaches to the development, operation and maintenance of software [1]. As any engineering discipline, it is tightly linked to the industry. Consequently, there is a need to prepare students for the world of industrial software development [2].

Preparing software engineers for the industry is a complex task that most universities are striving to accomplish. Such a task must not only transfer knowledge on core-computing concepts, but also allow students to become lifelong learners who are able to keep pace with innovations in the discipline [3]. The approaches to teaching the theoretical aspects of software engineering have been well supported by the universities. However, software engineering needs a more practical teaching approach than just expositive classes.

Moore and Potts [4] state "*Software engineering cannot be taught exclusively in the classroom. Because software engineering is a competence, not just a body of knowledge, any presentation of principles and experience that is not backed up by active and regular participation by the students in real projects is sure to miss the essence of what the student needs to learn*". Practice-based software engineering seems to be the best instructional approach to deal with the contemporary software engineering education [5]. Unfortunately, not all computer science and software engineering programs offer experimental courses on software development that allow students to participate in a real life software development experience.

Integrating these experiences into the courses is a huge challenge for universities. This demands significant effort in terms of time and human resources to support the instructional process, such as utilizing coaches). These experiences also need instructors with expertise in software development, and they frequently require real clients that act as counterparts in

978-1-4799-3922-0/14/$31.00 ©2014 IEEE

these projects. These restrictions make this challenge difficult, if not impossible to address for most universities.

There is currently no predominant approach, nor right or wrong way to conduct these practical experiences, but there are certainly approaches that are more effective than others depending on the teaching context. For these reasons we have conducted a systematic mapping study to determine which approaches are available, and which of them seem to be preferred or are more effective. Our study was conducted following the procedure proposed by Petersen et al. [6]. Systematic mapping studies are intended to give an overview of a specific topic; in our case, it is the practical approaches for teaching software engineering. A systematic mapping study is typically classified as a secondary study and indicates the quality and quantity of work already done by the research community in the area. The papers used in the mapping study are selected and classified according to the research questions proposed in indexed scientific databases, and the papers selected are called primary studies.

One of the main goals of a systematic mapping study is to categorize an area of knowledge providing evidence that there is information and possible clusters where further research could be done.

The rest of the paper is organized as follows. Section 2 describes the paper's selection process used in this work and it explains the process conducted to perform the mapping. Section 3 reports the obtained results. Section 4 discusses the threats to validity of this systematic mapping. Finally, Section 5 presents the conclusions of this study.

II. Study Research Selection

In this study we followed the process proposed by Petersen et al. [6] (Figure 1). Next, we describe in detail how the process steps were performed.

A. Definition of Research Questions

The research questions to be answered in this systematic mapping study are the following:

***RQ1**: What are the main approaches used to address the practical experiences in software engineering education?* The idea was to create a comprehensive map with the main instructional approaches, and identify those most commonly used.

***RQ2**: Is there an emerging tendency to address this challenge?* This question tries to determine if the lessons learned from past experiences have generated consensus about the best ways to teach software engineering from a practical point of view.

***RQ3**: Which software process models are used to support the practical experiences in software engineering courses?* The use of a particular process model involves certain challenges and usage efforts for the instructors. This research question intends to determine which models can be afforded by the instructors during a course, considering the models complexity, and also the time and effort required for using them.

***RQ4**: Have the universities changed the way of conducting these experiences over the years?* This question intends to determine the evolution of the software engineering teaching approaches, in order to identify promising trends that help researchers and instructors to define new solutions to address the stated challenge.

***RQ5**: What are the main forums to seek information on practical approaches for teaching software engineering?* By identifying the main information sources on this topic, we intend to ease the searching and retrieving of new proposals on this research topic.

Although there are several proposals on how to teach software engineering, there is no clear consensus on whether or not there is a better approach to do it. The literature shows a multitude of anecdotal information indicating what is best and what everyone does. However, there is no empirical evidence supporting these suggestions. Therefore, this systematic mapping study intends to provide a more comprehensive and objective point of view on the practices used by universities to address the practical experiences in software engineering courses.

