Collaboration Support in an International Computer Science Capstone Course

Robert Adams[1] and Carsten Kleiner[2](✉)

[1] School of Computing and Information Systems, Grand Valley State University, Allendale, MI 49401, USA
adams@cis.gvsu.edu

[2] Department of Computer Science, University of Applied Sciences & Arts Hannover, Hannover, Germany
carsten.kleiner@hs-hannover.de

Abstract. Many computer science programs require some kind of culminating "capstone" course where students demonstrate skills learned in their CS curriculum. These capstone courses typically focus on the technical skills that students have learned, but one skill that is becoming more critical in our ever-global world is the ability to work in an international setting. Specifically, working on a team with students from a different country and/or culture. Over the past three years we have successfully offered an international capstone experience requiring students to work on a virtual team with students from a different country. For instructors, the primary challenge in offering such a course is collaboration between the instructors prior to the start of class. For students, the primary challenge is collaboration while the course is underway. This paper examines how we support instructor/instructor, instructor/student, as well as student/student communication and collaboration. This paper highlights how current web-based technologies provide support for collaboration. More specifically at least shared online storage for standard documents such as text or spreadsheets as well as video conferencing facilities are required for all the relations. Additionally, shared code repositories (and corresponding presentation) as well as online and offline messaging is necessary for a satisfactory experience. Software project management platforms provide additional important features. We show how technologies such as GitHub, Google Drive, Google Hangouts and Redmine provided the necessary support in several projects. At the same time other project teams have employed other similar technologies successfully as well. Our hope is that others are encouraged to attempt similar international efforts in order to broaden their students' non-technical skills as all the technologies are already in-place, well-known and stable, thus lowering the barrier for these important international experiences significantly.

Keywords: Computer science capstone course · Collaboration support · International software project · Software engineering in virtual teams · Collaboration tools

G. Meiselwitz (Ed.): SCSM 2016, LNCS 9742, pp. 313–323, 2016.
DOI: 10.1007/978-3-319-39910-2_29

1 Introduction

Many computer science and software engineering programs require some kind of culminating *capstone* course where students demonstrate skills learned in their subject-specific part of the curriculum. Many of these courses focus on a large software development project, and require students to apply their technical (*hard*) skills as well as their non-technical (*soft*) skills to a specific project or implementation. Successful completion of the capstone course demonstrates that students have learned, and can successfully apply, a large subset of the skills learned in prior CS courses. In addition, this course is usually a good dry run for how the future graduates will find their professional workplace.

Although these capstone courses typically focus on the technical skills that students have learned, one skill that is becoming more critical in our ever-global world is the ability to work in an international setting. Like many other CS topics, the meaning of "internationalization" in the curriculum varies. On the technical side, internationalization can mean adapting software to support other languages (e.g., alternate keyboard layouts, and displaying program strings in a different language). As this aspect is very frequently found in practice, there are technological solutions for this issue. The particular solution depends on the specific development language and system architecture, but important foundation to use these properly has usually been laid in technical classes throughout the curriculum. On the non-technical side, internationalization can mean equipping students with the skills necessary to operate within a global company (e.g., managing time zones and working on physically distributed teams). Issues in this area consist of organizational as well as social challenges. It is this latter aspect that we focus on in this paper.

Over the past four years we have successfully offered an international capstone experience as joint effort between Grand Valley State University, Michigan, USA, and the University of Applied Sciences & Arts, Hannover, Germany. Our joint capstone requires students to work on a virtual team with students from a different country. Naturally, offering such a course requires overcoming several challenges for the instructors, as well as for the students. For instructors, the primary challenge in offering such a course is collaboration between the instructors prior to the start of class. For students, the primary challenge is collaboration while the course is underway. This paper examines how we support instructor/instructor, instructor/student, as well as student/student communication and collaboration in an international setting.

This paper is organized as follows: we start with a brief general overview of the course design and how it fits into the two different programs. Thereafter we elaborate on the specific challenges regarding the different collaboration models required throughout the course. Then we suggest technological solutions that help remedy most of the collaboration issues and explain non-technological solutions that are helpful on top. All the suggestions focus on the typical university course setting where budgets are extremely limited. After that we review other work on international capstone courses before we finish with a conclusion and outlook.

2 Course Overview

Before discussing collaboration issues, we provide an overview of the course structure. In the next section we discuss particular challenges of collaboration. Richards [12] identifies several key design choices faced by project-based courses (international or not). Here, we describe and justify our particular pedagogical choices.

