Social Thinking to Design Social Software: A Course Experience Report

Cédric Mesnage, Mehdi Jazayeri
University of Lugano
Faculty of Informatics
6900 Lugano, Switzerland
cedric.mesnage@lu.unisi.ch, mehdi.jazayeri@unisi.ch

Abstract

Open-source development, social production, social networks and other factors change the way we understand software systems. The paper motivates the use of social thinking to design social software. We report on our experience in teaching a social softare engineering course. We evaluate the course and the use of social thinking. We conclude with further research questions.

1. Introduction

Our research objective in the recent years was oriented towards the understanding of the mechanisms behind the engineering of social software, notably in building and recovering the architecture of the NEPOMUK –the social semantic desktop–[1] european project, and the study of web systems such as social tagging and social networks. These activities led to us to teach a social software engineering course.

Although we speak of it now as a social software engineering course, the course was originally on software engineering for web applications. The course was held in the context of atelier in the faculty informatics of Lugano [11] where students focus on group work.

We articulate the paper around three key ideas :

- Software has essential social aspects. It must be taken into account when building software systems.
- Social software engineering differs from traditional software engineering in principle and practice. We identify the major differences.
- It is possible to teach social software engineering. We report on our experience in a bachelor semester course.

[1]this work is partly founded by the NEPOMUK european project under the grant FP6-027705.

Bringing research aspects into education at the bachelor level and reflecting about it is challenging. Research in software engineering education is crucial [13] for the field to evolve and software to improve. The use of social networks in computer science education brings new interest and opens it to a different spectrum of students[1].

Structure of the paper Some of the discussion is double-sided as for each topic we must consider the topic in the 'real world' and when teaching it. Section 2 introduces social software. Section 3 gives reasons to use social thinking in the design process when designing social software. Section 4 compares social software engineering versus traditional software engineering. Section 5 describes the course. Section 6 reports on our experience. Section 7 gives an evaluation of the use of social thinking in the software process and the evaluation of the course. We conclude with open questions.

Building social software requires specific activities. People are central to social software systems. We argue that building social software systems requires social thinking to understand people and their interactions.

2. Is software social?

Any piece of software has a social dimension in the way it is designed, developed, used, or maintained. We understand social software systems as software systems that provide support for social interaction and production.

Email is social software. Human activities supported by email software systems are numerous. People use emails to help each other, to share information resources, to connect people, to plan and schedule meetings, to communicate and collaborate. Being conscious of the social nature of software changes the way we design and understand software systems.

Table 1 relates human activities and social software features.

Features	Activities
social networks	connect, help, communicate
forums	help, communicate, inform, question
tags	connect, share, organize, abstract
blogs	share, communicate, inform
shared repositories	share, program, document
emails	connect, help, share, communicate, collaborate
feeds	notify, inform, abstract
wikis	collaborate, educate, document
online polls	decide

Table 1. Social software systems architecture

3. Why do we want social thinking in the software process?

Social thinking is used here to refer to scientific reflection guided, informed, and/or inspired by social science approaches[5]. Social sciences gather evidence and build facts about social systems[3]. We outline some of the reasons that require us to understand social systems.

Open-source The traditional software engineering methods differ from open source development[2]. There is no client for open-source software, software is developed by a community because there is an apparent need for it or to develop an idea. The requirements are not specified and documentation is mostly technical. What methods can we use to justify these developments and document their design rationally?

The internet The growth of number of users of the internet network facilitates the development of services which are economically viable (think Google). These software systems are used by millions of people. How do we reason about these systems in order to improve them?

Social production The collaborative and open production of information artifacts (think wikipedia, del.icio.us) changes the way we consume information. Benkler in [2] discusses the economic implications of such changes. How do we design such systems? What are the social interactions to be supported?

Global communities The growth of social networks reveals the global interconnection of people in our society. These software systems support activities across cultural, geographical, and social boundaries. How do we develop applications which satisfy the diversity shown in such communities?

[2] `http://en.wikipedia.org/wiki/Open_source`

Hardware The cost of nomad computing hardware keeps decreasing (think netbooks[3], one laptop per child). Personal devices and network connectivity is cheap and ubiquitous (think cell-phones and 3G).

4. What is social software engineering?

Social software engineering is the activity by which people build software designed consciously to support social activity. We discuss here some differences between traditional software engineering and social software engineering.

Social problems vs contracts The software engineering process is a set of activities set up in a project to satisfy the goals specified in a contract. Social software engineering is problem-based, the goals being to design and implement ideas to solve problems for a community. The social software engineer needs tools to understand these problems.

