

9 Collaborative Coding Environment on the Web: A User Study

Antti Nieminen, Janne Lautamäki, Terhi Kilamo,
Jarmo Palviainen, Johannes Koskinen, and Tommi Mikkonen
Department of Pervasive Computing
Tampere University of Technology
Email: \{antti.h.nieminen, janne.lautamaki, terhi.kilamo,
jarmo.palviainen, johannes.koskinen, tommi.mikkonen\}@tut.fi

Abstract—Today, techniques made popular by Web 2.0 enable massive co-
operation of online users. In the spirit of Google Docs, where multiple editors
can cooperate in real time to craft a single document, we believe that it is
only a matter of time before software development takes the step towards
real-time collaborative online editing and development instead of artificially
forced interleaving implemented in version control. First, to study the dif-
ferent aspects of this approach, we have implemented a web-based collabora-
tive programming environment that extends the capabilities of a code editor
with numerous features commonplace in social media. Second, to gain the
developers perspective to real-time collaborative development, we arranged a
week-long experiment where participants composed a web application using
our environment and report the results here. Towards the end of the chapter,
we discuss the lessons we have learned about real-time collaborative software
development. Furthermore, we list the features that developers find essential
in such a system as well as compare our results with other reported research.

Keywords—Software development, online collaboration, user study.

9.1 Introduction

Today, software is being developed by teams of programmers who may be
globally distributed, work in dissimilar environments and come from various
backgrounds. Contemporary development approaches such as pair programming along with other extreme programming practices [2], global software engineering [13] and collaborative software development [14] illustrate this trend. However, while the main designs and principal guidelines are often crafted in a collaborative fashion in meetings of different kinds, the final design of software artifacts that will later be compiled and run are usually designed and written by individual developers, working in relative isolation from one another and with limited face-to-face or any types of contact. With such an approach, it is not surprising that communication among the developers has been considered a problem for over two decades [23].

Recently, various fields of industry as well as consumer applications are experiencing a paradigm shift towards web-based systems. Web 2.0 allows online collaboration beyond any previous expectations, making the web the first truly global platform for cooperation and collaboration. Fundamentally, Web 2.0 technologies combine two important characteristics, collaboration and interaction. The former refers to the "social" aspects that allow a vast number of people to collaborate and share the same data, applications, and services over the Web. The second, equally important aspect of such technologies is that they enable building web services that behave much like desktop applications with the added benefit that the user does not have to install any software on their computer; any device with a web browser can be used to access the applications. We believe that these two properties are paving the way towards shifting software development to the web. Examples of such systems include Cloud9 IDE\(^1\), Eclipse Orion\(^2\), GitHub\(^3\) and Codeanywhere\(^4\). From these premises, we have implemented a web-based real-time collaborative software development environment, CoRED, that has been introduced from the technical perspective in [17, 19].

In this chapter, we evaluate collaborative software development with a number of developers who jointly compose a web application using CoRED. With CoRED, several developers can access and modify the same code base in parallel, with no need to use separate version control, following the spirit of real-time collaborative software such as Google Docs. The contribution of this chapter is the introduction of the developers’ view to the collaborative browser based integrated development environments (IDE), including both the experiment setup as well as the most important lessons we have learned during the experiment.

The rest of this chapter is structured as follows. In Section 9.2, we in-

---

\(^1\)https://c9.io/
\(^2\)http://www.eclipse.org/orion/
\(^3\)https://github.com
\(^4\)https://codeanywhere.net/
roduce our collaborative web-based development environment. In Section 9.3, we define the research approach and the experiment setup. Section 9.4 presents the data collected and introduces the main results we have learned from the experiment. Section 9.5 further discusses the research questions with discussion and lessons learned. Future work is presented in Section 9.6. Section 9.7 covers work related to our research, and finally, Section 9.8 concludes the chapter.

9.2 Collaborative Development: The CoRED Approach

Moving a software development environment to the web has at least two major benefits. First, developers get rid of the process of installing, configuring and updating the environment for developing, testing, running, and debugging the applications; everything is readily available on the web. Second, the web can enhance collaboration and communication. Most developers are already familiar with using various co-operation tools on the web. Integrating such tools into the development environment can make the collaboration more tightly combined with the process of writing code.

Previously, we have introduced our proof-of-concept system CoRED [17, 19]. In addition to editing the code in real-time collaboration, developers are able to communicate with each other using means familiar from social media. The architecture of our system is typical for a web application. While the client side running inside the browser enables interactions with the system, most of the business logic of the system runs on the server back-end. Our system is implemented with the Vaadin [10] framework. Vaadin provides user interface components, implemented as Google Web Toolkit [22] widgets, that are integrated with the server-side Java code. With this approach, Vaadin applications can be written entirely in Java. The Vaadin framework takes care of connecting the server-side Java code to the user interface components loaded into the browser.