One important consideration is team size and team formation. Our choice was for teams of about four students. Our experience confirms that of [11] and [17]: teams with fewer students limit the dynamic, collaborative experience that we wanted students to achieve, and don't accurately reflect the challenges of working on a globally distributed team. Likewise, teams with more than about six students are often overwhelmed by team management issues, especially given the short total time frame of one semester mandated by our curricula. For example, with more than four students, it is often difficult (if not impossible) for students to coordinate their school/work/life schedules to find a common meeting time. We've found that approximately four students presents the right balance between challenge and frustration.

There is broad discussion on whether students should form their own teams or if the instructors should do so. In contrast to [11] and [12], we decided to give students latitude to form their own teams, typically based on the interest in a shared project idea. Students share possible term-length project ideas with each other and from those ideas, self-form teams. The team formation process is moderated by one of the instructors who, after publishing the pool of project ideas, collects student votes for three subjects from this pool. Based on these votes *good* project ideas to be realized will be selected and students assigned. In our experience it is almost always possible to only execute projects with at least four votes while accommodating at least one of the three choices of each student. Even though being able to form their own teams may not accurately reflect the situation in their future professional lives, we note that it provides a boost in motivation due to a shared common interest in the project idea. We have noted that students want to work on their projects because they feel they *own* the project, rather than having it imposed on them. Also having team members that fit well on an emotional level significantly improves motivation. High motivation is extremely important in any software development project [19] and can be difficult to achieve by the instructor if the general project ideas do not suit students.

After teams are formed, teams work cooperatively to create a working software project, as well as several document artifacts. All of these artifacts (and thus the development process) together with due dates are defined in a shared effort by the two instructors before the class starts. Each team reports to only one of the two instructors, so that additional effort on the instructor side due to the international setting is minimized and the students learn to work with a remote product owner. Teams first produce a project prospectus describing their project. Project ideas are generated by the teams themselves, although there is nothing to prevent the instructors from using industry-sponsored projects, or even in-house projects that have been added to the pool of project ideas. Instructors evaluate the prospectus based on content and scope. In terms of content, instructors ensure that the project involves aspects from several different areas of computer science. The project is meant to be a *capstone* experience, and cannot be limited to a single domain of computing (e.g., solely graphics or AI). The prospectus is also evaluated in terms of scope: ensuring that the project is ambitious enough to require the team the entire semester to complete.

After the prospectus is approved, teams conduct a feasibility study to generate a feasibility report and initial burndown chart based on an initial set of items for the product backlog. The feasibility study requires teams to conduct initial research on those areas of the project whose solutions are currently unknown (novel algorithms, data structures, languages, APIs, etc.), and present initial solutions that were discovered. Teams must demonstrate that their target platform, language, libraries are installed on their development machines. Because of the distributed nature of the course, teams must also identify how and where course artifacts will be created. Finally, teams must create a burndown chart based on initial estimates for the items in the product backlog showing the intended timeline for development taking the general course schedule into account.

During the project development phase, teams are required to complete a certain set of items from the backlog to produce a working part of the final product after every sprint. In recent years we have split the semester into 4 sprints, each about 3 weeks in length. After each sprint, teams will generate a sprint report and meet with their instructor to discuss progress to date, and challenges faced. Another important part of every sprint (as well as the preliminary phases) is a peer evaluation that is to be submitted to the instructor individually by each student. The peer evaluations help instructors to identify potential problem spots where intervention by the instructor may be necessary. This is particularly important for issues within the project teams, especially on the social level, that do not show in the intermediate result.

The final result of the term is presented to the entire class. Teams collaboratively create the presentation slides, and must carefully orchestrate the order in which the team members will speak. Besides the course artifacts themselves, teams must also submit periodic peer evaluations. One way of supporting the student's evaluation of their team members is the mandatory use of a version control system [6]. Using the *blame* feature available in systems, instructors can determine how much each student is contributing to a solution. This information helps to support or refute student comments about their peers.

Grading throughout the semester follows a predefined and publicly available grading rubric that has been set by the instructors in a common effort. All grading is done by the instructor that a team reports to with some commonly graded teams in the first two years to align application of the rubrics by the different instructors. The final grades from the percentage results of the grading rubrics, however, are determined by the local instructors in order to align these grades with local specifics of the programs regarding grades.

