Knowledge Transfer in Collaborative Teams: Experiences from a Two-Week Code Camp

Terhi Kilamo, Antti Nieminen, Janne Lautamäki, Timo Aho,
Johannes Koskinen, Jarmo Palviainen, and Tommi Mikkonen
Tampere University of Technology
Department of Pervasive Computing
Korkeakoulunkatu 10
FI-33720 Tampere, Finland
firstname.lastname, antti.h.nieminen@tut.fi

ABSTRACT

Software engineering has both technological and social dimensions. As development teams spanning across the globe are increasingly the norm and while the web enables massive online collaboration, there is a growing need for effective collaboration tools. In this paper, we describe experiences on collaborative programming as a tool for learning software development. To investigate the nature of collaboration in software engineering education, we arranged a two-week-long course experiment where students used a collaborative online integrated development environment to create different kinds of web services. We present lessons learned from the experiment and discuss how collaboration can act as a tool for knowledge transfer among learners.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques - Programmer workbench.; D.2.6 [**Software Engineering**]: Programming Environments - Interactive environments; D.2.9 [**Software Engineering**]: Management - Programming teams.; K.3.1 [**Computers and Education**]: Computer Uses in Education - Collaborative learning.; K.3.2 [**Computers and Education**]: Computer and Information Science Education- Computer science education.

General Terms

Experimentation, Human Factors

Keywords

Case study, Software engineerin education, Collaboration

1. INTRODUCTION

In contemporary software development approaches a team of people with versatile expertize and backgrounds develop pieces of software as a joint effort. Approaches such as extreme programming practices, culminating in pair programming [3], global software engineering [20] and open source software [6] are examples of development settings, where teams work in a globally distributed environment using numerous tools that assist in cooperation. The fashion working together online takes place has been radically altered by the rising trend, where software and services are moving to the web, thus enabling massive collaboration of people from all over the globe. The so called Web 2.0 technologies incorporate collaboration and interaction allowing online cooperation with shared data and services. While software development has not been at the forefront of adopting the new Web 2.0 practices in terms other than providing implementation techniques [32], social tools such as wikis have been commonly adopted in software projects [41, 7, 24]. Likewise social media has been proposed to be used in the context of software development [4, 37].

The paradigm shift towards the web is reshaping numerous industries. One of these is education, where learning is inherently both the learner's own active effort and a social activity. Communities of Practice (CoP) [26, 40]– a group of individuals that share competence or profession – are an example of learning as an outcome of participation in a community. Social constructivism [33, 39], activity theory [9] and situated learning theory behind CoPs [26] approach learning as an activity that is social and, furthermore, context-dependent. As software engineering teams are globally located, understanding collaborative development, team work, and management of multi-site software projects is indispensable. While approaches to include these into software engineering (SE) curricula have been introduced [8, 14, 18, 34], education is still lacking in collaborative development on many fronts that are common-place in the software industry.

In this paper we introduce an approach to learning collaborative software development through a coding course case where learners developed web applications in collaborating teams with a shared web based integrated development environment (IDE). We argue that working in cooperative teams on similar projects promotes learning, and that knowledge is transferred among the learners through sharing of ideas, experiences and development outcomes. We discuss these through a course experiment – a two-week code camp, where a concrete software artefact was created. During the camp, European university students worked in self-organized teams

ICSE Companion'14, May 31 – June 7, 2014, Hyderabad, India
Copyright 2014 ACM 978-1-4503-2768-8/14/05...$15.00
http://dx.doi.org/10.1145/2591062.2591156

to developed online applications. The experiment was run as an intensive course during Summer 2013.

The rest of this paper is constructed as follows. Section 2 gives the background of our work. Social learning theories, teaching collaborative software development, and the idea of a collaborative IDE get covered with related work. Section 3 presents a concrete course setting and the collaborative approach experimented there. Section 4 gives the results of the experiment. Section 5 presents the evaluation of the approach, and Section 6 discusses future work. Section 7 concludes the paper with some final remarks.

2. BACKGROUND

The role of the learner is central in contemporary pedagogical approaches. They approach learning as an active effort where the learner builds new knowledge based on their previous life experience and knowledge. One key driver for learning is the intrinsic motivation of the learner. Appropriate teaching methods and learning environments aid in the process. Albeit positioning learners themselves in the center, learning is also a social process. The importance of this aspect is accentuated in software engineering by the nature of software product work done within a development team or a community, where members must communicate and share product artifacts and ideas throughout development. Examples of this kind of collaborative development approaches range from open source communities to in-house teams within a software company. Hence, combining social, collaborative aspects into teaching software engineering in general and software development in particular helps not only to aid learning collaborative skills but can on the whole help to reach better learning outcomes.