The architecture of CoRED is illustrated in Figure 9.1. The client side contains an open source Ace code editor. Ace is implemented in JavaScript, wrapped in our system inside a GWT component which in turn communicates with the server using HTTP communication channels provided by Vaadin. Other IDE components are implemented in a similar way, some of which are provided by the standard Vaadin package while others are developed by us or other 3rd party developers. Furthermore, the server-side code utilizes Java

\footnote{http://www.java.com}

\footnote{http://ace.ajax.org}
Figure 9.1: The architecture of CoRED

Development Kit for parsing and compiling Java code, and other 3rd party libraries for various tasks. For a more detailed description of the system, the reader is referred to [17, 19].

Figure 9.2 shows a screenshot of a CoRED project that is currently edited by two developers. On the left sidebar there is a list of files in the project (marker with number 1), a deploy button (2), and an avatar of each collaborator currently viewing the project (3). In the bottom left corner, there is a chat box for communication (4). Most of the screen space is reserved for the source file under edit. On the right, the users that view the file are shown (5). Additionally, the cursor positions as well a possible text selection of other users is shown within the opened file. The more rarely used features are accessible via the menu bar on the top of the screen. A video clip demonstrating the main capabilities is available at http://cored.cs.tut.fi.

CoRED offers various features to support software development, most of which are familiar from traditional desktop IDEs. In addition, some features geared towards collaboration exist. Next, we briefly list the most relevant facilities.
Project support. In CoRED, projects are used to constitute compilable and runnable systems, similarly to many IDEs. Upon entering the CoRED web page, the user can select to join any of the projects that are currently under development.

Error checking, suggestions and code completions. Another feature seen also in many installable IDEs is the ability of CoRED to pinpoint Java errors in the code. In addition, the editor suggests possible code completions. The suggestions can be invoked in two ways, by using a special key combination or by typing a dot after, e.g., an interface or an object name.

Single click deployment. The developer can compile and deploy the project with a single click. The deployed system will be available online as a service, and a related URL (uniform resource locator) is provided to the user to test the system.

User identity. In order to have meaningful identities for users, it is possible to log in using a Facebook\(^7\) account or a Gravatar\(^8\) avatar. In addition, a guest login with a simple nickname is available.

Collaborative editing. To enable users to edit the code in real-time col-

---

\(^7\) [http://facebook.com](http://facebook.com)

\(^8\) [http://gravatar.com](http://gravatar.com)
laboration, we have used the Differential Synchronization algorithm by Neil Fraser[7]. It is a robust collaborative editing algorithm with open source implementations available on several languages, including Java and JavaScript.

*Pop up notes and chat.* Besides editing the text itself, the users have the option to augment the code with pop up notes in a fashion normally associated with text rather than code. The goal of these facilities is to allow the separation of comments that are related to the actual code, and discussions that developers have over a particular solution. Users can place notes into the code (number 6 in Figure 9.1), and view the related discussion (number 7 in Figure 9.1) by moving the cursor on top of the note. In addition to the inline notes, CoRED also offers a project-wide chat.

*Code locking.* In CoRED, a single editor can lock a portion of the code for exclusive editing. For example, if a user wanted to make changes to a specific function without anybody interfering, he/she could lock a function by selecting the function and clicking a “Lock for me” button. After locking, the locked area cannot be modified by other users until it is released by the developer who originally locked the area.

### 9.3 Experiment

We set out to study how developers approached CoRED and real-time collaborative software development through an empirical experiment [1, 15]. In addition to the experiment that ran as a week long code camp, personal opinion surveys [16] on the participants’ background and experiences were conducted. During the experiment we also conducted interviews and recorded development data into log files. Also, to get preliminary feedback on CoRED, two short proof-of-concept sessions were held before the actual code camp as precaution for latent problems in the setup.

#### 9.3.1 Research Questions

The main motivation for the experiment was to get a developer’s view to real-time collaborative coding and to evaluate how well CoRED would work when developing a web application in small developer teams. Additionally, there was a need to identify the features developers find necessary in real-time collaborative development environment.

The research questions of the study were:

**Q1** How do developer teams use a collaborative IDE such as CoRED?

**Q2** What features do developers expect from a real-time collaborative IDE?
To find answers to Q1, an experiment where teams of computer science students used CoRED to develop web applications was set up. Section 9.3.3 discusses the experiment setup in more detail. Additional data was gathered through personal opinion surveys (pre- and post-camp surveys and individual interviews) discussed further in Section 3.4. Section 9.4 answers to Q1 by presenting data from surveys, log files and interviews. Combined with the participants exposure to real-time collaborative programming, the interviews and surveys also provided answers for Q2, presented in Section 9.5.

9.3.2 Proof-of-concept Session

In order to manage risks, two short proof-of-concept sessions were conducted before the actual experiment. The purpose of the sessions was to validate the ability of our installation to support the planned number of developers and to find possible fatal bugs or deficiencies in CoRED in order to get any such issues fixed before the code camp. The proof-of-concept session was held twice, on consecutive days, as a part of a Web Services course at Tampere University of Technology. The participants, 11 on each session, were undergraduate software engineering students, allowed to split freely into groups of 1-3 people. The students familiarized themselves beforehand with the very basics of the underlying application platform Vaadin. The session consisted of a brief introduction, an hour of programming, and a small survey. The assigned task was to insert deleting functionality to an existing REST-based image search client.

