Teaching Software Design with Social Engagement

Damian A. Tamburri
Dept. of Computer Science
VU University Amsterdam, Netherlands
d.a.tamburri@vu.nl

Maryam Razavian
Dept. of Computer Science
VU University Amsterdam, Netherlands
m.razavian@vu.nl

Patricia Lago
Dept. of Computer Science
VU University Amsterdam, Netherlands
p.lago@vu.nl

Abstract

Software designers constantly mediate with various stakeholders, agree with requirement engineers and interact with coders. Software design is a socially-intensive activity. Teaching software design should be equally socially-intensive. However academic courses still lack a beneficial balance between theory, practice and social engagement. This paper provides details on how we address this problem in our course on software design. The course is designed to engage students with real-life projects and using peer-review sessions within collaborative team clusters. These instruments embed the social conditions of software design within the students' learning process. We show the effectiveness of the course by discussing student evaluations.

1. Introduction

Software design is a critical activity in the process of constructing software. Designers often need to act as a social link between stakeholders, their requirements and the rest of the development crowd. Courses on software design, however, are often too dense with both theory and practice of software design, and lack in an essential third ingredient, i.e. engagement as social connotation [23]. *We argue this social connotation is vital for students to understand the role of designing complex software systems.*

Software design should be taught within a professional learning community [13,22] in which the social engagement of students takes the upper hand. This paper presents our software design course and reports on our experience in delivering it by blending theory, practice and social engagement in a balanced way. This goal is realised with a combination of three factors: (a) giving students a challenging, real-life project from an industrial partner (playing the role of industrial customer); (b) organising student teams in collaborative clusters; (c) inspiring students' mutual learning through peer-reviewing and competition within clusters. Through weekly peer-review sessions, students also learn how to cope with different people's perspectives, backgrounds, requirements/concerns and shared understanding [4].

We have observed that this approach helps students in learning the main challenges of software design [18, 19]: (a) *Accountable and Rational Design Decisions* - students learn how to reason and really be accountable for their own design decisions; (b) *Collaborative Design* - students can brainstorm constructively with "opponent" teams, reaching a deeper understanding of the designer role; (c) *Iterative Design* - students learn from other people's mistakes and solutions pro-actively revisiting their own design, which is far more effective than any of the many examples we used in the previous years; (d) *"Social" Design* - students learn to make teamwork effective and cope with

978-1-4673-5139-3/13/$31.00 © 2013 IEEE

critical reviews driven by different backgrounds and expertise (hence offering feedback from very different perspectives).

The rest of the paper is structured as follows: section 2 describes our course; section 3 provides student evaluations and discussions. Finally, section 5 concludes this paper.

2. Service-Oriented Design Course at VU University Amsterdam

Software design means to apply the theory of software engineering to designing software-intensive systems. This process involves stakeholders of various nature [2] and is highly people-intensive. Also, software design is an incremental and highly interactive activity. We structured the Service-Oriented Design course at VU University Amsterdam such that theory, practice and social engagement become equally important. Social engagement is realised in practice by using a combination of peer-review sessions and stakeholder reviews. In the following we further explain the structure of the course.

Fig. 1 shows the weekly structure of our course yielding three types of sessions. Each week, on day 1, theory is given in the form of service oriented design lectures. On day 3, teams unite to constructively review the work of others, while on day 5 randomly selected teams present their results to the industrial customer. Both peer-review and customer presentation sessions are designed to increase social engagement among students and lecturers.

Our course is structured in three types of sessions: (1) theory; (2) peer-review; (3) stakeholder reviews. Each week is organised as follows: (1) two hours of design theory; (2) two hours of peer-review; (3) 2 hours of progress review presentations, audited by the "customer", i.e. the industry providing the real-life industrial case-study. The week starts with theory of service-oriented software architectures and the process of designing them following a service-oriented style (top-left on Fig. 1). Students are asked to start working immediately on the week's assignment so that on wednesday night (within 23:59 hours), each cluster can share internally the current version(s) for the peer-review session on the following day. At the end of the week the students are randomly selected to present their progress so far, stressing on design issues and concerns that were of particular importance/difficulty. Supervisors and industrial stakeholder, enact a classroom-learning session asking questions to clarify the issue or further exploring it, e.g. by splitting the issue into multiple questions. In the following we focus on three key aspects, namely: (1) real-life project; (2) peer-review clusters; (3) stakeholder reviews.