B. Literature Search

We started our mapping study by identifying keywords and search strings that would be useful in finding relevant articles. We followed the suggestions of Kitchenham et al. [7] who recommend: (1) breaking down the research questions into concepts, (2) finding synonyms, abbreviations and alternative spellings, (3) considering subject headings of relevant journals and works in the area, (4) using Boolean operators in the search strings.

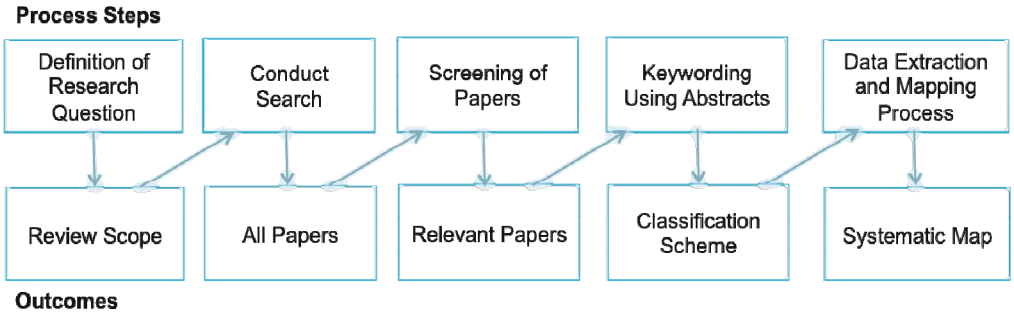

Figure 1 - Systematic Mapping Process by Petersen et al. [6]

Table 1 shows the search strings that were used to identify relevant papers. In such a process we matched these strings against the papers title, abstract and keywords, when possible. The following scientific databases were used in this searching process: IEEEXplore, ACM Digital Library, Web of Knowledge (former ISI Web of Science), Science Direct (Elsevier), SpringerLink and Wiley International. These databases are relevant sources, indexing the major journals and conference proceedings in the software engineering area. They also allow automatic searching.

Table 1 - Strings Used in the Literature Search

Search Strings
"software engineering education" **OR**
"software development course" **OR**
"software development capstone" **OR**
"software development education" **OR**
"software engineering capstone" **OR**
"software engineering course" **OR**
"teaching software engineering" **OR**
"software engineering instruction" **OR**
"teaching software development" **OR**
"software practice education" **OR**
"software engineering projects"

The search was done from October 1st to 30rh ,2013. A total of 7,517 documents were found. Table 2 shows the number of articles retrieved from each database. Provided that many papers are indexed by more than one digital library, we removed the duplicated articles and thus obtained a corpus of 3,725 documents related to the research topic. This corpus of literature represents the input used for the next phase of this study (i.e. the screening). Figure 2 shows the percentages of papers used from each literature database.

Table 2 - Amount of Articles Retrieved by Source

Source	Retrieved
ACM Digital Library (http://dl.acm.org)	2,100
IEEEXplore (http://ieeexplore.ieee.org/Xplore/home.jsp)	996
Web of Knowledge (http://www.webofknowledge.com/)	2,279
SpringerLink (http://www.springer.com)	934
Science Direct (http://www.sciencedirect.com)	2,075
Wiley International (http://onlinelibrary.wiley.com)	1,212
TOTAL:	**7517**

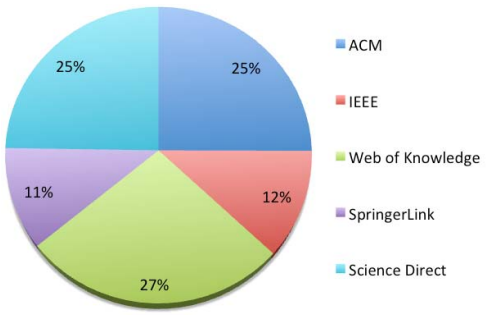

Figure 2 – Representativeness of the literature sources

C. *Screening of Papers and Keywording*

The main goal of the screening stage is to select studies that address the stated research questions. In our screening process, two inclusions criteria and eight exclusions criteria were used and presented in the following list:

Exclusion Criteria:

1. Documents whose full text is not available.
2. Documents that are only available as an abstract.
3. Documents describing a panel.
4. Documents describing a workshop.
5. Documents that are a letter from the editor.
6. Documents in which the context is not related to teaching software engineering.
7. Documents that report the use of software engineering as a means of teaching a specific programming language or framework.
8. Documents that report opinion papers and philosophical theories.