After the session, many participants were happy about the easy access to IDE without installation, simplicity of deployment of applications, and the work being immediately available for other group members to see. Some considered CoRED to be a good tool for pair programming while some thought that it is probably only suitable for developing small applications.

No critical flaws that would prevent the conduction of the experiment came up during the proof-of-concept session. Anyhow, the participants reported deficiencies such as the lack of multiple editor views open at the same time, no indication that another person is editing the same file, and Java error checking not working in all cases. Fixing all these problems required no changes for the architecture and therefore we were able to fix them before the code camp, although the last one only partly: the user still may need to manually press a "compile all" button in some rare cases. Other comments about CoRED addressed the general immaturity of the tool, lack of IDE features such as "go to method definition" option and lack of proper debug support. These problems were not addressed for the code camp version of CoRED. For the sake of answering the research questions and not wanting
<table>
<thead>
<tr>
<th>Day</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Camp starts. Lectures on the tools and the REST architecture. The teams form and start designing their project.</td>
</tr>
<tr>
<td>2</td>
<td>Development starts. Teams use CoRED and are encouraged to work in the same classroom.</td>
</tr>
<tr>
<td>3</td>
<td>Development continues. As a short experiment, each team chooses a suitable, socially adept person to act as the project leader. The rest of the team is randomly assigned onto a different project for two hours. The project leaders are located into a separate room from which they communicate with their newly formed team by using CoRED.</td>
</tr>
<tr>
<td>4</td>
<td>Distributed development. The developers can choose their working environment freely and are encouraged to leave the classroom.</td>
</tr>
<tr>
<td>5</td>
<td>The teams finalize their projects. Presentations of the finished applications are given.</td>
</tr>
</tbody>
</table>

Table 9.1: Daily activities during the code camp

to let our own assumptions on developer needs to influence the study too much, we found it best to leave these points open for the code camp and let the experiment to bring light to where CoRED needs to be matured.

### 9.3.3 Experiment Setup

In order to extend the amount of data to analyze, we organized a one-week (December 17th to 21st 2012) code camp for implementing a web application using CoRED. The task for students was to design and implement a Vaadin application that utilizes one or more external RESTful [6] APIs of their choice. Table 9.1 presents the flow of the entire camp and the daily alterations that were made to keep the participants’ motivation up and to make sure they put the collaborative features of CoRED to full use.

A total of 23 students participated on the first day of the code camp. Out of these, 19 students completed their group project and wrote the required assignment for passing the course. The size of the groups varied from two to four. As the code camp ran during an exam week and right before the Holidays four students dropped out during the first days of the camp. We were able enquire a couple of them for reasons for dropping out, and they mentioned lack of time and personal issues as the main reasons for not finishing the camp.

All the groups were able to produce a working web application, although some of the groups had to leave out certain features they had originally planned. Among the projects produced during the code camp was a system that generates funny meme pictures based on keywords or phrases, utilizing
<table>
<thead>
<tr>
<th>Day</th>
<th>Survey or interview</th>
</tr>
</thead>
</table>
| 1   | Survey for investiga-
     | ting background kno-
     | wledge, preconcep-
     | tions and expecta-
     | tions |
| 4-5 | Six volunteers were
     | picked and inter-
     | viewed about their
     | experiences in collab-
     | orative coding and CoRED |
| 5   | A survey for examined
     | the gained knowledge, ex-
     | periences from CoRED and
     | the different collaborative
     | programming experiences |

Table 9.2: Surveys and interviews made during the code camp

APIs of Twitter⁹ and Meme Generator¹⁰. Some of the other projects generated personal pony themed home pages, allowed users to share and browse old exam questions, or view venue information and other data provided by Foursquare¹¹.

### 9.3.4 Surveys

During the camp, the participants filled several surveys listed in Table 9.2. Furthermore, the participants got a delegated anonymization code that they used throughout the camp when filling surveys or taking part in an interview. The codes allowed us to track the evolution of the participants’ knowledge.

The objective of the before and end of the camp surveys was to monitor if the participants felt that they have learned something or gained new knowledge during the code camp, as well as to get feedback from the system and collaborative way of working. Interviews were used to get more informal feedback and for probing issues the participants wanted to emphasize.

### 9.4 Results

Next, we present the data gathered during our experiment. The data comes from three main sources: surveys, data logs, and interviews. The surveys provide basic data on the participants as well as their views on CoRED and the whole code camp. The data logs are used to find out how the participants actually worked during the camp. Deeper understanding of the participants views on real-time collaborative development was gathered during interviews.