People vs clients Many developed software systems are not designed for a particular client but for people (open-source, social networks). To better design these systems we need to understand the people who are using them and the interactions and activities they have. Social thinking gives us a framework to systematically study social systems, results from sociology and psychology can be used as a direct input for rational design.

Evidence vs requirements The elicitation of requirements is recognised by the software engineering community as a critical task in software development. Building social software requires understanding the people behaving in the social system. The evidence and facts provided by social thinking gives us strong basis to reason about social systems and design solutions to support understood social interactions.

5. The course

We report on our experience in teaching a course of software engineering for social web applications. The course intertwines theoretical principles of software engineering and practical aspects of developing Web applications with Web technologies[4].

[3] `http://en.wikipedia.org/wiki/Netbook`

[4] We chose Ruby On Rails as the main development framework. `http://www.rubyonrails.org/`

5.1. Students projects

The students developed open-source projects from the formulation and understanding of a social problem to the design, implementation, and evaluation of a software system.

Kiwish A solidarity network to work on World poverty. People can help each other to realise charity projects such as the construction of a well in Africa by creating wish lists of required goods and activities.[5]

OSLP[Open Source learning platform] Computers and education. A shared repository of pointers to online courses and other learning resources. People tag resources with keywords, grade them, and comment them to build a collaborative index. A trust network is maintained to rank people according to their reputation(karma). The system makes recommendations based on the user profiles.[6]

WinCS To improve communication in our faculty, the students developed a Web application to configure a physical screen in the entrance of the faculty. The screen displays a map of the rooms with current events, announcements, and upcoming events [7].

5.2. Constraints

Although we have some freedom in designing the course, there are some constraints which must be taken into account.

- The course is held during the fourth semester of a bachelor curriculum, therefore the level of the students must be considered.
- The course must relate to previous or concurrent courses and prepare for the future courses the students are going to take.
- The instructor supervises the course and the course is taught by the assistant.

5.3. Traditional software engineering educational objectives

We find the traditional software engineering educational objectives by looking at early work on software engineering education[8, 7, 6]. The experience reported in these workshops give us a basis to know what to emphasize in the course from the point of view of people who created software engineering courses, often as capstone projects. The technology evolved but elements such as problem solving, design, communication, style, and realism are still required for a software engineering class. Tomayko gives a historical view of software engineering education[17].

We understand the principles of software engineering based on [9, 10]. Dijkstra's chapter in [4] is a source for programming style. Parnas in [14] gives a more complete list of what a software engineer should know.

5.4. Course design

The course is one semester long composed of 14 weeks. Table 2 gives an overview of the activities held throughout the semester.

Activity	Week
Technology background	1
	2
	3
	4
	5
Problem formulation	6
Social research	7
	8
Rational design	9
	10
Software development	11
	12
	13
Evaluation	14

Table 2. Course activity schedule

Tuesday	assignment presentation
Wednesday	conversational lecture
Thursday	tutored assignment

Table 3. Weekly schedule

5.5. Course activities

Technology background The first weeks of the semester are used to train the students to use the Ruby On Rails framework. The students are given implementation assignments that they present in class. They implement a todo list management application. We introduce new aspects every week, starting from the basics of programming with the

[5] http://code.google.com/hosting/p/kiwish
[6] http://code.google.com/hosting/p/oslp
[7] http://code.google.com/hosting/p/wincs

Ruby language, discovering the framework, modeling relational data structures, dynamic user interfaces with AJAX[8], using remote application programming interfaces, testing and deployment. During this phase we check the knowledge and skills of the students and make sure they understand the necessary basics of Web development. The lectures are given based on what we see the students do not understand when they present their assignments. Table 3 represents the course weekly schedule during this phase.

Problem formulation Before a one-week easter semester break the students are asked to look for problems. We discuss in class the different problems they can see and abstract this list to general social problems. The students divide into groups of three people and have a brainstorming session to share their ideas regarding the problem they pick. The result of this phase are posters made during the brainstorming which will be left on the walls of the classroom for the rest of the semester.

Social research To better understand the problems and people's opinion, the students conduct social research. They design a short questionnaire composed of five questions about different aspects that they want to learn about. They collect data by doing structured interviews, going around in the faculty and university to fill in 30 questionnaires per group. The second week of this phase we analyze and interpret the data in class, crossing variables to reveal relevant findings. The phase is concluded by a formal presentation in which the groups report on their survey in front of the class and their coach. Following the presentations a coach is assigned to each group, and the team draws conclusions for the design.