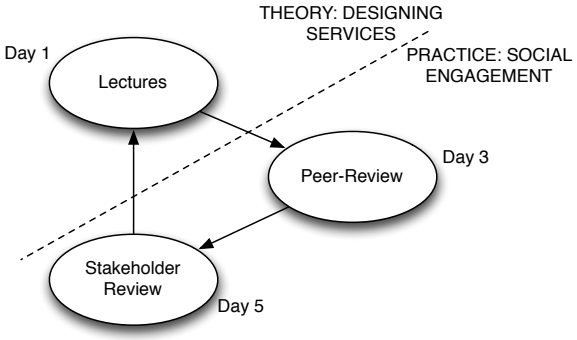

Figure 1: SOD Weekly Cycle - Theory of Services Design applied through social engagement.

1. **Real-Life Project**. Each academic year the project is provided by one of our industrial partners. Over the years we found that a non-trivial example of domain and development problem stimulates the creativity of students and challenges them to be actual designers rather than mere apprentices. In doing so, we tried to simulate an industrial environment with intensive, recurrent and fixed deadlines featuring partial increments, e.g. on a weekly

Table 1: Our Student Projects

Project Characteristics	In our Context
Number of Project Requirements	Every project satisfied an average of 16 requirements.
Number of Non-Functional Requirements	non-functional requirements or other desirable quality attributes were 9 in average. These needed to be exhibited and maintained by the designed service offer.
Number of Business Services	Projects normally exhibit an average of 10 business services that students must design into a complete service offer.
Size of the Design Space (in terms of Design Issues and related Options)	On average the projects exhibit 8 design Issues formulated in the form of design questions. Each question has at least 3 options.

basis. Tight deadlines, scheduling and organisational efficacy are therefore of key importance. Our final goal is that of giving real challenges from real-life scenarios in a real-life environment. The real-life project typically features a specific business domain (e.g. airports automation services, smart-building services) together with sizeable needs for the service offer to be designed by students. In previous editions, our projects featured in average for every group some 30+ requirements (specific business, functional/non-functional, etc.). Table **??** contains additional details describing our projects.

Pedagogic Value: *Software engineering literature, e.g. [23], suggests that designing software consists of finding a reasonable and rational equilibrium among many design alternatives and uncertainties. Teaching software design should also embed such uncertainty and variety of alternatives. Using project in a quasi-real context is not novel. We tailored our approach so that students have an industrial project featuring a real scenario as well as real customers and their requirements for a real environment. This is delivered through the weekly presence of industrial stakeholders. This makes students' challenge real.*

2. **Peer-Review Clusters**. Every week, students are called to review each other's assignment in a collaborative "cluster", i.e. a set of teams. Every cluster is structured as follows:

   (a) During the first week of the course, after the theory session, teams are arranged in groups of three (i.e. a cluster). The cluster remains the same for the whole course.

   (b) Each cluster is coordinated by a member of the teaching staff.

   (c) Weekly, the day preceding the peer-review session, every team shares the current, partially worked out version of the weekly assignment with the rest of the clustered teams.

   (d) On the following day, the clusters work in parallel peer-review sessions, during which the documents shared receive a review and feedback by the whole cluster.

   Pedagogic Value: *Software designers are constantly called to make, and consent with, design decisions [21]. Finally, design decisions themselves are constantly reviewed and upgraded, e.g. following bug-reviews and similar activities. In this year's edition of our software design course, peer-review sessions are a completely novel way for students to learn coping with different perspectives, gain a shared understanding of the problem(s) and collaboratively apply their own expertise/background to reach consensus over design issues. Teaching staff steers clusters into a shared understanding of issues and solutions by challenging decisions and asking clarification questions.*