3 Collaboration Challenges

As noted above, there are several points of collaboration between student/student, instructor/instructor, and instructor/student. In this section we make explicit the challenges and requirements of collaboration at each of these points.

Overall, instructors must agree on the purpose of the course. Without this fundamental agreement, a student's experience in the course would depend highly on who their instructor is. Our goal is to mitigate instructor differences by ensuring a common vision.

Between the instructors, much of the collaboration happens before the class even begins. At minimum, instructors must co-create a syllabus for the course outlining the following items:

- What student learning objectives are central to the course? As with any course in higher education defining the learning objectives for the students is the first central issue to fix. Usually learning objectives for these courses at each individual institution have already been set externally when originally designing the course. The specific collaboration challenge for the instructors is thus to check whether there is a sufficient overlap between those objectives. This can be done virtually (e.g. video conference) but a personal discussion is usually favorable.
- What content do the instructors deliver in class? It is important that all students on the virtual teams start with as equal knowledge regarding software engineering projects as possible. Note though that there is substantial difference in both technical as well as organizational skills even within a single local class. So, at least the same level of variation is to be expected in distributed teams as well. Focus here should be on a common understanding of the development process itself. This collaboration challenge can be solved by exchanging information about prerequisites as well as material to be shared in class. Shared document storage is usually sufficient.
- What software development model/process will students use to create their projects? Similarly to the previous item all students need to work on the same common process model with the same timeline. This can only be achieved if previously agreed and defined by the instructors.
- What deliverables will students be submitting for grading? There should be common understanding and deadlines regarding deliverables throughout the project. As discussed in the previous issue the same development process will be used, thus it should be rather simple to also define common deliverables to be submitted by the students. Usually most of the deliverables are immediate consequences of the chosen process model. Some additional deliverables specific to university courses as opposed to professional software development should be added based on a common decision (i.e. frequent formal peer evaluations).
- How will student deliverables will be graded? Using what criteria? For each of the deliverables mentioned above there needs to be a common set of criteria that will be used to assess the quality. In addition, there should be a common grading rubric (usually percentages have been proven useful). Also, common agreed weights should be used for each of these criteria as well as for overall grading combining grades for all deliverables.
- When will student deliverables be due? After deciding on the process model, deliverables and overall timeline, common due dates for each of the deliverables should be defined. Note that these due dates have to take a potential time difference into account. Due dates for deliverables should be unique for the whole course whereas

presentations may be scheduled individually for distributed teams as close to the presentation times of local teams as possible.
- How will teams be formed and by whom? This is one of the more difficult issues to solve as there are different ways to do this, each with specific advantages and drawbacks. On a meta level it is important that the instructors agree on the process of team formation and decide who will make ultimate decisions in the case of complaints. To a certain degree the formation process depends on other parameters of the course (self-defined projects vs. externally provided, keeping groups of friends on a team vs. purposely separating them). In our setting the process as described above has proven to work well.

After the course begins a whole new set of collaboration challenges arise between the instructor and students, and between the students themselves. Between the instructor and students the main challenges are:

- How to find common meeting times for instructors and students to meet? As teams are distributed, there can be no regular meetings according to the regular university schedule. Thus each team's meeting times with the instructor have to be set individually. As this process tends to be difficult because it extends regular class hours, all parties will need to bring a lot of flexibility. Also, in order to keep organizational overhead minimal an agreed process on how to set these times is necessary.
- How to conduct meetings with students when they are not in the same time zone or country? Whereas the previous challenge already holds true for any distributed team, the international setting with different time zones makes finding common meetings times even more difficult as the number of constraints increases significantly. Thus, a strict process and time-wise flexibility are even more important.
- How to keep track of a team's progress throughout the semester? The deliverables and due dates selected by the instructors should be rather fine-grained in order for the instructor to keep track of the team's progress continuously. In the distributed setting the deliverables are usually the only chances for the instructor to track progress as frequent in-class meetings or informal meetings are not suitable. It is also important to keep an eye on each student's individual participation in the teams. On the other hand the organizational overhead for the instructor must remain manageable.
- How to facilitate communication about a team's performance throughout the semester? Grading of each of the deliverables throughout the project is only helpful if the team gets feedback on their performance shortly after submission. This is particularly important in case of imperfect deliverables to let the team know how to improve the quality for the upcoming artifacts. As this cannot be done in a personal meeting as usual, a way to communicate assessment needs to be defined that is both quick and easy to use.