Inclusion Criteria:

1. Documents reporting teaching/learning software engineering experiences.
2. Documents presented as full papers; short papers and works in progress are counted if they are within the research scope.

The authors individually applied the inclusion and exclusion filters, and then examined the title, abstract and keywords of each selected paper. The results were later compared and the conflicts were discussed between the authors. A total of 247 papers were selected after this first filter. Then, a skimming through the introduction and conclusion sections of each paper was conducted in order to select the primary studies that help answer the research questions. As the result of this skimming, 173 papers were selected as primary studies, which were used for the keywording stage.

According to Petersen et al. [6], "*keywording is a way to reduce the time needed in developing the classification scheme and ensuring that the scheme takes the existing studies into account"*. This stage involves two steps: (1) look for keywords that reflect the concepts and contribution of the paper, and (2) combine the keywords to classify and understand the contribution of the research. Following this proposal, we defined three criteria for classifying our primary studies:

Research type – It identifies the research approach used in the studies reported in the papers. Table 3 shows a description of the research type categories used to classify the articles. These categories are based on those proposed by Petersen et al. [6] and Nacimento et al. [8].

Teaching approach – It identifies the pedagogical teaching approaches used in these practical experiences. Table 4

presents the categories of teaching approaches and their description.

Software process model – It indicates the type of software process used to guide these experiences. Software development processes are "*used as guidelines or frameworks to organize and structure how software development activities should be performed, and in what order"* [9]. Table 5 shows the categories of software processes used to classify the experiences, and also their description of each category.

Table 3 – Research Types

Category	Description
Experience report	Author´s personal experience describing what and how something was done. This usually includes a lesson-learned section.
Case study	Report of investigation of a real-life phenomenon. It may include quantitative evidence, relies on multiple sources of evidence, and also may report one or several cases.
Action Research	Problem solving actions implemented in a collaborative context with data-driven analysis or research, with the intent to enlighten underlying causes enabling future trend prediction.
Experiment/Quasi-Experiment	Manipulation and controlled testing for causal analysis. Normally, one or more variables are manipulated to determine their effect on the dependent variable.
Solution Proposal	A solution is proposed and the results of using the proposed solution are reported. An example or a line of argumentation must show the applicability of the solution.

Table 4 - Teaching Approaches

Category	Description
Case Studies	Students develop skills in analytical thinking by reading and discussing complex real-life problems.
Learning by doing	There is no formal reported approach stated in the paper, but report the experiences on using an approach where students take on a project and have to address the clash of software engineering theory and practice.
Game based learning	A game development is proposed as a fun way to teach software engineering and catch the students' attention.
Maintenance	Students have to learn how to deal with software created by other developers, and to participate in its evolution or correction.
Open Source	Students have to learn software engineering by participating actively in an open source community or project.
Problem/Outcome Based Learning (PBL/OBL)	These experiences address practical problems as way to drive the learning process. They can also be focused on getting certain outcomes as a way to emphasize a more functional knowledge than a declarative one. Students learn about a subject through the experience of solving a problem, looking for results and by working in groups.
Simulation	Students work on projects that try to simulate a part of a real life project. For instance they have to play a role, deal with a client, address a problem, and work in an environment similar to industrial scenarios.
Traditional	Involves a lecturer standing in front of students, with the course content divided into a number of topical lectures. Typically, a set of practical assignments is used to help students apply the theoretical concepts to the professional work.
Service Learning	Students develop and use their academic skills to address real life problems within their own environment.
Inverted Classroom	The traditional lecture is placed online for students to utilize during their study time. Their classroom time is used for other activities, such as workshops, interactive work time with the instructor, or demonstrations.