3. **Stakeholder Reviews**. At the end of every week, industrial stakeholders are invited to take part to students' progress presentations. Students are encouraged to present least understood requirements and/or issues found during the week. The customer is encouraged to give feedback based on its expectations from the final product. Customer and students come to synergy and co-create best-fit solutions. Finally, students are provided with the perspective of: (a) presenting their work to practitioners in industry, e.g. to receive real-life feedback in real-life conditions; (b) working on follow-up studies; (c) winning a prize for the best project. Example prizes from last editions are visits/presentations to big industrial players (e.g. IBM) or free entry tickets to Amsterdam professional/networking events.
   Pedagogic Value: *Software designs have value insofar as their customers perceive it [7]. In addition, designers must account for their own decisions, since these can compromise overall system quality. Thanks to stakeholder reviews, students realise the importance of understanding the customer's needs and quality concerns, such that customer satisfaction can be achieved. In addition, students learn to be accountable for their decisions, such that they can convince the customer of the value of their proposed design solution. Finally, we have observed that the acknowledgement from the customer creates incentive and reward mechanisms for students [7].*

2.1. Student Population

Thus structured, our course is addressed to students currently undertaking the first year of their master study program. In previous editions, the course hosted around 30+ students every year. In this year's edition, the course hosted an increased number of students coming from different cultural, social and skills backgrounds, due to a new collaboration with the University of Amsterdam. While addressed mainly at students coming from a software engineering specialisation track, we welcome different computer science backgrounds, e.g. information- or knowledge management.

Course contents feature service-oriented analysis and design. Contents focus on how to analyse business and domain requirements to identify and design a correct and complete service offer for the identified requirements. Contents focus around the service-identification methodology proposed in [16]. The methodology, and the course in general, require heavy modelling and reasoning abilities. Therefore the course requires curricula with our Software Modelling course within them. Moreover, we strongly advise students to get acquainted with a UML modelling tool very early in the beginning of the course. This allows students to practice UML to quickly work out and refine a service-oriented design during our two-month intensive course. Table 2 provides more details on the student population. Column 1 names the characteristic for the student population, while Column 2 discusses our case. Contents and the intensive structure of our course, prepare students to attend classes focusing on Software Architectures and Distributed Software Systems.

2.2. Course Theory: Service Oriented Design

The theory part of our course focuses on *service-oriented design* i.e., designing a software that is *truly* service-oriented [11]. Our learning objectives in the theory part include:

- *Service aspects:* According to [3], the introduction of the SO paradigm introduced a shift in the way we conceive a "software system": from a large system to a set of small pluggable elements. We therefore, argue that the decision making of the designers should also be in-line with such shift. In this course we teach the students what makes a software truly service-oriented and what essential characteristics of services should drive their design decisions.

Table 2: Our Design Course, details on student population

Characteristics	Our Student Population
Previous Background	Students are expected to have solid knowledge of UML and similar modeling notations to support software design, e.g. using a service-dominant perspective.
Tutors per Student Ratio	We experimented with student groups of multiple sizes and found that a group of 50 students circa, can be managed easily with a single Course Lecturer and two teaching assistants.
Number of Teams	Students are normally arranged in teams of 4, if the student population is around 50. In general, we found as a rule-of-thumb, to keep the total number of teams (manageable with a Course Lecturer and two Teaching Assistants) around 20.
Project Perspectives	Students are encouraged to apply themselves on their project by stressing that their end-results are employed in a follow-up course in which they will (partially) implement their design.

- *Service analysis:* Our course teaches students how to translate business requirements to a number of business service candidates. In service analysis the students learn how to identify candidate software services as well as the participants providing or consuming those services.
- *Service design*: Many existing service design approaches assume that by identifying the candidate services, the design of services can be done in a straightforward manner. Unfortunately, devising a solid design that supports requirements and goals and is in-line with SOA principles (service aspects) is not straightforward and needs to be aided in a step-by-step manner. In this course we aim at supporting the students' line of reasoning in designing services. Within service design the students learn how to design and model a software services and/or reusing it to compose a service-based application.
- *Design reasoning*: Our course provides the theory about various design techniques are taught, including the use of software design reasoning tactics and documenting the architectural knowledge pertaining a software design solution.

  To pursue the above goals we have developed a service analysis and design methodology which have been taught in the last five editions of the course. This methodology, reported in [16], puts a special emphasis on modeling (in the most straightforward and pragmatic way possible) the elements that are left implicit in the existing service oriented notations and that in our experience aid students reasoning. For modeling purposes we reused state-of-the-practice notations like UML and SoaML, and extended them only where they revealed to be insufficient (for extensions see [16]). Figure 2 depicts the steps of our Service-Oriented Analysis and Design methodology. Across the years we noticed the necessity of providing the students with a template to document their design decisions. Without such a template, design decisions are documented in an unstructured way. This template is reported in [12].

