Software Engineering Projects in Distant Teaching

Philipp Bouillon Jens Krinke Stephan Lukosch
FernUniversität in Hagen
Germany

Abstract

Software engineering education is most often complemented by a software engineering project where a team of students has to develop a large software system. At a distance teaching university such projects challenge the students in communication, coordination, and collaboration, because team members work in different places, many miles away from each other. We present an ECLIPSE-based unified platform that leverages available tools and solutions and discuss the problems involved. Besides using plug-ins that support the students during implementation, our platform integrates a collaborative distant education environment and a software project management system that eases the students' collaboration in the software engineering project.

1: Introduction

Teaching software engineering at a university is a multi-level education starting with teaching programming up to teaching advanced topics. Software engineering education has to be accompanied with hands-on experience, typically in software engineering projects where the students have to work in teams to develop a complete software system. Such projects have the goals that students (1) design, validate, verify, implement, and maintain software systems, (2) understand processes and models, and (3) obtain and improve team and communication skills. A major requirement is that the projects are realistic, i.e. the system to be implemented is of a non-trivial size and the students use realistic (or even better, real) tools.

At the FernUniversität in Hagen, the first distance teaching university in Germany, most software engineering courses are taught completely as distant learning courses (online or via snail-mail). However, the software engineering projects in the graduate level have to pay special attention to problems arising from the distribution of our students, because the students are not working together at the same place, but rather they are all working at home, possibly hundreds of miles away from the others. At present, the students visit the university in Hagen to form their teams and to be introduced to the project circumstances. Then they return to their respective homes, implement their part within the IDE of their choice, and discuss their work and communicate via a system called CURE (Collaborative Universal Remote Education environment) [6], the CSCL (Computer-Supported Collaborative Learning) platform of the FernUniversität. To allow for the collaboration of a group during the project, the group must be supported in terms of *communication*, *coordination*, and *collaboration*. Those areas are often used to classify groupware. Communication essentially means the exchange of information between the students. Coordination is used to synchronize shared tasks, while collaboration finally addresses the achievement of group goals. From the description given above, several problems for the FernUniversität arise:

Communication. Students at the FernUniversität usually use electronic means to communicate, i.e. mail, instant messaging, and telephone. However, as all students have individual working habits, most communication is *asynchronous*.

Coordination. Because of the distance between the students, a high need for coordination arises to manage tasks and to allow for collaboration. Synchronization processes are often supported by communication means or by shared artifacts (e.g. a shared project plan).

Collaboration. Because students do not work on campus (or even better, in central computer pools), they also have to use electronic means for collaboration. Even worse, because of asynchronous communication, they need support to find out *what has been done in the project* and *what needs to be done*.

Henceforth, we will use the term *cooperation* as a substitute for communication, collaboration, and coordination.

We created a uniform environment based on ECLIPSE which allows for students located in different areas to collaborate in a team to produce a large software system. To accomplish this goal, we incorporate existing tools into ECLIPSE and provide some *gluing-code* which allows for the plug-ins to work together. "Traditional" universities will benefit from the IDE as well, because the processes taught and supported by the IDE also apply. For example, two of the major problems of software engineering projects at universities, communication and project management [4], are eased with this environment.

Section 2 gives an overview about the current situation in teaching software engineering with Eclipse including a detailed description of the problems involved. Our approach of an Eclipse based IDE to support distant teaching is depicted and discussed in Section 3. Section 4 presents future work while finally Section 5 concludes the article with a summary.

2: The situation today

Already at undergraduate level, a computer science student will be taught how to write a program in at least one programming language. Those lectures usually aim to instill an understanding of the concepts of the programming language so that the student will technically be able to develop large-scale programs without further guidance by the lecturer. To improve the understanding of large-scale software systems, theoretical software engineering lectures are held which confront the students directly with the complexity of large systems.

Pure theoretical study of software engineering does not suffice to gain a feeling for cooperation problems which arise during the development of a software project. Computer science students tend to focus on the technical problems involved and trust that social conflicts will not arise in their project development process. To remedy this situation, many universities, including the FernUniversität, offer practical software engineering courses where a group of students has to develop a larger software system as a team. To facilitate those practical courses, a variety of tools exist. For ECLIPSE, there are as yet not many plug-ins that support collaboration processes. Some plug-ins, however, exist that integrate available instant messaging solutions like the IM-Plug-in[1] or PepeMax[2]. But collaboration is much more than just instant messaging; other plug-ins and projects like Jazz [3] seek to integrate collaborative capabilities into ECLIPSE. However, its focus is synchronous communication and collaboration of teams working in close proximity; a situation different

[1]http://eimp.sourceforge.net/d/

[2]http://pepemax.jabberstudio.org/

than the one found in distant teaching. On the other hand, the cooperative learning platform of the FernUniversität already provides support for synchronous and asynchronous communication and collaboration. This system, called CURE [6], facilitates collaborative learning in distributed teams using standard browsers over the Internet. CURE is based on the *room-metaphor* which is commonly used to structure collaboration [5, 8]. Additionally, CURE combines ideas from web-based working spaces (wiki) [7] with communication and coordination tools. Those tools include an e-mail system, document management and a calendar with an appointment synchronizer. Students at the FernUniversität are already used to this system so an integration into our new platform for software engineering projects is sensible.

