

A Comparison of Communication Technologies to Support Novice Team Programming

Davor Čubranić

Margaret-Anne D. Storey
University of Victoria, Canada

Jody Ryall

{cubranic, mstorey, jryall}@uvic.ca

ABSTRACT

This paper describes an initial investigation of how different conditions for conducting a team programming exercise impact learning. We conducted a series of in-depth case studies on the use of various communication technologies and compared them with face-to-face case studies of team programming. We explored how these communication technologies can help improve students' learning. We summarize the findings from these studies and give guidance to instructors and to tool designers on how future tools can be improved to support collaborative learning in team programming.

Categories and Subject Descriptors: D.2.6. [Software Engineering]: Programming Environments—Integrated environments; H.5.3 [Information Interfaces and Presentation]: Group and Organization Interfaces—Computer-supported cooperative work; K.3.2 [Computers and Education]: Computer and Information Science Education—Computer science education

General Terms: Human factors, Design

Keywords: teaching programming, Gild

1. INTRODUCTION

In the software engineering education community it is well recognized that learning how to program is challenging for many students. Most educators agree that extensive hands-on experience is required to learn programming. Working in groups is recognized as one effective way to increase a learner's knowledge and achieve deeper learning [5, 7]. Student pair programming, a practice borrowed from extreme programming that involves students alternately taking turns writing code while the other student advises and explores the problem space, has been explored in recent years [8]. Despite evidence indicating its benefits, many students and instructors are reluctant to use it. This reluctance stems in part from social issues such as mismatched pairs with respect to skill and a perceived intrusive collaboration. Team programming, however, is more generally accepted, and yet relatively little research has been conducted into how team programming exercises can be engineered to facilitate collaborative learning and critical thinking.

From our experiences as educators, we have unfortunately witnessed that team programming often results in some students having to *do less*, rather than to *learn more*. On surveying and interviewing students, we learned that the cause is often logistical issues, such as the ability to meet at a common time in a convenient place. To address some of these logistical issues in collaborative

programming, we have been investigating how to use technology to not only facilitate distributed team programming but also actively engage students in the collaborative learning process. There has been some research into the effectiveness of communication technologies to facilitate distributed team programming, but none that we are aware of that has also considered the impact on learning. Our long term interest in this research problem is that we want to provide learning support for collaborative programming exercises into an integrated learning and development environment.

This paper describes an initial investigation of how different conditions for conducting a team programming exercise impact learning. We conducted a series of in-depth case studies on the use of various communication technologies, including VNC (shared screens), instant messaging (IM), and Skype (voice over IP), and compared them with face-to-face case studies of team programming. We summarize the findings from these studies and give guidance to instructors and to tool designers on how future tools can be improved to support collaborative learning in team programming. The results from this study also pave the way for further investigations on this topic.

2. GILD

Gild is an integrated development environment for teaching and learning programming that has been developed by our group [4]. Gild was designed to simplify and add pedagogical support to an existing "industrial-strength" IDE, Eclipse, to make it more appropriate for novice programmers and their instructors. For the students, Gild reduces and simplifies the user interface, clarifies the steps in the save-build-run-debug process, and offers novice-friendly explanations of common Java syntax errors. It also supports active learning through its integrated web browser that allows the students to jump from the assignment instructions into editing and running code. For the instructors, Gild provides support for grading assignments and for creating course units that can be easily distributed to students and imported into their IDE workspaces.

Since Gild's release in January 2004, it has been used in introductory classes at the University of Victoria, as well as at several universities. We collected feedback on its usefulness, ease of use, and desired new features through focus groups and course experience questionnaires. The initial experiences at the University of Victoria are very positive. One of the most requested features has been support for collaborative learning.

Our current goal for Gild is to extend it with groupware features to support collaborative learning. As a first step in supporting collaborative learning, we extended Gild with a code-sharing feature. The prototype is implemented as an Eclipse plug-in that extends the existing user interface in Gild with a conceptually simple front end to a source code versioning system. Students working in Gild

can share their work with their teammates through two operations: by *uploading* the code changes contained in their workspace into a shared repository, and by *downloading* the most recent changes from the repository into their local workspace. The user interface includes a “diff” facility to allow the user to preview the effect of a download or upload operation on the local workspace and the repository, respectively.