---

⁹ [www.twitter.com](http://www.twitter.com)
¹⁰ [http://memegenerator.net](http://memegenerator.net)
¹¹ [www.foursquare.com](http://www.foursquare.com)
<table>
<thead>
<tr>
<th>Skill</th>
<th>Initial</th>
<th>Std. dev. of Initial</th>
<th>&quot;Learning&quot;</th>
<th>Std. dev. of &quot;learning&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming</td>
<td>4.4</td>
<td>0.6</td>
<td>-0.1</td>
<td>0.4</td>
</tr>
<tr>
<td>Java</td>
<td>3.1</td>
<td>0.8</td>
<td>0.4</td>
<td>0.6</td>
</tr>
<tr>
<td>Vaadin</td>
<td>1.6</td>
<td>0.7</td>
<td>1.5</td>
<td>1.1</td>
</tr>
<tr>
<td>REST</td>
<td>2.1</td>
<td>1.1</td>
<td>0.9</td>
<td>0.9</td>
</tr>
</tbody>
</table>

Table 9.3: How much the participants felt they learned during the code camp

### 9.4.1 Survey

Out of the participants, 21 - 16 master's degree and 5 postgraduate students – answered the pre-camp survey. Both the pre-camp and post-camp survey was answered by 14 of them. Table 9.3 presents the quantitative data collected from the questionnaires. In both questionnaires we asked the participants to give themselves a value from 1-5 to describe their knowledge of programming, Java, Vaadin and REST, respectively. The number one stands for no experience and five means that the developer is very experienced. Table 9.3 shows the average initial skill levels of the participants as well as the learning outcome during the course. As expected, the skills with the lowest start values improved the most. The overall result indicates that the participants feel they have learned during the code camp. During group work, people were able to compare their own and other students skills and we speculate that to be an explanatory factor behind the small drop in basic programming skill during the course. Based on observations made on raw data, it seems that the course was most rewarding for the people with basic knowledge of Java and no skills in Vaadin or REST.

In freeform feedback, the following themes repeated several times:

*Deployment.* "Deployment is hard while people editing." To be able to deploy and test an application, the code had to be compilable. In a group with several people, this requires some coordination and was therefore considered difficult.

*Testing tools.* "No test tools. Testing was sometimes hard because many people were editing the code at the same time", “No access to server logs” and “Desktop IDE is much better. Real time debugging and much more (shortcuts, smooth, dynamic); only downside maybe not seeing others code”.

*Version control.* "No version control or timeline." Developers often have a need to be able to restore or study older versions of code. However, in CoRED this was not supported except by exporting project versions as zip files.
### 9.4.2 Log Data

CoRED recorded data on each group's work to a log file. By analyzing the logs, we could get some insight on how the groups collaborated during the code camp. The most important item type recorded was *edits* by the group members. An edit is a small change to the project: addition, removal or a combination of both. The system sent an edit to be synchronized with the shared document after the user had paused typing for 500 milliseconds, in which point it was also recorded to the log. Thus, a typical edit is a line of code or less.

As confirmed by the logs, the participants developed their application mostly during "office hours" when the whole group was present in the same classroom. However, there were some instances when a group member contributed to the project in the evening on their home computer, or even in a bar at night (as one of the participants described in an interview). The chat of the development environment, which was also logged, was not used that much, mostly because all the group members were sitting next to each other most of the time. The only time when the chat was in heavy use was during the two-hour experiment when project leaders were physically separated from their teams.

Some group statistics as well as collaboration metrics are shown in Table 9.4. The first two columns are the group id and the number of group members. The next column is the average skill level of the group as reported by the group members themselves in the initial questionnaire. The projects were evaluated independently by two members of the course staff based on the originality of the idea and the quality of the implementation. The *score* column shows the group score, smaller number being better. The table is sorted by the score, the best group being at the top. The next column shows the total project size, measured as lines of code (LOC).

<table>
<thead>
<tr>
<th>Group Id</th>
<th># members</th>
<th>Avg. level</th>
<th>Score (smaller is better)</th>
<th>LOC</th>
<th>Collab. files</th>
<th>Collab. edits (%)</th>
<th>Most active developer (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
<td>3.3</td>
<td>3</td>
<td>691</td>
<td>3/11</td>
<td>11</td>
<td>67</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
<td>2.7</td>
<td>5</td>
<td>747</td>
<td>5/13</td>
<td>8</td>
<td>63</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
<td>2.9</td>
<td>5</td>
<td>1482</td>
<td>3/11</td>
<td>7</td>
<td>31</td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td>3.0</td>
<td>7</td>
<td>800</td>
<td>1/2</td>
<td>13</td>
<td>67</td>
</tr>
<tr>
<td>E</td>
<td>3</td>
<td>2.0</td>
<td>10</td>
<td>240</td>
<td>0/1</td>
<td>1</td>
<td>88</td>
</tr>
<tr>
<td>F</td>
<td>2</td>
<td>2.4</td>
<td>12</td>
<td>874</td>
<td>3/7</td>
<td>9</td>
<td>93</td>
</tr>
</tbody>
</table>

Table 9.4: Some metrics on the groups and collaboration
The remaining columns of Table 9.4 are intended for describing the collaboration aspects of the groups. We considered a file to be collaboratively edited if no one developer is responsible for over 80% of the file's content. The Collab. files column shows the number of collaboratively edited files compared to the total number of files in the project. If another team member has edited the same file less than 30 seconds before the latter, the edit is considered collaborative. The Collab. edits column describes the number of edits that were collaborative. The Most active developer (%) column describes the percentage of the characters of the total project produced by the most active member of the group.