2.1 Social Learning

Modern constructivist learning theories [15] lift the learners themselves on the central stage. From this perspective, learning is a product of the learner's activities, and it is influenced by the learner's earlier experience and knowledge. Furthermore, learning is driven by the learner's own motivational factors. Among others, Hase and Kenyon [19] argue that self-determined learning approaches are the best fit for the modern world. The modern context calls for flexibility, creativity, the ability to work together in teams, and the ability to apply skills to different situations. Such skills can best be learned in flexible, learner-centric environments.

The socio-cultural learning theories [39, 33, 1, 9] express that people learn from each other through observation, interaction and communication. Viewing learning through its social aspect emphasizes the fact that no one lives in a vacuum. We are part of a community, organization, team – a group of people – where there is competence knowledge already established. A group of individuals that share competence or profession is a Community of Practice [40]. CoPs are seen as ideal vehicles for leveraging tacit knowledge and learning. Furthermore, CoPs explore the participation metaphor of learning; people learn, not necessarily intentionally, as they have a shared preoccupation and interact over time.

Through the social aspect, the idea of information or knowledge flow is accentuated. In his theory of learning as a networking process, Siemens [35] views networks as places in which knowledge resides. Drawing from this, a new learning theory, connectivism, is presented; the challenge today is not what you know but who you know, learning becoming a connection/network-forming process.

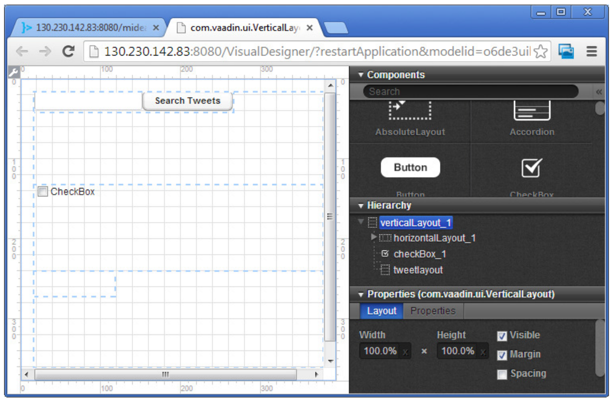
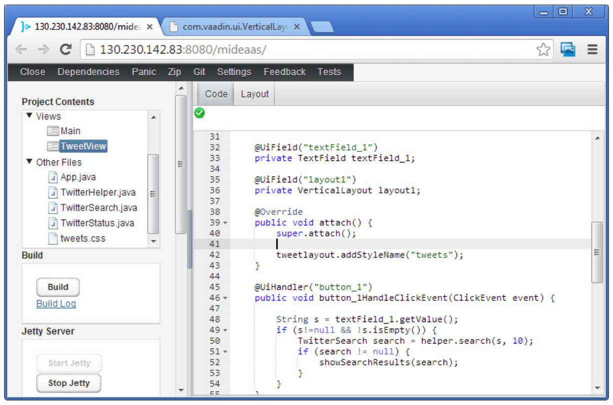


Figure 1: Screen capture of the visual user interface design tool of the development environment.



```

    @UiField("textField_1")
    private TextField textField_1;

    @UiField("layout1")
    private VerticalLayout layout1;

    @Override
    public void attach() {
        super.attach();

        tweetlayout.addStyleName("tweets");
    }

    @UiHandler("button_1")
    public void button_1HandleClickEvent(ClickEvent event) {

        String s = textField_1.getValue();
        if (s!=null && !s.isEmpty()) {
            TwitterSearch search = helper.search(s, 10);
            if (search != null) {
                showSearchResults(search);
            }
        }
```

Figure 2: Screen capture of the code editor of the development environment.

Both the learner's role and the social aspect of learning need to be recognized in teaching. Emphasis needs to be placed on knowledge flow, thus allowing learners to participate, observe, imitate, engage in conversation and network within the educational context.

2.2 Collaborative Development Environment

The environment, which the students used in this experiment, is a cloud based IDE called Mideaas[1]. The IDE has been developed at TUT and is integrated with a hosting solution into which completed applications can be deployed as services. The IDE is targeted to users who need a simple web-based tool for implementing and publishing web applications written in Java using Google Web Toolkit (`www.gwtproject.org`) and Vaadin [17], which is an open source web application framework for rich internet applications. No other tool except the browser is needed, implying that no installation is necessary in any phase of the development. Furthermore, the IDE supports concurrent editing; that is, multiple users are able to edit the code of the same project at the same and see each others edits in real time.

The IDE contains both a visual user interface design tool (see Figure 1) and a code editor (see Figure 2). With the

[1]`http://130.230.142.83:8080/mideaas/`

visual design tool, the user can create a new UI just by dragging and dropping elements and layouts, and setting their properties. The created UI is stored as an XML document which can also be manually edited in the code editor if needed. The web application is implemented in the Java language, and the code editor is used for the actual programming of the functionality. The visual designer is integrated with the code, e.g. automatically generating a click handler for a UI button.