Any software engineering project within distant teaching is always *distributed software engineering (DSE)*. DSE is supported by project management or groupware solutions ranging from communication and collaboration support to software-centric solutions typically found in open source projects, e.g. SourceForge[3] or GForge[4]. These seem to be a good infrastructure for software engineering projects and indeed, some universities use GForge for that purpose. However, we have decided to use a commercial variant, CodeBeamer[5] from Intland, due to its improved usability and increased functionality.

The third major requirement is supervision, observation, and grading. The teacher must be able to supervise the groups' advances, guide the students through the project, and quickly identify problems to offer help or intervene. For this purpose and for the later grading, she must be able to analyze the communication in the group, the rendered documents, and the developed software. There exists some support to analyze software inside and outside ECLIPSE which can be used by the teacher. The JRefleX project [10] provides plug-ins for collaboration analysis and evolution analysis.

So, new plug-ins have to be developed and existing plug-ins have to be grouped together and *glued*, so that they are aware of each other and facilitate the collaborative working of students in software engineering projects.

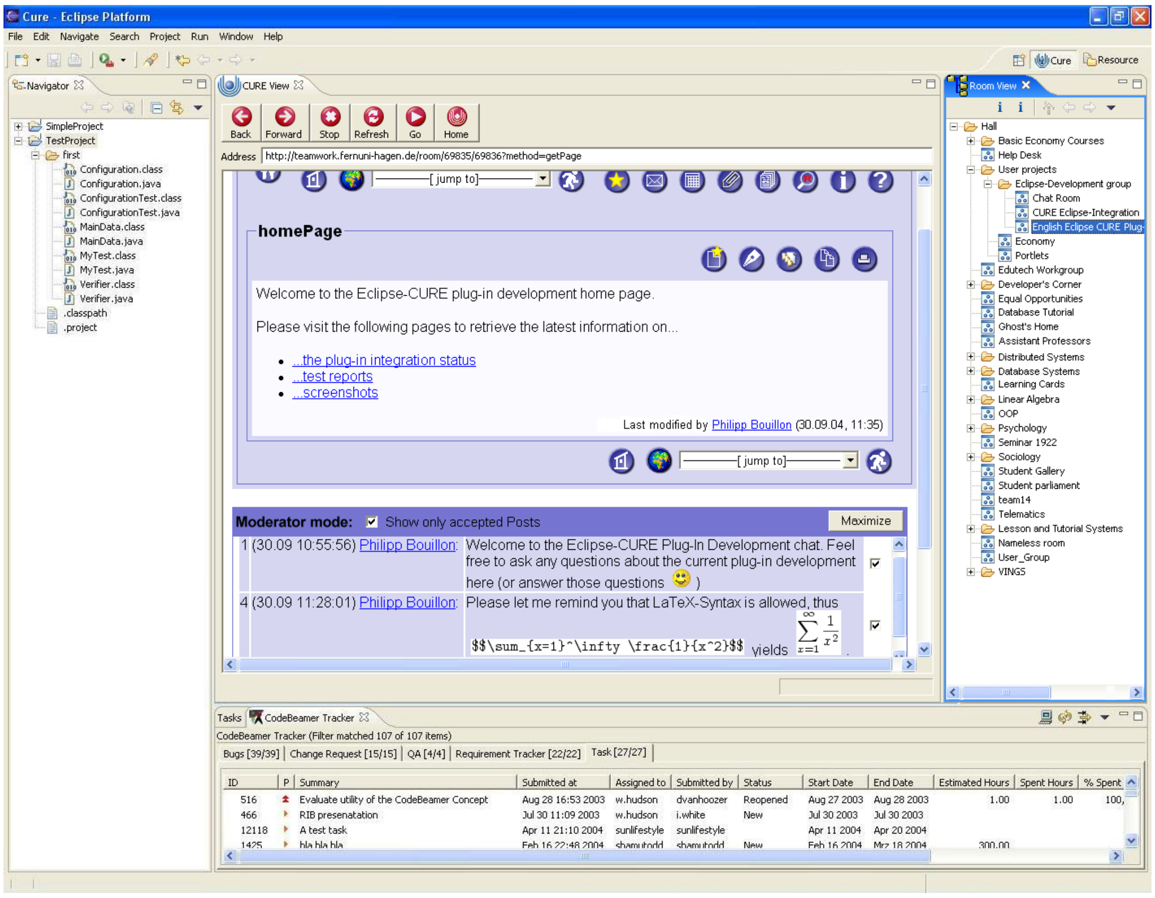
3: An IDE for distant teaching

The first question that arises when discussing a new IDE which is supposed to support teamwork for students that are not close together is: *Which functionality is needed?* Although the question is rather simple, the answer is not. Students (as well as professionals) all pursue a different methodology when working: Some tend to prefer working in the night, some get up early to get some work done before breakfast, some tend to plan first and then implement, while again others may implement and then test it. So, how can an IDE help to support the preferred working technique of a programmer without making the team depend on a certain methodology? The key issues in this setting are *communication*, *coordination*, and *collaboration*. As previously discussed, two systems will have a major role in software projects at the FernUniversität: The CURE and the CodeBeamer system; both facilitate cooperation.

3.1: Cooperation in CURE

CURE [6] facilitates collaborative learning in distributed teams over the Internet. Users can access shared information independent of their current location. Distributed teams can use CURE

[3]`http://www.sourceforge.org/`
[4]`http://www.gforge.org/`
[5]`http://www.intland.com/`



Address http://teamwork.fernuni-hagen.de/room/69835/69836?method=getPage

homePage

Welcome to the Eclipse-CURE plug-in development home page.

Please visit the following pages to retrieve the latest information on...

- ...the plug-in integration status