2.3. Evaluation Scheme

Being a project-based course, the evaluation of our course was only based on the students projects. The project constituted 7 weekly assignments. Based on the content of each assign-

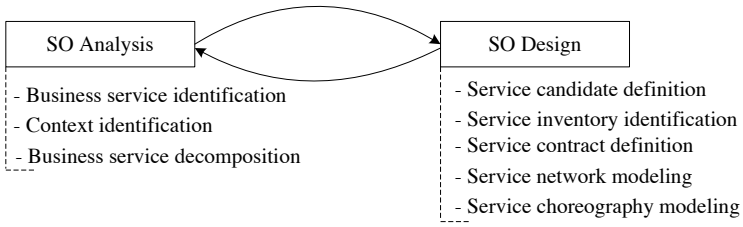

Figure 2: Service Analysis and Design Methodology in Our Design Course [16]

ment we had a different evaluation scheme. Those different schemes, however, can be generalized to the one shown in Table 3.

Table 3: An Overview of our Evaluation Criteria

Evaluation Criteria	Description
Accuracy and Correctness	The extent to which the analysis models/design issues/ service design models are accurate and correct
Completeness	The extent to which the analysis models/design issues/ service design models are complete related to the project scope and context
Relevance to project	The extent to which the analysis model/design issue/ service design model are relevant to project context and scope
Relevance to services	The extent to which the analysis model/design issue/ service design model address service aspects
Originality	The extent to which the students have used their creativity or reflection
Document internal consistency and traceability	The consistency between requirements, design, design issue and solution
Document quality	English, style, clarity

3. Discussion of Student Evaluations and Lessons Learned

After course completion, we asked all 49 students attending our course to fill a qualitative evaluation form. This featured two open questions. First, we asked students feedback concerning the peer-review clusters. Second, we asked students to evaluate the course, e.g. focusing on collaborative work based on the amount of work provided, time, resources and course-structure. In *italic* we point out lessons learned.

3.1. On the Structure of Peer-Review Clusters

All students have showed interest and active participation in the peer-review clusters. Of the 49 students attending, 42 students positively evaluated the use of peer-review sessions. Among them, 10 students did not have any criticism. In the following we discuss the constructive feedback we received for future improvement.

First, 3 respondents were indifferent to peer-review sessions and did not perceive any added value. In addition, 4 respondents considered peer-review a waste of time and resources. These students apparently perceived the added value of brainstorming together against design concerns

and problems but identified the need for more guidance and supervision in each cluster. Comments regarding the need for supervisors were varying, ranging from the presence of uncooperative behaviour to superficial/unproductive feedback to the being stuck on difficult issues for too long. *Peer-review clusters need to be steered by means of fixed and written guidelines.* In our experience, we found that students do not yet possess the critical thinking skills needed to constructively evaluate somebody else's work. To solve this issue, checklists can be used by the lecturer to drive the focus of reviewers to key issues based on the week's assignment. Each checklist item can address a critical learning point, i.e. something that is expected to be learned and that is critical for the success of the students' work.

Moreover, 17 respondents recognised the need for more structure within clusters, since their learning objective was not clear or impossible to reach without structured guidance. Finally, 15 respondents pointed out that there is a heavy overlap between the learning objective of peer-review sessions and that of stakeholder reviews. *Students must be provided with a clear-cut explanation of the learning objectives of each session, very early in the course.* Many students claimed that the overall structure and learning objectives of each phase in the course were not clear. Many also claim that their learning abilities were inhibited by this lack of vision. Providing a written, day-by-day roadmap to the students can help in clarifying the learning objectives of every phase of the course as well as giving the lecturer the possibility of verifying if all the learning objectives are met by the students after the course is done. This can also be used as a mechanism for grading.

3.2. On Collaborative Work

Whilst most of the students considered the collaborative work being valuable by enabling mutual learning, evaluations raised some issues.