As can be seen in the Table 9.4, in all the groups except one, there was one member who wrote over 50% of the total code. As an exception, the work in the group C was notably evenly distributed: each of the four members contributed approximately a quarter of the content. In two of the groups with the worst scores, a single person wrote 90% of the code. This could be an indication of either poor collaboration within the group or lack of relevant skills of other project members. The latter conclusion is somewhat supported by the lower self-reported skill levels by the two groups compared to other groups. In both of these groups, the person with most Java experience did most of the coding.

The collaboration level, expressed in Table 9.4 by the Collab. edits metric does not seem to have a large correlation to the group performance. In most of the groups, 7-13% of the edits were "collaborative" as defined earlier. The only exception is group E, where there were almost no collaborative edits at all. The group did not produce that much code in terms of LOC, and the majority of the work was done by a single member. In other, somewhat larger projects, even though the group presumably try to structure their work in such a way as to avoid working on the same parts of the project, around 10% of the edits are collaborative. Some of them may be purposeful collaboration between project members while others are accidental. It is not easy to differentiate between the two cases based on the logs.

The column Avg. level in Table 9.4 shows the number that is the average of all the skills (Programming, Java, Vaadin, REST) of all the project members. We could make a hypothesis that instead of average, the maximum value for each skill level would better predict the performance of a closely working group such as the ones in our experiment. That is, a group with one member with a lot of experience on Vaadin, another with a lot of REST experience, etc. would outperform a group with each member having average skill in each of the categories. In this study, though, we did not find evidence in either direction; the averages of the maximum skill levels (not shown in the table) were quite close to the average level.
To get an overview of the typical collaborative work during the code camp, Figure 9.3 shows a four-hour long period of one of the projects. 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. That is, the edits between two lines are in the same file. There is nothing very surprising in how the groups worked on the projects. Most of the time a single person edits a part of the project alone with occasional closer collaboration here and there in some parts of the project. In the logs it was quite rare to see two or more people working on the same piece of the project for very long at a time.

At the end of the log analysis, it should be noted that the logs we analyzed may not contain all the coding work for all the members. We analyzed the logs of only the main project of each group. In addition to the main project, some groups may have used other test projects to try out new features that may be later included into the main project. The possible additional projects are not considered in this log analysis. However, in each of the groups, most of the work seemed to happen in the main project and in any case all the features had to eventually be integrated into it.

9.4.3 Interviews

During the code camp, a total of six semi-structured interviews were held. Interviews were conducted between Wednesday afternoon and Friday midday, taking a half an hour each. In principle, all the interviews followed the same structured pattern with 11 themes. However, the aim of the interviews was to get some free form feedback and therefore we did not try to get answers to all the questions if there was a good flow of conversation. Interviews mostly strengthen the findings of the survey, but also some new themes appeared during conversations. During the interviews it was also possible to get more detailed feedback.

Collaborative editing. Students liked to try out something new and mostly positive feedback was given concerning real-time collaborating. It was reported that with the collaborative IDE working felt more like group work compared to a traditional environment with asynchronous version controlling such as SVN or Git. It was reported that feeling of group work came from the ability to see in real-time what others are doing. Furthermore, they also considered it as a motivational aspect to know that others are able to see what one is actually implementing.

Concerning team size in real-time collaboration, the interviewees reported that a group of four people is too large for working on an ad-hoc basis. Therefore some structure for working was needed. In one group of four
The he is arbitrary.

This best done. The flows are the boundaries i.e. values between the same lines are on the same field. The order of the developer, announced as Ache, Bob, and Charlie. The vertical axis represents the character location where the event occurred. A four-hour piece of a log of one of the projects, borders are marked with different symbols for each.

Figure 9.3: A four-hour piece of a log of one of the projects. Borders are marked with different symbols for each.
people, the problem was solved by splitting into two minigroups with tasks of their own. One example was that the less experienced half implemented user interface while the other half was doing business logic. Furthermore, in the minigroups the pairs often adopted two separate roles: a coder and a "googler" who was searching for solutions to problems.

Students seemed to quickly realize that one big strength of a browser based IDE is the ability to use any device in a location independent way. While most of the work was done during the office hours, the students liked that they had an option to continue work when and where ever they felt like.

Deployment. Deployment was generally considered the most severe problem in CoRED. Most of the interviewees mentioned that a lot of time was wasted while waiting other group members to finalize their tasks. In some groups this lead to situations that lots of code was commented away from compilation. As no connectivity to version controlling existed, the comment blocks were also used for storing obsolete code and later it was difficult to know that which parts of the code should be restored from the comment blocks and which can be removed. Some solutions for solving the problem were suggested, such as storing an error-free versions of each file and using them instead. However, the deployment problem also had some positive outcomes. It was reported that because of the problem, the developers mostly concentrated on one problem at a time and tried to finalize their code as quickly as possible.

Project management tools. To make work more effective, some project management tools were suggested to be integrated as a part of the system. For example, it was suggested that a project backlog for future tasks would be a good addition to CoRED. In addition, in the current implementation of CoRED, all the users logged into the system are able to read and write all the projects and files. During the camp, the problem was solved by allocating a private virtual machine to each of the groups and no problems concerning access policies were reported. However, it was suggested that each file and project should have an owner that would be able to share read and write permissions. Some refactoring tools were also asked to be added to CoRED.