Rational design Parnas in [15] decribes why and how to fake a rational design process in order to produce valuable documentation. The groups make design decisions and document them in a wiki based on a given tabular template. Their design is iteratively reviewed and discussed. The students give the rationale behind their decisions based on the findings their survey revealed. The phase ends with a formal presentation reporting on the functionalities, features, architecture, usage dependencies, implementation plan and initial demo of their application.

Software development The applications are developed following an agile process. The groups iteratively develop the features starting with the ones with no dependencies. They present them in class to get feedback and decide on next steps. The groups code collaboratively in a version controlled repository. The phase ends with a formal presentation the last week in front of the coach in which they show the code for their main features and demo their application. They receive feedback to finalize their projects.

Evaluation and presentation The last week of the course is used for wrap up, discussing the deviations from the initial plans, what was learned, what didn't work. The final presentation is made in front of the whole faculty in which the groups report on all the activities and present a final demo of their software.

5.6. Course features

We discuss here some of the features that are transversal to the course.

Abstraction The training of the abstraction skills of the students is continuous, making them conscious of when they need to abstract and when they are abstracting. The assessment of their level of abstraction at the beginning of the course is a good measurement to filter the students who will have too many difficulties to continue and are asked to drop the course.

Time management Lectures are added when needed. This makes it difficult to keep the schedule as planned.

Assessment Students are assessed on a weekly basis. Their assignments are graded by the assistant during the first phase. The students grade themselves during the project development reporting individually on their activity within the groups.

The use of Ruby On Rails The framework is particularly useful for rapid prototyping. There are many problems with it. The available documentation the students find on the web varies in quality and is often outdated, leading the students on wrong paths. Most of the libraries and plugins are not documented. This makes it a perfect platform to teach accidental complexity.

Open-source collaboration platform The projects are hosted on Google code. The platform provides open-source project support with a wiki, an issue tracking system and a Subversion repository. The students are urged to use the wiki to write their documents.

Resources During the semester the course takes an average of 20 hours/week of the assistants' time involving preparing and giving the lectures, tutoring, giving feedback and reviewing documents. The course takes about 6 hours

[8] Asynchronuous Javascript and XML

per week of the instructor's time, involving the design of the course and attendance to lectures. About 15 hours are spent before the semester to prepare the course. The coaches spend about 10 hours throughout the semester.

The students are expected to spend 10/15 hours per week on the course including the class time of 6 academic hours per week.

6. Experience

This is the third time this course is being held and the first time we report about it. The course design has evolved to its current state as different assistants were teaching it, the technology being used (Ruby On Rails) has not changed. We give the historical log of the assistants involved in the course, the instructor has not changed :

Spring 2008 Cédric Mesnage.

Spring 2007 Jeffrey Rose, Cyrus Hall.

Spring 2006 Cédric Mesnage, Jeffrey Rose.

The first year the students were divided as well into small groups of 3 or 4. The second year the whole class worked together on the same application using open-source tools.

7. Evaluation

We evaluate two aspects : the course and the use of social thinking in the software process.

7.1. Course evaluation

The evaluation of the effects of such a course on students is a hard problem. We do not have a control group and the evolution of the students is affected by many external factors such as other courses and personal situations.

We have only taught this course for a small number of students and we do not know if such a course can scale up.

A formal evaluation of the course was conducted by the quality service of the university, a month before the end of the course. It reflects the feeling of the students before starting the development of their projects. Moreover this survey does not distinguish between the professor and the assistant, which can be confusing for the students.

The semester ends by student presentations. The groups presented their projects in public. The feedback from the public gives an evaluation to the students.

The final assignment of the course is the writing of an essay. The students are asked to describe their experience in the course by reporting on three negative aspects, three positive aspects, and proposals for improvement. These texts give us a quantitative evaluation of the course from the point of view of the students.

The opinions of the students on the course are diverse. Most of them like the development of the project and realise they learned a lot. Opinions differ essentially on the teaching methodology. Some are reticent to self-teaching and self-assessment, it is more demanding on the students. Some others see it as a training for real-life when they have to evolve by themselves. The question is why we did not manage to have every student understand the rationale behind the methodology, we have to be clearer from the beginning of the class.

7.2. Social thinking evaluation

Does the use of social thinking affect the process and the product and in which way? The students found useful the use of social thinking, essentially in the survey they conducted. They assimilated it as a normal software engineering activity and felt necessary to question people and analyze the collected data to drive the design.