- ...test reports
- ...screenshots

Last modified by Philipp Bouillon (30.09.04, 11:35)

Moderator mode: ✓ Show only accepted Posts

1 (30.09 10:55:56) Philipp Bouillon:	Welcome to the Eclipse-CURE Plug-In Development chat. Feel free to ask any questions about the current plug-in development here (or answer those questions 🙂)	✓
4 (30.09 11:28:01) Philipp Bouillon:	Please let me remind you that LaTeX-Syntax is allowed, thus `$$\sum_{x=1}^\infty \frac{1}{x^2}$$` yields $\sum_{x=1}^{\infty}\frac{1}{x^2}$.	✓

CodeBeamer Tracker (Filter matched 107 of 107 items)

Bugs [39/39] | Change Request [15/15] | QA [4/4] | Requirement Tracker [22/22] | Task [27/27]

ID	P	Summary	Submitted at	Assigned to	Submitted by	Status	Start Date	End Date	Estimated Hours	Spent Hours	% Spent
516	▲	Evaluate utility of the CodeBeamer Concept	Aug 28 16:53 2003	w.hudson	dvanhoozer	Reopened	Aug 27 2003	Aug 28 2003	1.00	1.00	100,
466	▸	RIB presenatation	Jul 30 11:09 2003	w.hudson	i.white	New	Jul 30 2003	Jul 30 2003			
12118	▸	A test task	Apr 11 21:10 2004	sunlifestyle	sunlifestyle		Apr 11 2004	Apr 20 2004			
1425	▸	bla bla bla	Feb 16 22:48 2004	shamutodd	shamutodd	New	Feb 16 2004	Mrz 18 2004	300.00		

Figure 1. Eclipse with CURE and CodeBeamer View

to organize and manage their own shared learning processes by cooperatively editing the learning material presented in their shared spaces. They can then communicate via synchronous and asynchronous means. A group is defined as the set of all users of a shared workspace. Shared workspaces and their access rights in CURE are modeled by using the room-metaphor. The access to a working space is thus managed by the key-metaphor. To become a member of a group which accesses a certain working space (that is: room), a user must possess the appropriate key for that room. By combining rooms, keys, the rights associated with keys, and the operations defined on keys, various forms of group forming are supported in CURE. So, users can create rooms for specific groups and purposes. Room owners can restrict access rights. A room contains pages, resources and cooperation tools, which are created, manipulated, navigated and read by users of the room. A simple wiki syntax is used to write the content of pages including formatted text, images, and TEX for expressing mathematical formulas. Each room may have its own chat and room mailbox that are kept persistent. All users in a room can simply chat with all other online users in that room, or view and send mails to the discussion threads in the room's mailbox. An integration of NetMeeting[6] and DyCE [9], a component based groupware framework, allows for the synchronous cooperation of the users. Previous experience with CURE in software engineering projects showed that students just use the asynchronous features of CURE. This has two reasons: (1) synchronous communication is rare (as explained above) and (2) students reverted to widespread instant messaging solutions.