Version control. A graphical tool for browsing old versions of a project was considered to be mandatory. One of the interviewees stated that currently most of the old code can not be deleted, but is moved inside a comment block and therefore it is later difficult to know which of the code blocks are still needed. Interviewees stated that certain problems concerning undo and redo functionalities exist in CoRED. A version control system would also ease these problems, as code from older states could be easily restored using version control if undo fails.
As miscellaneous feedback, it was suggested that two weeks would be a more suitable length for a code camp instead of one week. Furthermore, the possibility to include resource files like images and css style sheets to the project was requested. Some of the interviewees exposed that a collaborative IDE can also cause feelings of confusion: it is sometimes impossible to understand the state the project and what are the responsibilities of each group member as the target is constantly changing. In addition, it was stated that collaboration would be easier in a group with no high skill level differences.

9.5 Developer Expectations for Real-time Collaborative IDEs

Based on the surveys, interviews and observing the teams working during the code camp, valuable information concerning real-time collaboration was collected. Above we described how the teams approached real-time collaborative programming, thus answering to the research question Q1. In this section, we answer the second question Q2 by presenting the features important for a real-time collaborative IDE.

The second research question was:

Q2 What features do developers expect from a real-time collaborative IDE?

Being able to test a web-based real-time collaborative integrated development environment, even with certain deficiencies, gives more perspective for analyzing and reasoning on what features developers expect from such a system. Based on the experience from the code camp we have collected a list (see Table 9.5) of the main features that should be included in a real-time collaborative IDE. In addition to those findings, we speculate that by fixing all the features and organizing a new code camp it is probably possible to find new demands for the system.

The real-time collaboration aspect brings additional twist to deploying and running the application under development. In the basic case, if files contain errors or unfinished code blocks, the compilation fails and the application can not be deployed or tested. During our experiment, it turned out that in collaborative work, most of the time projects contain unfinished features and therefore deploying the application requires some coordination. This issue did not come up in the two-hour proof-of-concept sessions as the task at hand was much smaller, and the team members could thus more easily coordinate their actions, but was a major hindrance in larger projects. An important realization confirmed in our experiment was that even though
<table>
<thead>
<tr>
<th>Feature</th>
<th>Implemented in CoRED</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edits should be visible in real-time</td>
<td>yes</td>
</tr>
<tr>
<td>Possibility to deploy a project containing errors</td>
<td>no</td>
</tr>
<tr>
<td>Project management tools</td>
<td>no</td>
</tr>
<tr>
<td>Single click deployment</td>
<td>yes</td>
</tr>
<tr>
<td>Sufficient awareness between the users</td>
<td>partly</td>
</tr>
<tr>
<td>Support for undo/redo</td>
<td>partly</td>
</tr>
<tr>
<td>Tools for communicating</td>
<td>partly</td>
</tr>
<tr>
<td>Tools for debugging and testing</td>
<td>no</td>
</tr>
<tr>
<td>Version control support</td>
<td>no</td>
</tr>
</tbody>
</table>

Table 9.5: Important features for a collaborative web based IDE, in alphabetical order

the developers edit the same code concurrently, the running and debugging of the application should be separate for each developer.

Many developers are used to traditional desktop based IDEs such as Eclipse\textsuperscript{12} and Visual Studio\textsuperscript{13} and have got used to debug tools provided by these environments. Thus, they have rather high standards for what a web-based environment should offer. Many of the participants considered features such as debugging tools and an ability to set breakpoints to the code to be vital.

Functional undo/redo is also a feature that users expect of a code editor. Collaborative editing makes undo and redo more difficult to understand, as it needs to be designed what is undoable and redoable. Are the users only undoing their own edits, or does the system allow to undo edits made by other users? How should the overlapping edits be undone?

Even in the realm of real-time collaboration, version control is a highly requested feature. Even though version control system is not needed to share the code, it is useful because it allows the developers to revert back to earlier revision as well as start separate development branches. There could be several ways to integrate a version control system into a real-time collaborative environment. It would be simple to just connect the development environment to one of the available version control systems and add a commit button to the user interface. However, how should the old versions be restored? In real-time collaboration, the restoring of old states would affect all the users and therefore it is probably not a good idea to allow. As an alternate option, we are considering a timeline approach that allows users to drag a slider, and have a read-only access to history. That would allow users to copy features

\(^{12}\text{http://www.eclipse.org}\)
\(^{13}\text{http://www.microsoft.com/visualstudio}\)
from the older versions to the current. However, branches, the possibility to create forks and full restorations would probably not be available in the user interface in such a case.

A different approach to version control could be to purposefully keep the responsibility of version control outside of the web-based collaborative development environment. In this approach, the developers could use any editor they want, but additionally have the possibility to start collaboration sessions on the web. At the start of a collaboration session, the code is retrieved from a version control system to be edited in real-time collaboration. At the end of the session, the code could be committed back to the version control system, and, if needed, another session would be started. Thus, the real-time collaborative environment would just simply be used in between two commits. This approach would have at least two benefits. Firstly, the developers who do not want to use real-time collaboration could use any environment they like and just collaborate via the version control system. This would also allow a development team to be split into smaller collaborating units of, for example, two people. Secondly, there is no need to implement complicated version control features into the web-based environment because most of the responsibility for branching and other version control management would be elsewhere.