Once completed, the developed web application can be deployed as an online service with a single click. Furthermore, the environment features a project-wide chat utility that allows developer discussions, expected to assist in interactions where all developers need to jointly agree on compilation or a test run for example. We have previously written about the code editor, as well as a more detailed description of the features [25].

The collaborative editing uses Neil Fraser's Differential Synchronization with shadows [12]. It is a robust and convergent collaborative editing algorithm with open source implementations available on various languages, including Java and JavaScript. Moreover, differential synchronization meets three basic demands often set for collaborative editing [38]: Firstly, it has high responsiveness. The edits can be done locally and only the differences are delivered to the server. Therefore the local actions are as quick as in a single-user editor. Secondly, it has high concurrency rate and multiple users can simultaneously edit any part of the document. Finally, it is able to hide communication latencies to some extent. Naturally, when latencies grow the conflict rate rises.

2.3 Related Work

Collaborative development facilities have been proposed ever since the famous Mother of All Demos by Douglas Engelbart back in 1968, where, as Wikipedia (`http://en.wikipedia.org/wiki/The_Mother_of_All_Demos`) puts it:

> "*The live demonstration featured the introduction of a system called NLS which included one of the earliest computer mouses as well as of video conferencing, teleconferencing, hypertext, word processing, hypermedia, object addressing and dynamic file linking, revision control, and a collaborative real-time editor.*"

Today, probably the best-known collaborative development environment is the Cloud9 system (`https://c9.io/`), which shares many features with our work including some running code as well. On the research front, the most prominent candidates are Collabode [16] , a real-time collaborative environment that aims at supporting novel development models to advance close collaboration, and Saros [31], a real-time collaborative Eclipse plugin targeted for distributed pair programming. Instead of using these existing tools, we used our own design in order to simplify data collection and processing, which could easily be added into our implementation.

Collaborative aspects have also been addressed in the field of software engineering education, but not in the same way we have addressed it here. An earlier paper [21] approached teaching collaborative SE development via a student-centric learning environment. The environment, KommGame, mimics real world collaborative project while adopting the idea of reputation systems in order to enforce the social context. There the focus was more on the social community aspects whereas here we focus on real time collaboration and knowledge sharing. On a similar note on web based collaboration tools, Minocha et al. [29] utilize a wiki as a shared authoring tool. Cooperation work over a wiki differs from the real time code development used as the platform here, and furthermore does not focus on code development. Favela and Peña-Mora [10] have proposed the use of various online tools for managing a collaborative software engineering course. The tools that were used reflect those that are commonly used in industry, and thus do not address collaborative coding per se. in the same fashion we are proposing. In addition, Burnell et al. [5] have experimented with distributed multidisciplinary environment, which is common in software engineering. Again, our approach where collaboration takes place at real time at code level is more directly targeted for developing the running software, and viewing it as a social learning environement, instead of managing the process of creating it. Collaborative aspects are also approached using OSS as a collaborative learning environment [36, 13]. In these experiments, the goal has been mainly in addressing the fashion software engineering takes place in open source communities, whereas we have a more general focus. To conclude, our viewpoint is not solely on team work and cooperation methods, but also on knowledge flow and learning through participation.

3. CODE CAMP EXPERIMENT

We investigated the role of collaboration in SE education by conducting an experimental course – a two-week long code camp. The research angle is on the lines of an empirical experiment [2, 23]. In addition, the participants background and experiences were explored with two personal opinion surveys [22]. Learners from each collaborative team were interviewed during the camp, and the progress of development recorded into log files for later study.

The pedagogical goal of our work is to determine patterns of interactions as students compose code. This would help us identify problematic ideas in both composing a program as well as testing the outcome. Therefore, we have collected an extensive amount of data from the experiment, and although this data will be analyzed in depth only later on, we also include our primary observations from this analysis in the paper. Due to the nature of our experiment where the students were co-located, the role of the web-based communication tools was not very significant: the group members could simply talk to each other. Thus, the most interesting aspect of the collected data concerns the actual code editing.

From the data we could answer questions such as did the users work on the same part of the code at the same time, or did they split their work; how equal was the amount of edits done by different users; or were there other collaboration patterns to be found. As a side-effect – discovered independently of the experiment described in this paper – it is obviously also possible for an educator to investigate how a project progresses and to see if the students are stuck on some particular detail in their projects. This particular observation will be later integrated to an existing online learning environment javala.cs.tut.fi, where such features can support learning outcomes [27].

3.1 Research Questions

The motivation for the case was to see how collaborative team work, tools and a self-organized social learning environment could support learning and help transfer knowledge between divergent learners; the social component of learning and knowledge flow are of interest.