Table 5 - Software Process Models

Category	Description
Agile	The agile models are product-driven processes that propose a software development approach based on customer collaboration, iterative development and technology adaptable to change [10]. Little or no bureaucracy is involved in these processes.
CMM	CMM (Capability Maturity Model) is a general recommendation that identifies key process areas, and also different levels of work maturity for those areas. These experiences involve planning, performing or validating software process improvement using this recommendation as a guideline.
Mbase	Mbase (Model Based Architecting and Software Engineering) is a particular software process model, which is focused on managing the traceability of the project deliverables.
RUP (based)	Works that report the use of RUP (Rational Unified Process) or a variation of it to guide the development of software projects. Some processes derived from RUP are UP (Unified Process) or UPEDU (Unified Process for Education).

TSP/PSP	Papers that report on the use of TSP (Team Software Process) or PSP (Personal Software Process) [11]. These processes are focused on self-measuring and managing the development effort during a project.
Traditional	Works that report the use of waterfall, incremental or any other structured approach for software development. Typically these processes are opposite to agile methods in terms of bureaucracy required to obtain a product.
Various	Papers that report the use of more than one process approach simultaneously. For instance, using a structured model for developing a particular software. We however use an agile approach to address each process stage.
Ad-hoc (stated)	Reports that use a specific process approach, which is described in that work, and that do not match with previous categories.
Not specified	Authors do not state the process approach used to address the work.

D. Data Extraction

The data extraction process conducted in this study was particularly designed to answer five research questions. In order to do that, each paper that passed the screening process was read to determine the correct classification of the work according to the defined criteria. Moreover, we retrieved the following information from those papers: title, year, venue and author(s). The information required to classify the articles were in some cases stated in the article, and in other cases, they were manually classified by the authors considering the papers' content. In this work, none of the papers were classified in more than one category according to a same criterion.

III. OBTAINED RESULTS

The details of the 173 papers selected as primary studies are available in [12]. The results are presented according to the stated research questions. Figure 3 gives us an overview, a map of the approaches reported in the primary studies. The bubbles indicate the number of papers classified according to the already defined criteria; i.e., research type, software process model and teaching approach. Next, we present the obtained results for each research question.