The prototype is described in more detail in an earlier publication [2], along with a preliminary usability evaluation. This paper builds on that work by expanding the user study and conducting an in-depth analysis of the collected data.

3. STUDY

We recognize that although the code sharing feature of Gild should help students coordinate their coding activities, it would not necessarily help them in their collaborative problem solving activities as it lacks communication support [5]. Our goal now is to understand how it should be extended with other communication modalities offered by common third-party tools, such as MSN IM.

We designed an exploratory study as a set of case studies with a controlled programming task in a broad variety of conditions. We collected a wide range of data to help us identify the issues students face while learning collaboratively. Considering the early stage of research, we chose to control some aspects of the study, such as the assigned task and communication technology available to participants, but did not attempt to, for example, constrain participant variability or create a statistically sound experiment. We adopted a multiple case study approach in which we videotaped students as they were collaborating and recorded their communication, as well as gathered their own experience of the assignment by interviewing them immediately afterward. We thus gathered from a variety of sources a rich collection of data that we could then analyze using a combination of qualitative analysis and descriptive statistics.

3.1 Setup

Participants in the study were twelve unpaid volunteers recruited from students enrolled in the second part of a two-course introductory computer science sequence. The participants were asked to sign up as a pair, although one pair (P5) was formed from students who signed up without a partner. The participants in each pair were asked to collaborate on a programming assignment in which they were given a skeleton code for the computer game Pong, and had to implement the missing methods in four classes.¹

The pairs were randomly assigned to one of four conditions in which to work on the study task: face-to-face (two pairs: P1-F2F and P3-F2F), text-based instant messaging using MSN Messenger (also two pairs: P4-IM and P5-IM), audio-conferencing using Skype (one pair, P3-S), and instant messaging with screen sharing using VNC (one pair, P6-VNC).² Pairs in the F2F condition were located in a single room and could talk to each other directly. In other conditions, pairs were located in different rooms and had to communicate using the condition’s communication tool, Skype or MSN Messenger, respectively (see Fig. 1c-e). The pair in the VNC condition could, in addition to IM, also see the contents of each other’s screens. Even though the VNC software allows remote users to control the “host’s” desktop using their mouse and keyboard, the remote desktop was view-only. This VNC setup approximates the behaviour in the face-to-face situation where partners can monitor each other by “peeking over their shoulder” but

¹The code for the task was based on Grant Braught’s model assignment presented at CSE’03 [6].

²<http://www.realvnc.com>

hardly ever take over their computer even when they come over to point to something on the screen.

In all conditions, each member of the pair had his or her own workstation running Gild in which to work on the code. The code could be shared between the partners by uploading and downloading it to/from the shared repository, as described in Section 2. Pairs in the F2F condition were told that they could collaborate any way they wished, but that we wanted them to each work on at least part of the code individually. To further encourage individual work, we placed their workstations at a 90° angle to each other, separated by about 2 metres (see Fig. 1b).

The study task took one hour. All pairs started the session with both participants in a single room for the first 15 minutes, during which time they could discuss the assigned task but not write any code. In the non-F2F groups, one member was then moved to a different room, restricting the team to the respective computer-mediated communication (CMC) tool for the rest of the session. Pairs in all conditions had a further 45 minutes to work on their task, at which point we asked them to stop working and collected the code in their workspaces and the repository. During the study task we also videotaped the participants and recorded the contents of their screens using a screen capture program.

4. RESULTS

In addition to the code submitted by the students, we also collected data which consisted of approximately 18 hours of screen-captured video, two hours of video footage recorded in face-to-face sessions, and three hours of post-task interviews. We first report on their task performance, and then briefly summarize how we conducted a qualitative analysis of their communication events in the different conditions. We then report on the key findings per condition with respect to the participants’ skill. General observations across conditions and the interpretation of potential impact on learning are deferred until the next section of the paper.