The paper aims to answer two questions:
Q1: *How does collaborative team work facilitate learning SE development skills?*
Q2: *Does collaboration help transfer knowledge between learners?*

We answer Q1 through survey answers on the learners expectations and experiences after the code camp. The paper approaches Q2 through statistical analysis on the teams' work, log data and interview answers.

3.2 Organizing the Code Camp

The code camp was organized August 5-16, 2013 at Department of Pervasive Computing, Tampere University of Technology, Tampere, Finland. The learners were attending an advanced level Board of European Students of Technology (BEST, `http://www.best.eu.org/`) summer course. The 21 participating learners came from 14 different countries around Europe.

The development work was done in teams of three learners. The learners self-organized into development teams with the given rules. The teams were required to fulfill three requirements to ensure diversity: 1) all team members were of different nationality, 2) the team members were of more than one gender, and 3) at least one team member had to have prior programming experience. Teams of three were formed. The collaborative web-based development environment described earlier in Section 2.2 was used in the development.

From the organizing institute, two staff members acted as facilitators during the camp. They helped the teams to solve problems with the collaborative environment and project development in general. They, together with a third lecturer, also gave small background information tutorials on the central topics of the camp such as REST (see Section 3.2.1) and open development.

Table 1 presents the flow of the camp and the daily alterations, each day containing four to six hours of activities. The collaborative features are discussed in more detail in the following Section 3.2.2.

Table 1: Code Camp Activities

Day	Activities
1	Camp start. Tutorials on Vaadin, collaborative IDE and REST. Learners organize into teams.
2	Tutorial on the IDE. Teams start working on their projects.
3	Introduction to open development. Working on projects. Tutorial on developing a client.
4	Working on projects. First *group shuffle*.
5	Working on projects. Teams give an intermediate presentation on their progress.
6	Working on projects. Team member interviews.
7	Working on projects. Second *group shuffle*.
8	Working on projects. Team member interviews.
9	Finalizing the projects and planning the final presentations.
10	The teams finalize the projects and present the finished applications. Camp closing.

3.2.1 Projects

Each teams' task was to design and create a web application utilizing already existing web services APIs of their own choice. The small web applications utilized already existing web service interfaces, with RESTful APIs [11] being the advocated technology for accessing the services. The teams were given full freedom on the project theme, the API they wished to use, and the design decisions that were necessary to create a running web service. The chosen APIs were popular social media services such as the micro-blogging platform Twitter (`https://dev.twitter.com/docs/api`) and the social networking service Facebook (`https://developers.facebook.com/docs/graph-api`), the mapping service Google Maps (`https://developers.google.com/maps`) and the video streaming service YouTube (`http://www.youtube.com/yt/dev`). The projects themselves are discussed later in Section 4.1.

At the end of the camp the teams gave a demonstration on their results to everybody, sharing their experiences, successes and biggest obstacles during the camp. The camp facilitators gave the finalized projects scores estimating the overall complexity and quality of the work. The scoring was done as a tool for statistical analysis of the code camp data.

3.2.2 Collaborative Elements

There were two major collaborative elements in the experiment: the IDE used to develop the application and a group reorganization exercise called the group shuffle. As decribed in Section 2.2 the IDE made sharing of the project possible between the team members. All learners in the project team could see and work on the project simultaneously and edit the code together if necessary.

The group shuffle was conducted twice. The idea of the group shuffle is that each team chooses one person to stay with their home project and the rest of the learners move into another team. During the visit which is usually a few hours long the teams continue to work with the development work, thus causing the teams' knowledge to flow between the teams. The idea had been tried out once in a similar code camp which had given promising initial results [30]. In the experiment the two shuffles were ran three days apart first on day four and then again on day seven. The learners were free to choose which of the members went to another team. However, the person staying could not be the same one during both shuffles.

4. RESULTS

This section discusses the results of the learners' collaboration during the camp. Each learner was assigned an unique number for the purpose of keeping the results anonymous. The learners are identified by their number in the results here as well.

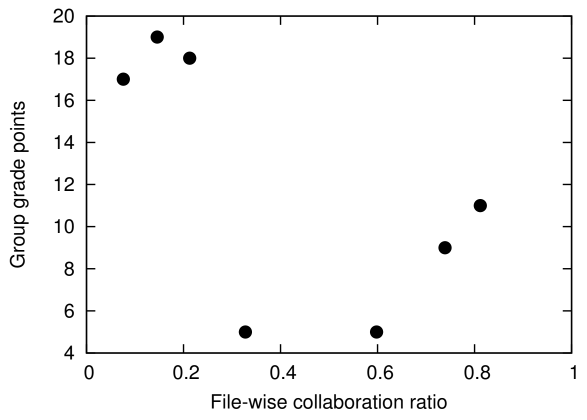


Figure 3: Group grade points in relation to the file-wise collaboration ratio for each of the groups in the code camp. The top three groups in the top-left corner have a low file-wise collaboration, implying the same files being edited rarely at the same time.