***RQ1**: What are the main approaches used to address the practical experiences in software engineering education?*

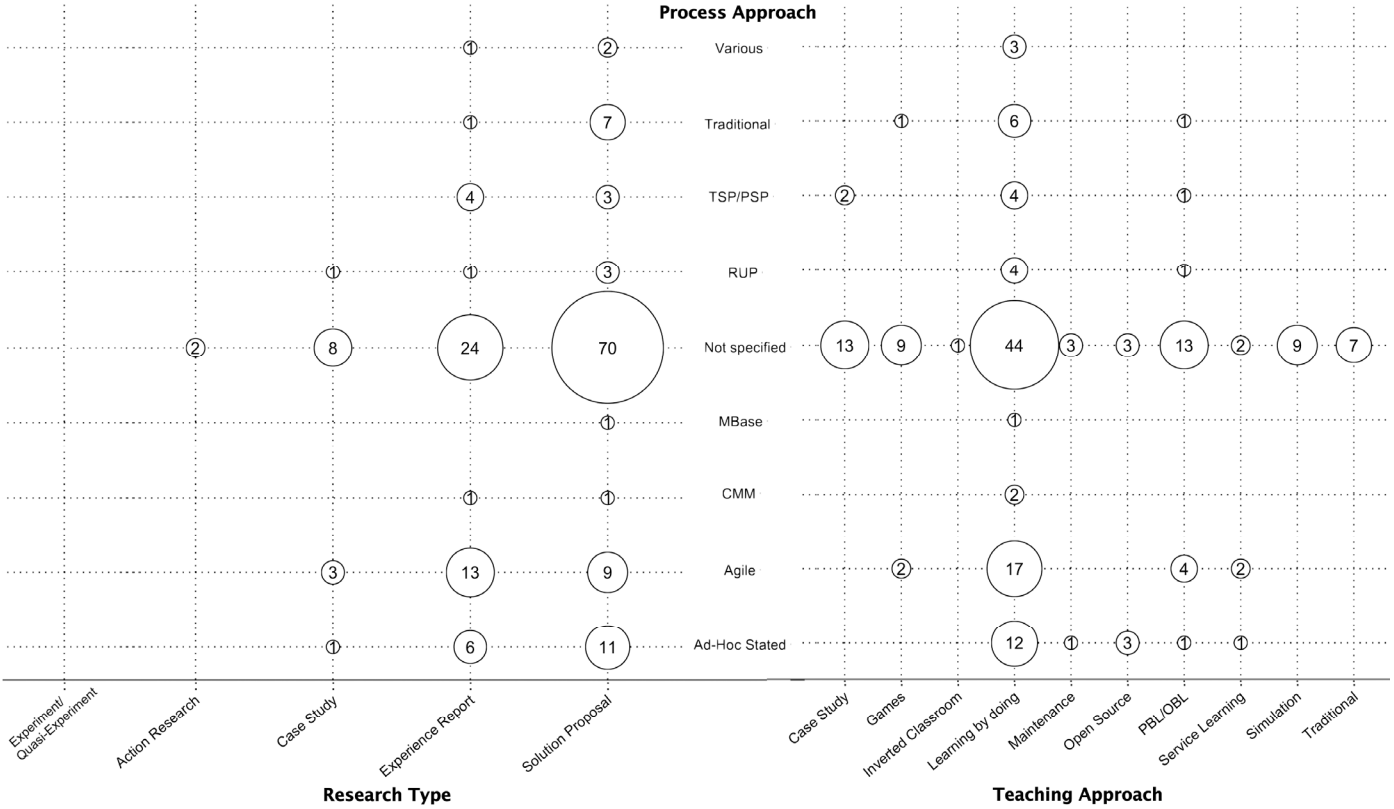

Figure 3 - Research Type versus Process Approach versus Teaching Approach

The literature reports ten instructional approaches to addressing these practical experiences. Clearly, the variety of alternatives is small.

The map shows that the *learning by doing* approach is the most used one, with 93 studies (54%), followed by the *PBL/OBL* approach with 21 studies (12%), *case studies* with 15 studies (9%), *games* with 12 studies (7%), and then *simulation* (5%), *traditional* (4%), *open source* (3%), *service learning* (2%) and *inverted classroom* with only one study.

Learning by doing and also PBL/OBL are flexible strategies that can be adjusted to take advantage of the capabilities and interests of the instructors and students. These approaches do not impose important constraints or require special capabilities of the instructors. This could be a reason why these alternatives seem to be frequently used to support these practical experiences.

***RQ2**: Is there an emerging tendency to address this challenge?*

The tendency seems to be defined by several criteria, for instance, the flexibility and ease of using the approaches, their suitability for being used by most instructors, and the effort, restrictions and skills involved in the use of these approaches. The map shows that approaches involving real projects or clients (that are most of the alternatives) are not particularly attractive for software engineering instructors. This is probably because these approaches are more complex to apply and also involve significant effort from instructors, teaching assistants, and sometimes also from clients and users.

We suppose that the traditional approaches (i.e., those that compliment theoretical lectures with practical assignments) are the most used, because it is not easy to change the instructional process used by the lecturers, and also because most computer science and engineering courses follow that pattern. However, there are only a few reports on the use of traditional approaches in the literature. This is likely due to such an approach being not as novel and it is rarely effective at delivering knowledge in this area.

***RQ3**: Which software process models are used to support the practical experiences in software engineering courses?*

Viewing the map and analyzing the software processes used to guide these practical experiences, we see that more than half of the studies do not report the process used in these experiences (104 studies – 60%). This result is not surprising, because many software engineering practical experiences tend to simulate the work scenario of small or very small software enterprises, due that will probably be the first professional niche addressed by the new software engineers. According to a study performed by Quispe et al. [13], the software process models used by these enterprises are typically informal, immature and guided by the deadlines. Therefore, if the experiences in the academia simulate this part of the software industry (that is the most representative one), clearly these experiences do not have a process to be reported in the papers. In summary, it is highly probable that the process used in the academia to guide the practical experiences is informal and guided by the deadlines.