4.1 Task performance

As expected, there was a large variation in the students’ programming ability and performance on the assignment. Only one of the pairs (P4-IM) finished the assignment in the time available, while others made varying degrees of progress. The least progress was made by the other pair in the IM condition, P5-IM, where only one class was completed and another started.

The main reason for the difficulty most students had with the assignment is that the Model-View-Controller paradigm that was used in the Pong code was new to the participants. Despite having provided documentation that described the system and specified the exact behaviour of code they had to write, many participants had difficulty understanding the “division of labour” that was required to modify the relatively simple code in the game model. The successful pair recognized this immediately and described the assignment as “very easy.” The difficulties that the participants encountered, however, encouraged them to communicate extensively.

4.2 Communication Measures

The goal of this exploratory study was to investigate the impact of collaborative technologies on learning. However, measuring learning is very difficult to do reliably in practice. Many researchers recognize that even exams tend to evaluate surface learning, and that deep learning is not something that would surface until long after a course has finished [5]. Consequently we decided to instead identify evidence of ‘critical thinking’ by capturing the transcripts of the students’ communication events and by interviewing them on their perceptions of the benefits of the technologies.

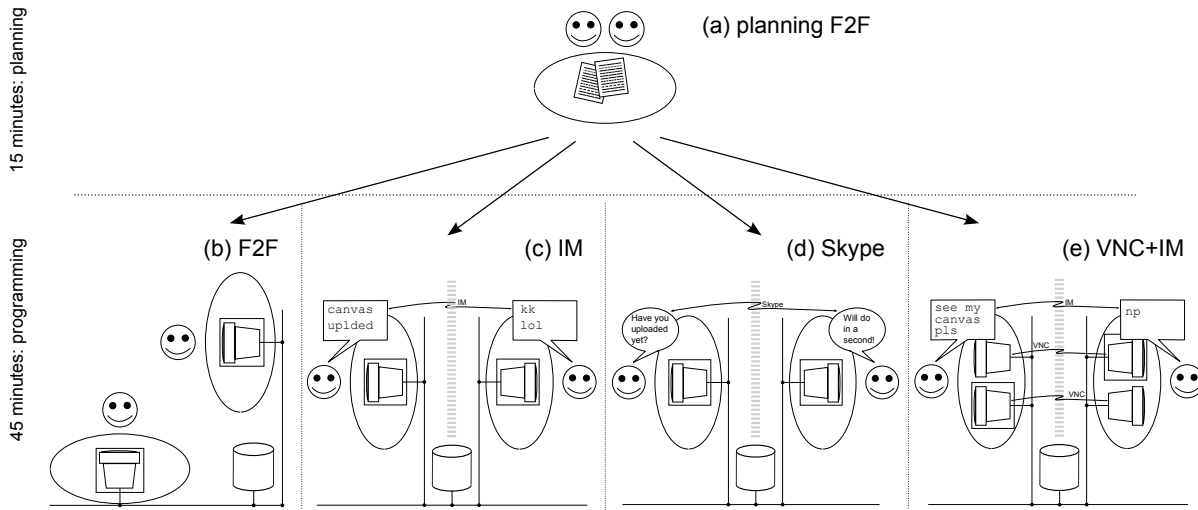


Figure 1: Study setup.

To conduct a qualitative analysis of the communication events in the different conditions, we collected the measures of the number of *turns* that members of each pair took speaking, and divided the communication into *episodes* which are a series of one or more turns on a single topic. We then classified the turns by type: seeking/giving explanation on writing code or Pong game design and behaviour; coordination (planning, seeking/giving activity information, acknowledging a turn); and other. We also looked at the number of occurrences a specific location in the code was referred to, either verbally or by pointing at it on the screen. The classification criteria used were based on a more general classification developed by Price for analyzing verbal interactions in a collaborative activity [7]. Our goal through our qualitative analysis was to look for indications that the different media supported different types of communication activity. The data is summarized in Table 1.

Pair	Turns	Episodes	Coordination	Explanation	Pointing
P1-F2F	142	24	11	13	28
P3-F2F	242	21	10	11	30
P2-S	225	32	7	19	7
P4-IM	138	31	22	10	5
P5-IM	29	9	4	6	3
P6-VNC	33	9	3	7	1

Table 1: Pairs’ communication measures.