4.1 Project Results

Each of the groups was able to produce an application that somewhat accurately resembled their original plan. The projects ranged from multimedia blogs to bar navigation applications. Some other examples of the finished applications include a travel advisor utilizing a weather web service and a spin-the-bottle kind of game with Twitter integration. Data on the groups and their work is listed in Table 2. To evaluate the relative success of each group, their applications were ranked by independent assessments of the facilitators. The projects were judged based on the idea, the overall complexity, and quality of implementation, each facilitator placing the projects in order giving the best seven points, the second best six et cetera, the least succesful finally getting one point. Finally these were added together, hence the higher the points, the more successful the project was considered overall.

4.2 Collaboration and Modularity

It would be interesting to know how much there was actual collaboration in the project and how it affected the process. Some measurements for each group are shown in the Table 2. In the table, the second column indicates the facilitator grade points for each group as discussed in Section 3.2.1. The third and fourth columns give the average pre- and post-camp survey results that illustrate how high students estimate their knowledge level to be. Next we see the percentage of edits done by the main contributor of the project. Interestingly, in some groups the contribution is very high, at most 72%. Group A is nearly evenly distributed.

For the actual collaboration, we present two measurements. Let the amount of *project-wise* collaborative edits be changes done to the coding project at approximately the same time. One coding project is typically divided into multiple code files. Thus, let the amount of *file-wise* collaborative edits be the edits done in the same code file at approximately the same time. We chose 30 seconds to be the limit for the edits to be considered occurring at approximately the same time and, thus, be considered collaborative. Percentages of both kinds of collaborative edits are listed in Table 2. As can be seen, the percentage of project level collaborative edits ranges from 2% to 35%, showing that at least 65% of edits were done while no one else was editing the project. It is important to note that amount of work done alone varies a lot. Thus, we define *file-wise collaboration ratio* to be the amount of file-wise collaboration out of overall project collaboration. The rationale behind this measurement is to track how likely it is that two students working at the same time are actually editing the same file. This can also be seen as a measurement of code modularity of the projects.

The project-wise and file-wise collaborative edits do not seem to be good indicators for project success. Even so, the file-wise collaboration ratio has a moderate negative correlation of -0.63 (with p-value = 0.13 statistical significance) with the project success grade (the points grade in Table 2). This is illustrated more precisely in Figure 3. Here we note that the groups with higher grades (higher in the chart) seem to have small file-wise collaboration ratio (are more to the left), thus implying that some kind of collaboration may have negative effect on the group effort. This might indicate that the more successful groups were more able to coordinate their work in such a way that each member could work on their own part of the project collaborating on the file level only when code level communication was needed. The phenomenon goes back to traditional principle of code modularity: the code should be divided into naturally independent units that have minimal interdependency.

Another view of the project collaboration can be seen in Figure 4. They illustrate a typical project work flow and show collaboration in one project (group A in Table 2) over two two-hour long programming sessions. The edits are shown as data points, different symbols for different project members. The y-axis is the total number of characters in the project and the lines mark the file boundaries, the edits between two lines being in the same file. As can also be seen in this figure, the members mostly worked on their own part of the project, with occasional closer collaboration in the same files with other members.

4.3 Knowledge Transfer

To gather information regarding students' views, we interviewed a total of eight learners. There was a representative from each team and one additional volunteer. Based on the interviews, the teams had three approaches to team organization. The most common was to have a team leader, who was typically someone with most programming experience. One team (team A in Table 2) had a clear division of tasks where each team member had their own area of responsibility. The rest had a discussion and joint effort approach with divided workload but no leader. These teams also showed a somewhat unbalanced work distribution having one member with less interest in development.

The skill estimate in the surveys done in the beginning and end of the code camp also support the idea that more skillful students helped teaching the less skillful ones. The greater difference in the first survey test scores seems to correlate positively with higher gain in the knowledge. To be precise, in team wise comparison, the beginning survey score difference between the best one and the two others had a correlation of 0.70 (statistically significant with p-value = 0.08) with the increase between the two surveys for the two lower scoring team members. Such correlation could not be found for, e.g., the team wise highest beginning score

Table 2: Collaboration Data

Group	Facilitator grade points	Skill points in pre-camp survey	Skill points in post-camp survey	Edits by main contributor	Project-wise collaborative edits	File-wise collaborative edits
A	19	10.0	12.7	34%	35%	5%
B	18	10.0	13.0	72%	18%	4%
C	17	8.0	12.7	57%	13%	1%
D	11	9.7	13.7	61%	3%	3%
E	9	7.0	12.0	53%	29%	22%
F	5	10.5	10.3	49%	7%	2%
G	5	8.3	14.0	55%	20%	12%