Concerning the reporting of articles, most of the processes used in these experiences indicate the use of an agile process. Agile processes are usually not too formal and guided by the deadlines, which support the previous hypothesis. Twenty-five studies reported the use of agile processes (14%), followed by ad-hoc process (stated in the study) with 18 studies (10%), traditional processes with 8 studies (5%), TSP/PSP with 7 studies (4%), RUP with 5 studies (3%), various other process with 3 studies (2%), CMM with 2 studies and MBase with 1 study. Supporting the hypothesis that the complexity and effort of conducting these experiences establishes the usage tendency. Few articles report on the use of RUP, TSP/PSP, MBase and CMM. The traditional processes represent an interim point because they are a bit bureaucratic (e.g. waterfall or incremental), but easy to use. Their usage does not involve significant effort by the instructor. Clearly, the preferred are the agile processes because they are easier to use.

***RQ4**: Have the universities changed the way of conducting these experiences over the years?*

Figure 4 shows a timeline with the reported teaching approaches used in these practical experiences. Initially the traditional approach and learning by doing were used; however, during the last ten years several other alternatives have been proposed and used indicating that this issue is still open.

Nowadays, the learning by doing approach is still in use and the traditional one has lost ground. Maintenance was used at some point, but did not have continuity of use. This could be because today the development of new solutions is much more frequent than the extension of the existing ones. Other approaches such as games, open source, PBL/OBL have started to be reported on over the last ten years and continue being used as an attempt to find new ways of addressing the stated challenge.

***RQ5**: What are the main forums to seek information on practical approaches for teaching software engineering?*

The majority of the primary studies selected were published in conference proceedings (125 studies – 72%) and only 48 studies (28%) were journal publications. The main forums on the theme were the Conference on Software Engineering Education and Training - CSEE&T (40), Frontiers in Education – FIE (18), International Conference on Software Engineering – ICSE (7), ACM Technical Symposium on Computer Science Education (6) and Conference on Innovation and Technology in Computer Science – ITiCSE (5) and the rest of the articles were published in 36 different conferences.

Concerning the articles reported in journals, 12 studies (25%) were published in the ACM SIGCSE Bulletin, 11 studies (23%) in the Journal of Computing Sciences in College, 8 studies (17%) in IEEE Transactions on Education, 3 studies (6%) in the Journal of Systems and Software and 2 studies (4%) in the Software Engineering Journal. The rest of the studies (12 studies – 25%) were published in several journals, which are not particularly focused on software engineering or engineering education.

IV. THREATS TO VALIDITY

Construct validity reflects to what extent the phenomenon under study really represents what the researchers have in mind, and what is investigated according to the research questions. The number of papers found indicates that the terms used in the search process are well defined. Only the data available before October 30, 2013 was considered in this study.

Data reliability focuses on the papers collected and analyzed in order to see if these processes were conducted in a way that they can be repeated. The search terms were defined and the procedures applied in this study followed the previously described guidelines. The corpus of literature used as a primary study is reported in [12], therefore other researchers can replicate this study. The extracted information could also be a source of reliability concern, but the authors made the data extraction in an individual and parallel way. In case of differences in the authors' opinion, a clarification process was conducted until reaching consensus.

The results found could be subject to limitations of the automated search engines used in this work. The primary studies considered here were the ones that entered the inclusion criteria and were not rejected by the exclusion criteria. The classification of the studies according the predefined criteria was not always straightforward (some classifications were implied by the authors). The classification of the studies was done mainly by reading the title, abstract and in a few cases, the whole article.

Internal validity is more related to analysis of the collected data. A mapping study does not have any statistical assumption; therefore there are no threats. External validity is concerned with generalization. Since the systematic mapping studies in general do not state conclusions, the external validity threats are not applicable.

V. CONCLUSIONS

This paper presents a systematic mapping study that characterizes the approaches available to address the practical experiences in software engineering education. We used a corpus of literature of 173 primary studies that were published in the main conferences and journals of the area. The results show that the academia is conscious of the need to teach software engineering in a practical way, and it is also conscious of the effort that this challenge represents for instructors, teaching assistants and universities.