Finally, the following challenges arise between the students on a team. These are probably the most important challenges as there much more students involved in the class than instructors. In addition there is no natural hierarchy within the student teams so that no ultimately deciding instance is present. Thus easy to use and commonly agreed solutions for these challenges will have to be available in order for the student teams to function properly.

- When and how do the students find common meeting times? Every project work requires some kind of synchronous interaction between team members. In a distributed team apart from local meetings, common meetings can only be virtual and have to observe many time constraints from each participant as well as from the international setting in general. Thus finding suitable meeting times is a real challenge with distributed student teams.
- How are they supported in a continuous team building process? Most components of traditional team building processes are centered on common experiences. These are not really available with distributed teams. It is almost impossible to hold a physical kickoff meeting or perform classical social events supporting team building. New ways of team formation both on the social as well as the organizational level (i.e. roles on the project team) have to be found. Typically those will at least slow the process down significantly which can be problematic given the time constraints of a semester.
- How are they sharing project artifacts (e.g. documents, source code, product backlog)? Throughout the project there is a need to share artifacts within the project team. Those range from simple documents stating ideas or project aspects over documents used as deliverables in the defined process (e.g. sprint reports, product backlogs, burn-down charts) to the source code of the product developed by the team itself. In a distributed team there needs to be a way to provide these sharing capabilities with minimal overhead. As many real-world development teams also operate in a distributed manner this challenge is not as big as it seems. Many professional development teams have faced similar issues and thus well-proven technological solutions are available (see next section).
- How/when are they meeting as a team? Apart from the first challenge which dealt with finding a concrete common meeting time, a related challenge is whether and how often they can be meeting as a team at all. The frequency will definitely be reduced when compared to a local team, so that working style in the project will have to be adjusted to fewer team meetings. This affects many aspects of project work from amount of independent work to level of documentation and ways to make decisions on the team.
- What software development model do they use (e.g. individually, peer programming)? Related to the previous challenge there is also an impact of the distributed team on the software development model. The number of choice is reduced as e.g. pair programming can only be performed by the local parts of a team. This in turn poses a challenge and restrictions on the separation of work within the team which may have to follow geographic aspects more than technical ones.

4 Collaborative Tool Support

According to [20] tools supporting collaboration of student development teams can be grouped into four categories: communication, goal tracking, information distribution (e.g. document sharing and management) and change management. We will focus on the first three of those as change management only impacts the instructor-instructor relationship over several instances of a course and is thus of limited importance for the student teams.

Given the set of challenges described in the previous section, we now discuss tools that were used to address those challenges. As mentioned previously, much of the collaboration between instructors happens before the class begins, and falls into two general types of collaboration: real-time synchronous communication, and asynchronous document creation. Agreement on the general vision and format of the course is best handled synchronously. Face-to-face communication is the best avenue for this, but in this case one instructor was in Germany while the other was in the United States. Therefore, a synchronous communication program like Skype, FaceTime, or Google Hangouts can be used, or if nothing else, a telephone. With synchronous communication instructors can quickly share their ideas for the course, their concerns, and work with the other instructor to create a shared vision for the course. Naturally, this could also be done via email, but the turnaround time makes it less than ideal.

After the general goals of the course are agreed upon, detailed course documents need to be created. We have discovered that an asynchronous document creation system works very well for this. In our particular case, we used Google Drive to collaboratively create course documents and spreadsheets. The course documents included a syllabus and course schedule. The syllabus not only describes the learning outcomes, but the course deliverables, software development methodology to be used, and grading rubrics. Asynchronous collaboration is appropriate because it gives instructors a chance to work on different parts of the documents simultaneously, as well as to thoughtfully respond to edits made by the other instructor. Also, Google Drive's commenting features, as well as its ability to track changes without having to email documents back and forth, ensures that both instructors always have access to the latest version of a document.

During the semester communication between the instructors diminishes except when clarification is required on grading rubrics or how to handle a specific situation with a team. In these cases, our experience is that email is normally sufficient, although a phone call or Skype session to talk through a situation is sometimes warranted.