First, 20 respondents considered splitting the work between team members being especially difficult. We found that such task management is carried out in an ad-hoc manner which possibly hinders the quality of work. For instance, some teams distributed the tasks based on different skills of students while some others assigned different portions of functionality to each member. Independently from the way teams split the work, all 20 respondents felt the need for help/supervision to enhance team collaboration. This suggests that *teams need some support on project management*.

Second, 9 respondents identified non-performing team members as a key hindrance for their collaborative work. They further emphasised that specific mechanisms are needed to deal with such problems, for instance peer-evaluation within the teams. While peer-evaluations can potentially improve this problem, we plan to address this problem using industrial practice. *Lecturers should steer peer-reviews, focusing on: (i) setting clear expectations, (ii) ensuring that non-performing members are held accountable, and (iii) setting internal deadlines for teams.*

3.3. Lessons Learned

In delivering SOD with the new structure we learned two key lessons. First, improving the social aspects of the course enabled us to manage an increased number of students (around 80) from many different backgrounds. Teams and clusters become increasingly self-managing as members' experience increases. This leaves the tutor to wander around clusters to steer and guide each cluster's work, or evaluating students performance. Second, the new structure allowed us to blend multiple backgrounds, cultural extractions and experience levels together. The very collaborative nature of the peer-review clusters allowed students to increase their performance matching that of others, or to compensate for the shortcomings of students with less experience. We conclude that the benefits of peer-review clusters far outweighs the invested organisation effort.

4. Related Work

Teaching software design has been extensively investigated and discussed in the literature. Many approaches, like in [9, 20], have mostly focused on teaching technical competencies and skills, while recognizing the intrinsic creative nature of software design. Brignall and Ramaswamy [5] study the differences between on-line versus on-campus teaching styles in an analysis and design course, and discuss the influence of the two teaching styles on social aspects like student cooperation, active learning, communication and teamwork.

Similar to Jarzabek et al. [15], Carrington [8] addresses the problem of team collaboration by offering the students with mechanisms to freely organize their own work within a team and experience with responsibilities and accountability. While useful for grading, this approach is general for any software engineering activity. We rather aim at focusing on the interplay between social aspects and technical aspects recurring in software design.

In [6] Budgen introduces a deep overview of software design and its implications on the quality of software intensive systems. Many other works explore both in education and practice the devices of software design, e.g design patterns in [14], or design notations applied to real-world scenarios, e.g. [1]. We inherit from such works the challenges of software design. In our work however, we do not introduce any challenge or device to tackle them, we merely introduce a course structure that can teach students both challenges and productive approaches to tackle them in practice.

Other works such as [10], explore the cognitive and human challenges that make software design very difficult and equally uncertain. Our motivation draws from this and similar works. We argue that software design is cognitively straining and the sum of many agreements (e.g. to satisfy many stakeholders and their concerns). In this vein, students must learn to confront many possible reviews, coming from many stakeholders with different backgrounds and technical expertise. Moreover, to be effective, this learning process must take place in the straining cognitive conditions of industrial practice, dense with deadlines and restrictions.

However, the only work directly related to ours is [17], where the authors introduce an approach to teach software design using the scaffolding pedagogical paradigm. Similarly to us, the authors stress the need for cooperative learning and heavy interaction among students. However, while they use cooperative learning as a tool to achieve learning objectives, we let cooperative learning emerge as a result of our approach dense with social connotations. In our approach, students learn to tackle design problems by understanding those of others. Also, students learn new ways of improving their designs by looking at innovative ideas from different backgrounds.

5. Conclusions

Software design is a social activity. While lectures in software design have typically focused on theory and practice alone, a strong social connotation is needed. This paper introduces our software design course that features social aspects as important as theory and practice. We discuss observations and lessons learned using our experience in teaching the course as well as students' evaluations. Our design course efficiently delivers a balance between theory, practice and social aspects of software design. However, we also learned that social engagement needs to be structured in a more systematic way. In addition, tasks and work management should be steered by lecturers explicitly. In the future we plan to improve the usage of peer-review and collaboration mechanisms, e.g. by using review checklists within peer-review clusters. More study also needs to be invested in confirming the outlines and lessons learned as part of the work contained in this paper. More in particular, both quantitative and qualitative studies should be invested in confirming the quality aspects of our course, e.g. by interviews to students with a statistically relevant perspective.