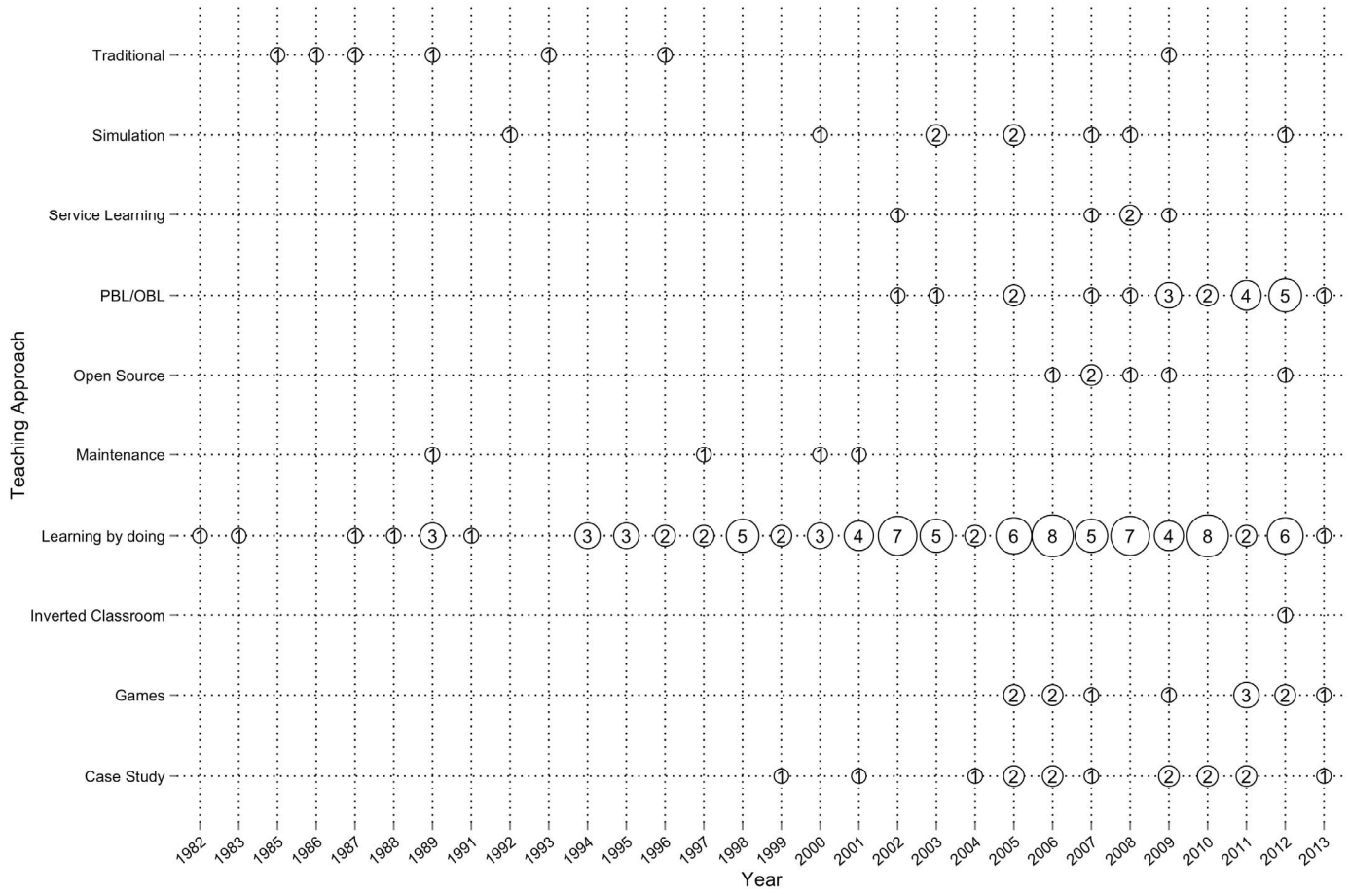

Figure 4 - Timeline of the teaching approaches

Although several approaches have been proposed to address this challenge, the preferred ones seem to be those that are easy to use, and represent minimum effort for the instructors. As there is no comparison of the effectiveness of these approaches, we therefore cannot say anything about it.