After the course begins instructors and students start collaborating in earnest. Once again, the challenges fall into two types: the need for real-time synchronous communication, and the need for asynchronous document/artifact sharing. One of the initial challenges is trying to find a common time when teams and their instructors can meet. This is normally accomplished through open source meeting scheduling tools like Doodle [doodle.com]. Because instructors and students are not located in the same country or time zone, meetings must be held online. We have had success using tools like Skype and Google Hangouts, both of which allow for multi-person "conference calls".

Document and artifact sharing (reports, spreadsheets, graphs, etc.) requires some kind of shared storage. We have successfully used Google Drive and Dropbox for this, although some teams choose to host their own websites. Sharing source code is a necessity, and we mandate the use of git. We chose git because it is free, and an industry standard. Some teams choose to share documents by uploading them to their GitHub repositories, as well as using GitHub's wiki feature to create documents, thereby ensuring that all course artifacts are located at the same location.

For our course we also require all students to keep track of the time spent on their project. Many accomplish this by uploading a document or spreadsheet as described above. Other students do this by using an online time tracking tool like Toggl[1]. Toggl allows users to start and stop timers, to specify what was being worked on, and then to generate reports showing time-on-task.

Finally, students on the same team face collaboration challenges, especially when they are not located in the same country. However, even students at the same university often have collaboration challenges due to work schedules outside of school, or family commitments. Like the collaboration between the instructors and teams, collaboration between students can be characterized as either synchronous or asynchronous. Many student teams reported that both Skype and Google Hangouts worked well for them, allowing them to hold live meetings from any location (one student reported that he "attended" a team meeting while his car was stuck in a snowbank on the way to campus). Of course, email also remains a primary tool used to convey information that may not be time-critical. Students also have a need to share artifacts (primarily source code), and this is easily accomplished using tools such as Google Drive and git as described above.

In general, integrated project management systems for software development teams such as Redmine may also provide a good choice as they provide solutions for a fair number of the challenges. They typically focus on the software project specific issues (e.g. wiki, document sharing, version control, issue tracking, time recording), though, still leaving the need for tools solving the communication challenges discussed above.

5 Related Work

The general idea of using distributed student project teams has been documented in the literature for several years, the first notable work being [5] with [1, 3, 10, 17] being more recent. The ideas and challenges mentioned in those papers have been leveraged in our course by re-using some of the ideas and purposely setting it up differently in other aspects. [18] also describes a similar effort compared to ours, but this paper focuses more on the specific collaboration challenges and how those can be solved with recent technological tools that have not been available 10 years ago.

The works just mentioned provide an overall description of course pedagogy. More specific work focuses on overall challenges for distributed teams [16], communication skills [9], student team organization [2], software testing techniques [13], software tools [7, 6], cultural factors influencing success [15], use of Agile and scrum methodologies [14], and student motivation [4]. [20] is a recent paper that categorizes tools used in a software engineering course, but does not elaboration on what those tools were. As discussed above [11, 12] discuss the student team formation process which we did not use here as our setting is different.

Some general work on effective tool support for distributed software development teams away from the University setting also exist, e.g. [8] discussing software quality and [19] explaining how to raise developer motivation in such teams. More papers exist,

[1] http://www.toggl.com.

but many are of limited applicability to a university student project as some of the key setup parameters are different (e.g. need for tools complying with no budget, specific student-instructor situation, externally fixed schedule etc.).

6 Conclusions

Many computer science programs require some kind of culminating experience for students, and it is clear that gaining experience working in an international setting is becoming a critical skill. Several schools are incorporating global learning into their capstone courses, and it is likely that other schools are beginning to consider doing the same. The main challenge in developing a course around distributed virtual teams is ensuring sufficient support for collaboration. Over the past three years we have successfully offered an international capstone experience requiring students to work on a virtual team with students from a different country. During that time the number and sophistication of tools that support collaboration has grown dramatically, to the point that lack of tools is no longer a viable argument against virtual teams.

Our experience has been that the need for collaboration among the instructors and students falls into two broad categories: the need for synchronous communication, and the need for asynchronous document sharing. In both cases current open-source tools are sufficient to support distributed software development teams, and these supporting technologies have significantly improved the collaboration of the student teams.

Although a wealth of tools exist in each category, we have successfully run our capstone course using tools that are well-known, well-supported, and completely free. Specifically, both Skype and Google Hangouts are comparable in features and effectively support synchronous discussions. For asynchronous document sharing, Google Drive, Dropbox, and GitHub are more than adequate. Finally, integrated project management systems such as Redmine also provide support for document sharing challenges.