For a more detailed evaluation we need to look at the design of the students projects and see how the use of social thinking influenced the design decisions.

8. Conclusion

We gave reasons for using social thinking when building social software, we have shown how social thinking can be used in the software engineering process as an input for the design phase, we described an experimentation in a semester long class with bachelor students, we gave an evaluation of our experience.

The classroom is an excellent environment to experiment with new methodologies. In the real world, the activities done in the course would happen concurrently and not sequentially, going back and forth to iteratively refine the projects.

The course can evolve in different ways, we are considering moving from Ruby on rails to using social networks frameworks such as Facebook or Google OpenSocial which both offer javascript facilities. We need to be clearer with the students on the social dimension of the course

For a better evaluation of our experience we should compare with related work such as the MIT[9] active learning method.

Our goal is to define a framework for problem-based[16] social software education. This involves different questions, how to integrate with social networks, what methods from

[9]The Massachusetts Institute of Technology develops methods to promote active learning through student projects which often have a social dimension, such as using cell-phones to support tuberculosis patients `http://web.mit.edu/edtech/home.html`

social sciences research[3] can be used and taught in such courses, how to teach abstraction[12].

Acknowledgements

We would like to thank the previous assistants, Jeffrey Rose and Cyrus Hall, the coaches Francesco Lelli, Monica Landoni and Mark Carman and the 35 students who attended the course over the last three years.

References

[1] C. Alt, O. Astrachan, J. Forbes, R. Lucic, and S. Rodger. Social networks generate interest in computer science. *SIGCSE Bull.*, 38(1):438–442, 2006.

[2] Y. Benkler. *The Wealth of Networks: How Social Production Transforms Markets and Freedom*. Yale University Press, New Haven, CT, USA, 2006.

[3] G. D. Bouma and R. Ling. *The Research Process*. Oxford University Press, USA, fifth edition, 2004.

[4] O. J. Dahl, E. W. Dijkstra, and C. A. R. Hoare, editors. *Structured programming*. Academic Press Ltd., London, UK, UK, 1972.

[5] Y. Dittrich, C. Floyd, and R. Klischewski, editors. *Social thinking: software practice*. MIT Press, Cambridge, MA, USA, 2002.

[6] R. E. Fairley and N. E. Gibbs, editors. *Software Engineering Institute and Wang Institute of Graduate Studies on Software engineering education: the educational needs of the software community*. Springer-Verlag New York, Inc., New York, NY, USA, 1987.

[7] P. Freeman and A. I. Wasserman, editors. *Software Engineering Education: Needs and Objectives*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1976.

[8] P. Freeman, A. I. Wasserman, and R. E. Fairley. Essential elements of software engineering education. In *ICSE '76: Proceedings of the 2nd international conference on Software engineering*, pages 116–122, Los Alamitos, CA, USA, 1976. IEEE Computer Society Press.

[9] C. Ghezzi, M. Jazayeri, and D. Mandrioli. *Fundamentals of software engineering*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1991.

[10] P. Inverardi and M. Jazayeri. Software Engineering Education in the Modern Age. *Software Education and Training Sessions at the International Conference on Software Engineering, ICSE*, pages 15–21, 2005.

[11] M. Jazayeri. The education of a software engineer. In *ASE '04: Proceedings of the 19th IEEE international conference on Automated software engineering*, pages .18–xxvii, Washington, DC, USA, 2004. IEEE Computer Society.

[12] J. Kramer. Is abstraction the key to computing? *Commun. ACM*, 50(4):36–42, 2007.

[13] T. C. Lethbridge, J. Diaz-Herrera, R. J. J. LeBlanc, and J. B. Thompson. Improving software practice through education: Challenges and future trends. In *FOSE '07: 2007 Future of Software Engineering*, pages 12–28, Washington, DC, USA, 2007. IEEE Computer Society.

[14] D. Parnas and L. Chik-Parnas. Goals for software engineering student education. *SIGSOFT Softw. Eng. Notes*, 30(4):6–8, 2005.

[15] D. L. Parnas and P. C. Clements. A rational design process: How and why to fake it. *IEEE Trans. Softw. Eng.*, 12(2):251–257, 1986.

[16] M. Savin-Baden and C. Major. *Foundations of problem-based learning*. Berkshire [England]: Society for Research into Higher Education/Open university press, 2004.

[17] J. E. Tomayko. Forging a discipline: An outline history of software engineering education. *Ann. Softw. Eng.*, 6(1-4):3–18, 1999.