Communication is essential in software development. In our experiment the role of the communication tools did not play that big a part because the teams were mostly co-located and could simply talk to each other. During the two hour experiment the sole communication channel between the project leader and their team was the chat box in the environment, which, based on the interviews, succeeded in doing its job. It may not be necessary to try to include all the communication channels to the development environment. Teams could, for example, use an external Voice over IP software for communication. In our experiment, one of the groups reported using a separate chat program for communicating, mostly because they were used to that.

In a collaborative IDE that is open for everybody to use, some project management tools are needed. In a relatively small scale use, such as our experiment, the system can be based on the trust between the users. However, in a system with a large user base, tools for sharing and restricting read and writes to projects and files are necessary.

In real-time collaborative tools, awareness is an important concept. There has been lot of research done on, as well as multiple definitions of awareness. Basically awareness means “knowing what is going on” [5]. When people work face-to-face, they exchange a lot of information with their speech, facial expressions, gestures, and other actions as well as by manipulating their shared environment. A large part of that information is lost when using
software tool for collaboration. That is why the tool must deliberatively add features to improve the awareness of other collaborators. Information such as who are currently available for collaboration, what are others looking at, and what are they doing should be transmitted via the collaboration tool.

9.6 Future Work

As already discussed, CoRED is a research artifact and does not fully meet developers’ expectations. Via our experiment, we gathered hints on which additional features would bring the most benefit for our system as well as directions for future research on real-time collaborative software development. Before moving on to the ideas for future work, we briefly evaluate the current state of CoRED based on the essential features of a collaborative development environment envisioned above.

The last column of Table 9.5 shows how CoRED currently fulfills the requirements we discussed earlier. When considering the features that are not available in CoRED, no major technical issues are foreseen in their implementation, as long as we can rely on already existing concepts and implementations. We are presently evaluating the relative importance of these from the developer perspective. So far, the ability to deploy a partially erroneous project and the support for version control have been considered as the most important ones.

Deploying and running the collaboratively developed applications turned out to be one of the most apparent hindrances in CoRED. To be able to run the application, the whole project has to be compilable, a condition often not satisfied during real-time collaboration. One way of solving this problem is given in [9], discussed in more detail in Section 9.7. Another deficiency related to running the application was the lack of debugging support. Creating a mechanism that allows the developers to print debug text that would be shown in the IDE would be useful and relatively simple to implement. Setting breakpoints in the code to stop the execution of the application at certain position and inspect the values of variables is a feature seen in many traditional IDEs. Such feature in CoRED would be possible although laborious to implement.

Integrating a version control system into real-time collaborative software development is another subject for future work. As discussed in Section 9.5 and additionally in [19], there could be various ways on how to integrate version control into a system such as CoRED. We plan to further develop models for using version control systems in a real-time collaboration, implement such support in CoRED and try them out in later experiments. As a
related issue, the undo/redo feature does not work optimally in the current version of CoRED in the face of collaborative edits. We need to rethink the undo/redo behavior, and possibly incorporate into some kind of version control or timeline approach. The role of revision control in the light of collaborative software development has also been studied by Magnusson et al. [18]. We assume that the same mechanisms will be reusable in the context of our work, where micro-scale revisions that are created in a collaborative fashion form the technical artifact.

When implementing new features to a real-time collaborative IDE, heavy emphasis needs to be put on the awareness features of the system. Supporting awareness of other project members and the state of the project would be even more important if the developers are physically separated from each other, as opposed to what was the case most of the time in our experiment. In the current version of CoRED, some steps for improving awareness have been taken. CoRED shows who have opened the project, who edits which files as well as their cursor positions. Still, some of the interviewees said that they had trouble keeping track of what is going on. Awareness could be further improved by more clearly indicating where the others are looking at and when they wrote new code. Introducing a version control system with the possibility of branching would pose even more challenges to the awareness aspects of the user interface.

Our goal is to expand the research on real-time collaborative programming in two dimensions. Firstly, we will implement some of the features discussed above and study how they affect the collaborative software development process. Secondly, we seek to broaden the scope of the collaborative programming experiments to a distributed setting by organizing an intercontinental code camp. A development team where the members do not have the benefit of physical proximity to each other would pose greater challenges to the cooperation and collaboration aspects of the tools, as well as possibly expose aspects of real-time collaborative programming not yet captured in the mostly co-located setting described in this chapter.

9.7 Related Work

Computer-based tools that support collaboration, often called groupware [4], have been created, used and studied heavily in the multidisciplinary field known as computer supported cooperative work [21, 11] for decades. Also in the domain of software development, collaborative processes and tools have been a subject of study. Gutwin et. al. [12] have studied distributive collaborative software development from the perspective of group awareness.
DeFranco-Tommarello et. al. [3] also explore collaborative software development and analyze a list of tools that allow software developers to communicate, coordinate and design software artifacts collaboratively. Even though these kinds of collaboration tools for improving the software development process have been extensively used, the act of writing code in real-time collaboration is a more rarely seen phenomenon. Even though implementations for real-time collaborative code editors have existed much earlier, they have really gained popularity only along with the web becoming a viable application platform.