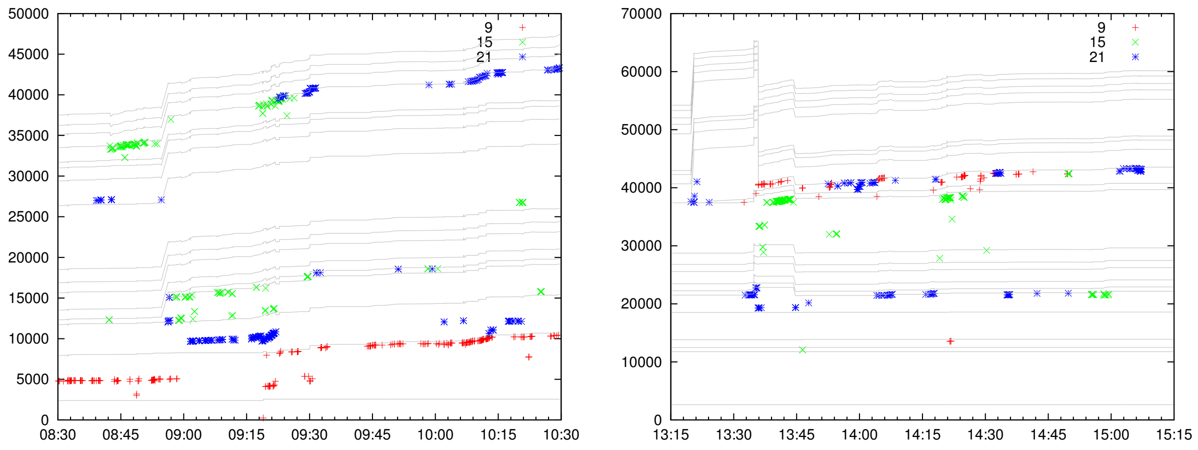


Figure 4: A typical two-hour long editing session from one of the teams. The edits of each team member are shown as different symbols. The y-axis presents the size of the project as characters. The gray lines are file boundaries; edits between two lines are on the same file. The order of the files is arbitrary.

which stresses the meaning of differing skill levels. However, we need to note that the sample is very small, only seven teams, and the survey test only asked students' own opinion on their skills. Thus, final conclusions should not be made.

The group shuffle highlighted the flow of knowledge from the ones who had more experience to the learners for whom this was one of their first web application development experiences. Analysis on the interview data enforces the notion of knowledge flow from the more experienced learners, thus helping other teams in areas they have knowledge gaps. Selected interview transcripts that report the trend (the number is the interviewees unique identifier):

6: *"It was a good thing. New members brought in new knowledge and knew some things that were very helpful in implementation."*

12: *"First shuffle changed groups. They explained what they were doing. [It] was a productive day. They were working on something similar and showed what the were doing. Was helpful for the team."*

5: *"Team got experience from the other team. Different working style can be brought from the other team if good practices were found."*

On the other hand, difference in knowledge is acknowledged in the interviews:

17: *"Tried to help the team but not sure if is was helpful. Got more help than gave. Difference in the knowledge."*

The interviews further showed that earlier knowledge indicates the role on the giving side of information.

15: *"More on the giving side of information"*

8: *"Gave information to the new group and not the other way around."*

5. DISCUSSION

Next, we will revisit research questions listed above in the light of the results obtained from the experiment.

Q1: *How does collaborative team work facilitate learning SE development skills?* The experiment gives promising results that working not only in self-organized teams but having additional collaboration between the teams helps the learners gain wider knowledge overall. From the software engineering perspective, the results enforce the value of divided responsibilities between developers. A high number of collaborative edits can hinder the progress of the overall project. A balanced work load also produced the most highly valued end result further supporting the notion.

Attention should also be paied to the change in the learners' knowledge estimates between the pre- and the post-camp surveys. Firstly, statistical analysis shows that team work does facilitate learning SE development. Secondly, there is a drop in the post-camp estimates in one case. This suggests that not only learning but also estimation of ones own skills is put into wider perspective and matures in the collaborative environment.

Q2: *Does collaboration help transfer knowledge between learners?* The code camp and its collaborative methods – self-organizing teams, collaborative IDE and the groupshuffle – indicate that a social learning environment where the learners themselves are in the driver's seat is a good fit for software development. The prior knowledge of those with more experience is transferred through collaborative activities and working environment without an unnecessary sense of teaching others.

The presence of knowledge flow is visible both in the survey data and in the interviews. The code camp acts as a community of practice – learners learn not only from the direct SE activities but also through sharing information and experiences among themselves. The camp recognizes learning both from the learner-centric and from its social and knowledge network angle.