The results, at face value, seem to indicate that the further one moves away from the most natural interaction (i.e. face to face), the more the ability to exchange information is reduced.

4.3 Communication conditions and the results

F2F: Neither of the F2F pairs successfully completed the task and both had trouble understanding concepts required to complete the task. Nevertheless, they were easily able to coordinate their activities, and actively explored the solutions they were striving to achieve. What is noteworthy is the extent at which the students gestured and pointed at the code. Both pairs would frequently congregate around a single display and discuss alternative solutions.

Skype: The single Skype pair did not successfully complete the task. Nevertheless, the audio connection enabled them to communicate very effectively during the task, especially once they were

used to what was a novel communication technology for this pair. Occasionally we saw evidence that it was cumbersome for the participants to indicate specific locations of the code being discussed, but they were always eventually able to successfully indicate the parts of the code of importance. This pair experienced an almost constant stream of communication throughout the task – in part because of the difficulty they had tackling the task.

IM: For two of the pairs, P5-IM and P6-VNC, we noticed that the instant messaging technology was used to facilitate more coordinations and was used less to explore the problem space. Pair P4-IM on the other hand had very rich interactions using this technology. The students in this pair had the most programming experience and were also frequent IM users. However, we did note difficulties referring to specific locations in the code, even for this pair and differences in the depth of their dialogue on the problem to be solved.

VNC+IM: As just noted, the VNC+IM pair also did not make as much use of the instant messaging technology as we would have expected. We were also surprised to see that the participants in the VNC condition seldom chose to review their partner’s code by looking at it on the VNC screen. Instead, they would use the code-sharing feature to get that code from the repository and look at it inside their own IDE. We queried on this in the post-study interview. The students did not know why they did not use the VNC display. Possible reasons are explored in the next section.

5. DISCUSSION

The communication measures presented above do not tell the entire story, however. Our analysis of video recordings and interviews yielded a number of observations that better illustrate the relative advantages and disadvantages of the different communication media, and that we believe will be useful for designers of collaborative development environments (CDEs) for beginning programmers.

Providing explanations: Perhaps not surprising for as focused an activity as the task used in this study, our participants concentrated on getting the code working in the relatively short amount of time that they had. Help given to a partner who had trouble getting his code to work tended to be very code-centric. Specifically, advice on fixing syntax errors never included an attempt to explain the underlying language concepts (for example, the difference between class and instance methods). Instead, the advice usually consisted

of instructions on how to change the code in question to eliminate the error message, even when it was wrong in terms of design.

We did, however, observe that having an audio channel (either F2F or Skype) made it more conducive to have discussions of why something did or did not work. Such discussions are arguably the type of peer interaction that can lead to deeper learning, and episodes including such discussion were both longer and more frequent in F2F and Skype pairs. IM, on the other hand, resulted in very terse directions that would make it far less likely that they would improve the recipient's understanding of important concepts.

The importance of gestures: As observed by Churchill and Bly [1], people collaborating over artifacts tend to use a lot of gestures to point at the artifacts when they are co-located. We observed similar behaviour in the two F2F pairs. Those pairs used a wide range of gestures: pointing at the screen, mimicking the motion of game elements by hands in the air, "drawing" with a finger on the desk, and sketching on a piece of paper. In one instance by pair P3-F2F, the two participants started a particularly animated discussion where one of them drew on a piece of paper, the other one then took the pen and paper away so she could illustrate her point, and finally returned the pen to her partner to finish the discussion. Clearly, these communication techniques are not available using technologies available to participants in the non-F2F conditions. Instead, those participants had to verbally describe the location about which they were communicating, which could be quite cumbersome:

[W]hen I want to talk specifically about this line or that line, now I see there is a number there, I could say the line number, but at that point it's like— Maybe, I was thinking if he was here I can point out this line, and say what I want to say, but using the MSN to type it, or something like that, is kind of difficult to communicate. (P5a-IM)

It should be noted that most existing software for remote desktop sharing, including VNC, includes remote mouse pointers, which can be used for pointing and gesturing. Although this feature is applicable only for synchronous collaboration, we plan on conducting further studies in which it is available to the participants.