A number of real-time collaborative software development environments have appeared on the web in recent years. Cloud9\textsuperscript{14} is one of the most popular of such environments. It contains a runtime environment for Node.js\textsuperscript{15} applications as well as an integration with various version control systems. Another real-time collaborative web-based IDE called DevTable\textsuperscript{16} supports creating HTML5 as well as Python applications that can be run in Google AppEngine\textsuperscript{17}. Codenvy\textsuperscript{18} have also added some real-time collaborative features in their IDE. Many other environments, such as Stypi\textsuperscript{19} and Collabedit\textsuperscript{20} offer a more limited system without possibility to run the applications in the environment.

There has also been some environments for research purposes. Collabode [9] is a collaborative web-based IDE, implemented as an Eclipse plugin. The most notable feature of Collabode is error-mediated integration. The goal of error-mediated integration is to prevent other collaborators’ erroneous code to interfere with other developers. It works by keeping a separate copy of a file for each developer, and additionally an error-free copy to which edits are only applied if that can be done to create an error-free file. Thus, even though everybody sees what the other developers write in real-time, everybody has their own version of the project that can be run whether the other developers are in the middle of writing or not. The need for this kind of technique became apparent also during our experiment. Collabode has also been used as a platform for user studies on real-time collaborative programming. Compared to our experiment, their study was smaller in scope (30 or 40 minutes, single file, Java console application) and mostly concentrated on validating their error-mediated integration algorithm.

\textsuperscript{14}https://c9.io/
\textsuperscript{15}http://nodejs.org/
\textsuperscript{16}https://devtable.com
\textsuperscript{17}https://developers.google.com/appengine/
\textsuperscript{18}https://codenvy.com
\textsuperscript{19}https://www.stypi.com
\textsuperscript{20}http://collabedit.com/
In our experiment, we did not instruct the developers to assume any roles (apart from the two-hour experiment on the third day); they were freely allowed to find a suitable way of working in a collaborative environment. A different approach, envisioned by Goldman [8], is that the real-time collaborative environment could offer for some novel development models to more effectively structure work and to advance close collaboration. They suggest three such models: test-driven pair programming where one person writes the code while another developer writes the tests, micro-outsourcing where the main developer could “outsource” small implementation tasks to other developers, allowing himself to remain on the same level of abstraction, and mobile instructor where one person takes the role of a teacher. As far as we know, a proper support for these kind of roles is not yet implemented in any IDE; they could offer additional interesting research directions.

Another development model often associated with real-time collaborative environments is distributed pair programming. That is, pair programming where the participants are not required to be co-located. Saros [25] is a real-time collaborative tool mostly intended for distributed pair programming, with various features for improving awareness. Unlike CoRED, Saros is not web-based but is run inside a desktop Eclipse client. As reported in the case of Saros, companies and open-source projects were reluctant to try out their tool for distributed pair programming [24]. That could be taken as an indication that, more generally, it may not easy to get the software developers to adopt new development models offered by real-time collaborative tools.

Even if developers are not ready to use real-time collaboration in ordinary work, it could be utilized in some specific situations where close collaboration is essential. One such situation is merge conflicts occurring in a version control system when combining two incompatible branches of development. To resolve the conflict, all the parties involved in causing the conflict could participate in a real-time collaborative conflict-resolving session. A more detailed explanation of such process as well as a web-based tool enabling it is given in [20].

9.8 Conclusion

The current paradigm shift from desktop applications to the browser based applications makes it possible that software development can shift from desktop based IDEs to the web. Among many other good features, this shift makes it easier to enable collaboration between the developers as the developers are connected to the same system.

Earlier, we have implemented a real-time collaborative code editor named
CoRED. To study the software developers perspective of CoRED and collaborative coding in general, we ran a week-long experiment. During the experiment, six small groups of students used the tool for one week while we both guided and observed the students. We collected feedback via questionnaires and interviews. Based on the lessons learned during the code camp we continue to develop CoRED, but what is more important we analyzed what kind of features the developers assume to see in a collaborative web based IDE. Naturally the users would like to combine all the features well-known from desktop based IDEs with the benefits provided by a web based environment such as a low barrier for starting to use the tool as well as collaboration features.

Towards the end of the chapter, we pinpoint nine essential features that should be considered when creating a real-time collaborative development environment. In addition we speculate that more such features could be found by implementing these and by doing another round of experimentation. Based on the results of this study a collaborative tools such as CoRED can offer developers more powerful environments than the current norm to develop software in teams.

Acknowledgments

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. The code camp was part of the projects TIVIT Digital Services\textsuperscript{21} and ITEA2 EASI-CLOUDS\textsuperscript{22}, supported by Vaadin inc.

References


\textsuperscript{21}http://www.tivit-services.fi
\textsuperscript{22}http://www.itea2.org/project/index/view/?project=10078


REFERENCES