References

[1] T. Baar. Improving software engineering education by modeling real-world implementations. In *In EduSym2012 - 8th Educators' Symposium @ MODELS 2012: Software Modeling in Education*, 2012.

[2] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Series in Software Engineering. Addison Wesley, Reading, MA, USA, 1998.

[3] A. E. Bell. From the front lines: DOA with SOA. *Commun. ACM*, 51:27–28, Oct. 2008.

[4] E. Bittner and J. M. Leimeister. Why shared understanding matters - engineering a collaboration process for shared understanding to improve collaboration effectiveness in heterogeneous teams. In *46th Hawaii International Conference on System Sciences (HICSS)*, Maui, Hawaii (to appear), 2013. 352.

[5] T. Brignall and S. Ramaswamy. Impact of different teaching paradigms on student learning in technically intensive courses: Observations from a software analysis and design course. volume 0, page 100. IEEE Computer Society, 2003.

[6] D. Budgen. *Software design.* International computer science series. Addison-Wesley, 1994.

[7] J. Cameron and W. D. Pierce. Reinforcement, reward, and intrinsic motivation: A meta-analysis. *Review of Educational Research*, 64(3):363–423, Fall 1994.

[8] D. Carrington. Teaching software design and testing. In *Frontiers in Education Conference, 1998. FIE '98. 28th Annual*, volume 2, pages 547 –550 vol.2, nov. 1998.

[9] A. Cowling. Stages in teaching software design. In *Software Engineering Education Training, 2007. CSEET '07. 20th Conference on*, pages 141 –148, july 2007.

[10] F. Dtienne. *Software design cognitive aspects.* Practitioner series. Springer, 2001.

[11] Q. Gu and P. Lago. Exploring service-oriented system engineering challenges: a systematic literature review. *Service Oriented Computing and Applications*, 3(3):171–188, 2009.

[12] Q. Gu, P. Lago, and H. van Vliet. A template for SOA design decision making in an educational setting. In *Software Engineering and Advanced Applications (SEAA), 2010 36th EUROMICRO Conference on*, pages 175–182. IEEE, 2010.

[13] S. M. Hord and W. A. Sommers. *Leading professional learning communities : voices from research and practice / Shirley Hord, William A. Sommers*. Hawker Brownlow Education, Moorabbin, Vic. :, 2009.

[14] H. Huang and D. Yang. Teaching design patterns: A modified pbl approach. In *Young Computer Scientists, 2008. ICYCS 2008. The 9th International Conference for*, pages 2422 –2426, nov. 2008.

[15] S. Jarzabek and P.-K. Eng. Teaching an advanced design, team-oriented software project course. In *Software Engineering Education &Training, 18th Conference on*, pages 223 –230, april 2005.

[16] P. Lago and M. Razavian. A pragmatic approach for analysis and design of service inventories. In *Service-Oriented Computing - ICSOC 2011 Workshops*, volume 7221 of *Lecture Notes in Computer Science*, pages 44–53. Springer Berlin Heidelberg, 2012.

[17] S. P. Linder, D. Abbott, and M. J. Fromberger. An instructional scaffolding approach to teaching software design. In *In CCSC06. Consortium for Computing Sciences in Colleges*, 2006.

[18] N. Medvidovic. Software architecture challenges and opportunities for the 21st century: dynamism, mobility, autonomy, services, grids, and clouds. In S. K. Aggarwal, T. V. Prabhakar, V. Varma, and S. Padmanabhuni, editors, *ISEC*, page 1. ACM, 2012.

[19] M. Mrtensson. The role of the software architect: Caring and communicating. *MSDN*, page 3, 2008.

[20] J. Naveda. Teaching architectural design in an undergraduate software engineering curriculum. In *Frontiers in Education Conference, 1999. FIE '99. 29th Annual*, volume 2, nov. 1999.

[21] D. L. Parnas and D. M. Weiss. Active design reviews: principles and practices. In *Proceedings of the 8th international conference on Software engineering*, ICSE '85, pages 132–136, Los Alamitos, CA, USA, 1985. IEEE Computer Society Press.

[22] D. A. Tamburri, P. Lago, and H. van Vliet. Organizational social structures for software engineering. pages 1–35. ACM Digital Library, ACM Computing Surveys, 2012.

[23] H. van Vliet. *Software engineering - principles and practice.* Wiley, 1993.