The effect of IM expertise: Of the three pairs who had IM available, only in P4-IM both partners were regular, intensive IM users, while the other two pairs were composed of one regular and one infrequent IM user. During the task, P4-IM developed a communication pattern typical in IM, with frequent and short exchanges providing a continuous stream of activity updates. This pair had over three times as many communication episodes as the other two (31 vs. 9) and four times the number of communication turns. Even more importantly, in the post-study multiple-choice questionnaire, both participants in P4-IM stated that they "always" knew what was going on with their partner, as opposed to P5-IM, where both stated "usually," and P6-VNC's "most times."

The drawback to this kind of communication pattern is that it requires constant switching between the IM window and the IDE, and leaves no time for quiet reflection. Since the P4-IM pair knew the course concepts well and had no trouble with the assignment, they arguably in this case did not need time to reflect more deeply. It would be interesting to see whether this kind of IM communication would help or hurt a pair of similarly experienced IM users who had more trouble with the assignment. While Grinter argues that youth today have learned how to multi-task effectively and integrate IM chat into their everyday life [3], there is also something to be said to be able to concentrate on a task without being interrupted.

Foreign language students: There are factors other than communication bandwidth that affect a medium's usefulness in a given

situation. In some cases, these factors may be surprising at first but can ultimately be more important for a system's adoption. In this case, we found that non-native speakers may prefer to use IM rather than an audio channel, regardless of their other relative advantages and disadvantages for collaboration in programming. As one of our participants said in the post-task interview:

[Using IM] is more clearer to other people. It's better for me, I'm not first language, I speak second language, so I think it's harder for people to hear me. It's easy to just see what I write. (P1a-F2F)

Given the significant proportion of ESL speakers in computer science and related disciplines, this is an important point that should be taken into account when designing a learning CDE. As we pointed out earlier in this section, there are ways to improve on existing IM tools and make them more useful for collaborative learning of programming. On the other hand, there are no technological fixes for user preference for a given tool because of his or her facility in a second language. As CDE designers, we need to be aware of both.

6. CONCLUSIONS

In summary, we observed that the face-to-face setting was the most successful with respect to coordination and knowledge construction activity. The ability to point and gesture was very convenient and was frequently used in the F2F conditions. Audio followed closely behind in effectiveness for supporting both coordination and knowledge construction. Audio could be further improved by combining it with VNC. IM was very effectively used by the IM experts but it lacked some of the richness of the F2F and audio conditions. Awareness of code upload and download status was a challenge in all conditions, and more visibility of the status should be provided in the environment. Combining IM with VNC support, as well as with anchored chats, could be a benefit for both novice and expert programmers.

Importantly, while more controlled studies would be valuable, we believe that the technology is currently at a stage where it is usable enough that it could conceivably be deployed in a class and research conducted by using questionnaires to collect student experiences.

7. REFERENCES

- [1] E. F. Churchill and S. Bly. It's all in the words: supporting work activities with lightweight tools. In *Proc. of GROUP'99*, pages 40–49, 1999.
- [2] D. Čubranić and M.-A. Storey. Collaboration support for novice team programming. In *Proc. of Group'05*, pages 136–139, 2005.
- [3] R. E. Grinter and L. Palen. Instant messaging in teen life. In *Proc. of CSCW'02*, pages 21–30, 2002.
- [4] D. Myers et al. Developing marking support within Eclipse. In *OOPSLA Eclipse Technology Exchange*, 2004.
- [5] D. R. Newman. How can WWW-based groupware better support critical thinking in CSCL? In *Proceedings of the 5th ERCIM/W4G Workshop*, 1996.
- [6] N. Parlante et al. Nifty assignments. In *Proc. of SIGCSE'03*, pages 353–354, 2003.
- [7] S. Price et al. A new conceptual framework for CSCL: Supporting diverse forms of reflection through multiple interactions. In *Proc. of CSCL'03*, pages 512–522, 2003.
- [8] L. Williams et al. In support of pair programming in introductory computer science courses. In *Computer Science Education*, 2002.