Concerning the software process models used to guide these experiences; the preferred ones seems also be those requiring less effort and complexity in their use. In that sense, agile methods were preferred considering the articles that reported this feature of the practical experiences. Most studies do not indicate the software process used in the experiences. If we draw a parallel between these experiences and the development scenarios of very small and small software enterprises, we can hypothesize that no process was used in the practical experiences in the academia. The process will probably be informal and guided by the deadlines assigned to the project; therefore there is no process to report.

This result is not surprising because teaching software engineering in a practical way is still an open issue, and represents an important challenge for any university. More research and experimentation is required in this area in order to find new proposals that address this challenge more effectively.

The map presented in this study shows some trends and issues in software engineering education. This gives us some directions for future research. We plan to perform a more in-depth analysis of the proposed teaching approaches and software processes in order to determine their effectiveness in the instructional process.

ACKNOWLEDGMENT

The work of Maíra Marques Samary was supported by the PhD Scholarship Program of Conicyt Chile (CONICYT-PCHA/Doctorado Nacional/2012-21120544). This work was also partially supported by the Project Fondef Idea, grant: IT13I20010.

REFERENCES

[1] IEEE Standards Collection: Software Engineering. IEEE Standard 610.12-1990, (1993).

[2] Chen, J., H. Lu, L. An, and Y. Zhou. “Exploring Teaching Methods in Software Engineering Education”. Proceedings of the 4th International Conference on Computer Science & Education (ICSE). IEEE Press (2009): 1733-1738.

[3] Begel, A., and B. Simon. "Novice Software Developers, All Over Again". Proceedings of the 4th International Workshop on Computing Education Research. ACM Press (2008): 3-14.

[4] Moore, M., and C. Potts. “Learning by Doing: Goals and Experiences of two Software Engineering Project Courses”. Lecture Notes in Software Engineering Education, Vol. 750. (1994): 151-164.

[5] Giraldo, F., C. Collazos, S.F. Ochoa, S. Zapata, and G. Clunie. "Teaching Software Engineering from a Collaborative Perspective: Some Latin-American Experiences". Proceedings of the IEEE Workshop on Database and Expert Systems Applications, IEEE Press (2010). 97-101.

[6] Petersen, F., R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic Mapping Studies in Software Engineering". Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering, (2008), pp. 71-80.

[7] Kitchenham, B.A. “Guidelines for performing systematic literature reviews in software engineering version 2.3”. Technical Report S.o.C.S.a.M. Software Engineering Group, Keele University and Department of Computer Science, University of Durham, (2007).

[8] Nascimento, D.M., K. Cox, R.A. Bittencourt, R. Souza, and C. Chavez, “Using Open Source Projects in Software Engineering Education: A Systematic Mapping Study”. Proceeding of the Frontiers in Education Conference, (2013).

[9] Scacchi, W. Process Models in Software Engineering. 2nd Edition. New York: John Wiley and Sons, Inc., (2001).

[10] Cockburn, A. A Human-Powered Methodology for Small Teams. Boston: Addison Wesley, (2004).

[11] Humphrey, W. "Using a Defined and Measured Personal Software Process". IEEE Software, (1996), pp. 77-88.

[12] Marques, M., A. Quispe, and S.F. Ochoa. “Corpus of Literature for a Systematic Mapping Study on Practical Approaches for Teaching Software Engineering”. Technical Report No. DCC-20140425-002. Computer Science Department, University of Chile. (2014). Available at: http://www.dcc.uchile.cl/TR/2014/TR_DCC-20140425-002.pdf

[13] Quispe, A., M. Marques, L. Silvestre, S.F. Ochoa, and R. Robbes. “Requirements Engineering Practices in Very Small Software Enterprises: A Diagnostic Study”. Proceedings of the XXIX International Conference of the Chilean Computer Science Society, pp. 81-87. IEEE Press. Antofagasta, Chile, Nov. 15-19, 2010.