6. FUTURE WORK

From the engineering perspective, there are numerous directions where we can take our system. These have already been partly addressed in [28], and although some of the data that has resulted from this experiment suggest improvements, we will in the following only focus on future work regarding education. In the future we plan to test the experiment setup and the collaborative learning approach in a globally distributed course environment. In comparison to the results reported here, we expect that the distribution to different countries will result in extended use of online discussions. Better yet, global distribution over several continents would create also cultural and time zone variations. At any rate, we should witness communication much to the same spirit software developers today use instant messaging, Skype, and other communication tools to discuss their activities when participating in open source projects. However, in open source the actual code-level cooperation takes place via version control systems and other means of contribution. Here, the unique element is that everything will be included in the same system, which also means that we can track down the progress in a much more detailed fashion. Naturally, gaining an extensive amount of data from real learning events would enable using big data methods to educational context, something we consider is an important new field that will gain increasing interests as MOOCs (massive open online courses) become more commonplace. Moreover, this would also enable comparison of learning results from different settings.

As already mentioned above, integrating the collaborative editing environment with an e-learning tool javala.cs.tut.fi where students solve small exercises instead of collaborating to create a larger project is one direction for future work. This can be experimented in two different ways, with peer-assistance where other students can help in completing the exercises, or with a special educator role who can monitor as students make progress. Moreover, as the original system includes numerous gamification elements, transferring them to collaborative setting is an interesting direction for future research.

7. CONCLUSION

Software engineering has increasingly become collaborative work. However, educational issues associated with collaboration in software development are complicated, as recording and analysing how collaboration took place to gain certain learning outcomes is problematic.

In this paper, we are presenting results from a code camp lasting two weeks during which the learners have been using a collaborative development environment accessible as a web service. In addition to plain collaboration in the form of joint authoring of program code, the participants also had an opportunity to use means embedded in the IDE for interacting with each other. Additionally, they had the opportunity to share experiences and ideas among the groups while still working on the application. This supports individual learning through cooperation in a joint effort and enforces knowledge transfer among the divergent learner group. Based on the experiences so far it is clear the collaboration aids learning and supports learners in gaining experience and skills useful in software development today.

8. ACKNOWLEDGMENTS

The code camp was a BEST summer learning event and part of the projects TIVIT Digital Services and ITEA2 EASI-CLOUDS, supported by Vaadin inc. The authors wish to express sincere thanks to all the participants of the experiment. At the same time, we wish to apologize for the few remaining (but still annoying) bugs in the system.

9. REFERENCES

[1] A. Bandura and D. C. McClelland. *Social learning theory*. Prentice-Hall, Englewood Cliffs, NJ, 1977.

[2] V. R. Basili, R. W. Selby, and D. H. Hutchens. Experimentation in software engineering. *Software Engineering, IEEE Transactions on*, SE-12(7):733–743, 1986.

[3] K. Beck and C. Andres. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, Reading, Massachusetts, 2nd edition, 2004.

[4] A. Begel, R. DeLine, and T. Zimmermann. Social media for software engineering. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*, FoSER '10, pages 33–38, New York, NY, USA, 2010. ACM.

[5] L. J. Burnell, J. W. Priest, and J. Durrett. Teaching distributed multidisciplinary software development. *Software, IEEE*, 19(5):86–93, 2002.

[6] K. Crowston, Q. Li, K. Wei, U. Y. Eseryal, and J. Howison. Self-organization of teams for free/libre open source software development. *Information and Software Technology*, 49(6):564–575, June 2007.

[7] B. Decker, E. Ras, J. Rech, P. Jaubert, and M. Rieth. Wiki-based stakeholder participation in requirements engineering. *Software, IEEE*, 24(2):28–35, 2007.

[8] J. DeFranco-Tommarello and F. P. Deek. Collaborative software development: a discussion of problem solving models and groupware technologies. In *System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on*, pages 568–577, 2002.

[9] Y. Engeström. *Learning by Expanding: An Activity - Theoretical Approach to D evelopmental Research*. Orienta-Konsultit, Helsinki, 1987.

[10] J. Favela and F. Peña-Mora. An experience in

collaborative software engineering education. *Software, IEEE*, 18(2):47–53, 2001.

[11] R. T. Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, 2000.

[12] N. Fraser. Differential synchronization. In *Proceedings of the 9th ACM symposium on Document engineering*, pages 13–20. ACM, 2009.

[13] M. German. Experiences teaching a graduate course in open source software engineering. In *Proceedings of the first International Conference on Open Source Systems*, pages 326–328, 2005.

[14] F. D. Giraldo, C. A. Collazos, S. F. Ochoa, S. Zapata, and G. T. de Clunie. Teaching software engineering from a collaborative perspective: Some latin-american experiences. In *Database and Expert Systems Applications (DEXA), 2010 Workshop on*, pages 97–101, 2010.

[15] E. V. Glasersfeld. *Constructivism in education.* Pergamon Press, Oxford, England, 1989.