Our hope is that others are encouraged to attempt similar international efforts in order to broaden their students' non-technical skills. All the necessary collaborative technologies are already in-place, well-known, and stable, lowering the barrier for these important international experiences significantly.

References

1. Al-Janabi, S., Sverdlik, W.: Towards long-term international collaboration in computer science education. doi:10.1109/EDUCON.2011.5773118
2. Bruegge, B., Dutoit, A.H., Kobylinski, R., Teubner, G.: Transatlantic project courses in a university environment. doi:10.1109/APSEC.2000.896680
3. Ciccozzi, F., Crnkovic, I.: Performing a Project in a Distributed Software Development Course: Lessons Learned. doi:10.1109/ICGSE.2010.29
4. Clear, T., Kassabova, D.: Motivational patterns in virtual team collaboration. In: Young, A., Tolhurst, D. (eds.) Proceedings of the 7th Australasian Conference on Computing Education, ACE 2005, vol. 42, pp. 51–58. Australian Computer Society Inc., Darlinghurst (2005)

5. Daniels, M., Petre, M., Almstrum, V., Asplund, L., Bjorkman, C., Erickson, C., Klein, B., Last, M.: RUNESTONE, an international student collaboration project. doi:10.1109/FIE.1998.738780
6. Glassy, L.: Using version control to observe student software development processes. J. Comput. Small Coll. **21**(3), 99–106 (2006)
7. Gotel, O., Kulkarni, V., Scharff, C., Neak, L.: Working Across Borders: Overcoming Culturally-Based Technology Challenges in Student Global Software Development. doi:10.1109/CSEET.2008.16
8. Gotel, O., Kulkarni, V., Say, M., Scharff, C., Sunetnanta, T.: Quality Indicators on Global Software Development Projects: Does ‘‘Getting to Know You’’ Really Matter? doi:10.1109/ICGSE.2009.8
9. Johansson, C., Dittrich, Y., Juustila, A.: Software engineering across boundaries: student project in distributed collaboration. doi:10.1109/47.807967
10. Makio, J., Betz, S.: On educating globally distributed software development — A case study. doi:10.1109/ISCIS.2009.5291874
11. Oakley, B., Felder, R.M., Brent, R., Elhajj, I.: Turning Student Groups into Effective Teams. J. Student Centered Learn. **2**(1), 9–35 (2004)
12. Richards, D.: Designing project-based courses: with a focus on group formation and assessment. ACM Trans. Comput. Educ. **9**(1) (2009). Article 2
13. Richardson, I., Moore, S., Paulish, D., Casey, V., Zage, D.: Globalizing Software Development in the Local Classroom
14. Scharff, C., Gotel, O., Kulkarni, V.: Transitioning to Distributed Development in Students’ Global Software Development Projects: The Role of Agile Methodologies and End-to-End Tooling. doi:10.1109/ICSEA.2010.66
15. Swigger, K., Alpaslan, F., Brazile, R., Harrington, B., Peng, X.: The challenges of international computer-supported collaboration. doi:10.1109/FIE.2004.1408738
16. Swigger, K., Brazile, R., Serce, F.C., Dafoulas, G., Alpaslan, F.N., Lopez, V.: The Challenges of Teaching Students How to Work in Global Software Teams. doi:10.1109/TEE.2010.5508836
17. Tabrizi, M.H.N., Collins, C.B., Kalamkar, V.: An international collaboration in software engineering. Proceedings Of The 40th Acm Technical Symposium On Computer Science Education (SIGCSE 2009), pp. 306–310. ACM, New York (2009)
18. van der Duim, L., Andersson, J.: Good practices for educational software engineering projects. In: 29th International Conference on Software Engineering, ICSE 2007, pp. 698–707. IEEE Computer Society Press (2007)
19. Sach, R., Sharp, H., Petre, M.: Continued involvement in software development: motivational factors. In: Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM 2010). ACM, New York (2010). doi:10.1145/1852786.1852843. http://doi.acm.org/10.1145/1852786.1852843
20. Knutas, A., Ikonen, J., Ripamonti, L., Maggiorini, D., Porras, J.: A study of collaborative tool use in collaborative learning processes. In: Proceedings of the 14th Koli Calling International Conference on Computing Education Research (Koli Calling 2014), pp. 175–176. ACM, New York (2014). doi:http://dx.doi.org/10.1145/2674683.2674706