[16] M. Goldman, G. Little, and R. C. Miller. Collabode: collaborative coding in the browser. In *Proceedings of the 4th international workshop on Cooperative and human aspects of software engineering*, pages 65–68. ACM, 2011.

[17] M. Grönroos. *Book of Vaadin.* Uniprint, 2011.

[18] I. Hadar, S. Sherman, and O. Hazzan. Learning human aspects of collaborative software development. *Journal of Information Systems Education*, 19(3):311–319, 2008.

[19] S. Hase and C. Kenyon. From andragogy to heutagogy. *Ultibase Articles*, 5(3):1–10, 2000.

[20] J. D. Herbsleb. Global software engineering: The future of socio-technical coordination. In *2007 Future of Software Engineering*, pages 188–198. IEEE Computer Society, 2007.

[21] T. Kilamo, I. Hammouda, and M. A. Chatti. Teaching collaborative software development: a case study. In *Proceedings of the 2012 International Conference on Software Engineering*, pages 1165–1174, Piscataway, NJ, USA, 2012. IEEE Press.

[22] B. A. Kitchenham and S. L. Pfleeger. Personal opinion surveys. In *Guide to Advanced Empirical Software Engineering*, pages 63–92. Springer, 2008.

[23] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. E. Emam, and J. Rosenberg. Preliminary guidelines for empirical research in software engineering. *Software Engineering, IEEE Transactions on*, 28(8):721–734, 2002.

[24] F. Lanubile, C. Ebert, R. Prikladnicki, and A. Vizcaino. Collaboration tools for global software engineering. *Software, IEEE*, 27(2):52–55, 2010.

[25] J. Lautamäki, A. Nieminen, J. Koskinen, T. Aho, T. Mikkonen, and M. Englund. Cored: browser-based collaborative real-time editor for java web applications. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, pages 1307–1316. ACM, 2012.

[26] J. Lave and E. Wenger. *Situated Learning: Legitimate Peripheral Participation.* Cambridge University Press, 1991.

[27] T. Lehtonen. Javala–addictive e-learning of the java programming language. In *Proceedings of Kolin Kolistelut/Koli Calling–Fifth Annual Baltic Conference on Computer Science Education.*, pages 41–48, Joensuu, Finland, 2005.

[28] T. Mikkonen and A. Nieminen. Elements for a cloud-based development environment: online collaboration, revision control, and continuous integration. In *Proceedings of the WICSA/ECSA 2012 Companion Volume*, pages 14–20, New York, NY, USA, 2012. ACM.

[29] S. Minocha and P. G. Thomas. Collaborative learning in a wiki environment: experiences from a software engineering course. *New Review of Hypermedia and Multimedia*, 13(2):187–209, December 2007.

[30] A. Nieminen, J. Lautamäki, T. Kilamo, J. Palviainen, J. Koskinen, and T. Mikkonen. Collaborative coding environment on the web: A user study. *Developing Cloud Software Algorithms, Applications, and Tools*, (60):275–300, October 2013.

[31] S. S. C. Oetzbek, K. Beecher, and J. Schenk. Saros: an eclipse plug-in for distributed party programming. In *Proceedings of the 2010 ICSE Workshop on Cooperative and Human Aspects of Software Engineering*, pages 48–55, New York, NY, USA, 2010. ACM.

[32] T. O'Reilly. What is web 2.0: Design patterns and business models for the next generation of software. *Communications & Strategies*, 1(17), 2005.

[33] J. Piaget. *The Child's Conception of the World.* Rowman and Allenheld, New York, 1960.

[34] C. Y. Shim, M. Choi, and J. Y. Kim. Promoting collaborative learning in software engineering by adapting the pbl strategy. *World Academy of Science, Engineering and Technology*, 53:1157–1160, 2009.

[35] G. Siemens. Connectivism: A learning theory for the digital age. *International Journal of Instructional Technology and Distance Learning*, 2(1):3–10, 2005.

[36] I. G. Stamelos. Teaching software engineering with free/libre open source projects. *Multi-Disciplinary Advancement in Open Source Software and Processes*, pages 67–84, 2011.

[37] M.-A. Storey, C. Treude, A. van Deursen, and L.-T. Cheng. The impact of social media on software engineering practices and tools. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*, FoSER '10, pages 359–364, New York, NY, USA, 2010. ACM.

[38] C. Sun, X. Jia, Y. Zhang, Y. Yang, and D. Chen. Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 5(1):63–108, 1998.

[39] L. L. S. Vygotsky. *Mind in Society: The Development of Higher Psychological Proce sses.* Harvard University Press, 1978.

[40] E. Wenger. Communities of practice and social learning systems. *Organization*, 7(2):225–246, 2000.

[41] J. Whitehead. Collaboration in software engineering: A roadmap. In *2007 Future of Software Engineering*, pages 214–225, Washington, DC, USA, 2007. IEEE Computer Society.