[6]`http://www.microsoft.com/windows/netmeeting/`

The instant messaging tools still fit on the screen with the student's IDE. In contrast, using CURE always enforced a context switch to the browser. Therefore, we developed an ECLIPSE plug-in that integrates a CURE View into ECLIPSE. A sample screen is shown in Figure 1. The middle view shows a page of a room and the tree view on the right shows the rooms currently available to the user. A software project is mapped to a room, while its pages represent the sub projects, milestones, and various other topics.

3.2: Project Management

Software projects in distant teaching need a strict project management. Experience shows teams using CURE for project management had better results than teams using implicit project management. However, project management with CURE is very cumbersome in comparison to project management tools. For this purpose we have chosen CodeBeamer, a server based software development solution with comprehensive collaborative features for development teams. It is a ready-to-use solution with light-weight project management based on *trackers*. These trackers can be used to manage requirements, tasks, bugs etc. and can be visualized in diagrams (e.g. gantt charts). This approach eases the traceability for the participating students and the teacher. The tracking system is able to present the list of things that have been worked on lately or that have to be tackled with next. CodeBeamer comes with a plug-in that integrates the trackers with ECLIPSE.[7] This service is provided by a server located at the FernUniversität and thus it can be accessed by any team member currently taking the software engineering course; besides the plug-in there is no software to be installed by the student. Figure 1 shows CodeBeamer's trackers in the bottom view.

3.3: Phases of a project

Besides the two major components described above, we have evaluated some available plug-ins for usage in a (distant teaching) software engineering project. There is varying support for the different phases of a software project, which we will present next. It is worth noting that the mentioned phases are coarse grained and flexible to accommodate traditional or agile process models during the project.

Phase 1: The creation of the project. When creating a project, a supervisor must be able to enter a project description and mark the milestones at which certain parts of the project have to be completed. A common project description can be put to a project room in CURE, which is accessible by the project group members. A more comprehensive description, i.e. including milestones, is achieved by using the task tracker of CodeBeamer, where every task can have an end date. It would be desirable to automatically enter those dates into a team-calendar which is shared by each team member, allowing for the addition of team-internal dates. We have not found a plug-in that provides calendar functionality. The optimal solution for this kind of (simple) calendar would be to integrate ECLIPSE with MS Outlook or IBM Lotus Notes since these are the applications used by most people to organize their meetings. Present open-source project management and/or cooperation solutions provide their own calendars which cannot be synchronized with others. Currently, there are even two calendars in the university's platform, one in the administration and e-learning platform and one in CURE; both are independent and cannot be synchronized. Neither one is used extensively by the students due to the lack of synchronization possibilities.

[7] We originally planned to use GForge as a free alternative, however, due to its architecture it is not easily integrateable into ECLIPSE.

Phase 2: Requirements analysis. In the beginning of a project, most activities imply discussing issues and documenting requirements. To allow for a vivid discussion among team members, both, synchronous and asynchronous communication must be provided and the final requirements document should be produced in the same environment without the need to constantly switch between applications. Therefore, a chat and mail functionality, as well as wiki-pages where team members can jot down their ideas is ideal. Most of this is available in CURE where students can discuss the requirements and can use the wiki pages to produce their resulting document. The CURE plug-in provides chat, mail, and wiki in the same style as the standalone CURE system. As CURE furthermore provides a document versioning system, it is also possible to retrieve older versions of any kind of document and thereby reconstruct how they evolved. The student team also has to generate tasks from the requirement. Together with the milestones given by the teacher, the students have to plan their project. For each requirement and each task the students have to generate a tracker entry. This approach eases the traceability for the students and the teacher. The tracking system is able to present the list of things that have been worked on lately or that have to be tackled next. Besides CodeBeamer, there are other tracker plug-ins, for example those that integrate BugZilla.

Phase 3: Design. A variety of UML design tools exist. However, none of the available open source tools allow for collaborative design. Very useful would be a graphical diff-view that directly shows the user, which changes were made (i.e. by highlighting modified, added, or removed elements in the diagram). Additionally, most UML-tools are not easily usable in a software engineering project, because they either have not enough functionality or are too complex for the novice user. Best suited for our needs are Together[8] and MagicDraw[9]; other tools are either not integrateable into ECLIPSE or suffer from stability problems. Furthermore, UML design can be accompanied by a chat window to support synchronous communication during the work (see discussion above). A promising project named GroupUML [2] is currently being developed at the TU München. GroupUML is a tool which allows multiple users to create and edit semi-formal UML diagrams. The tool is able to create standard UML diagrams and on the other hand supports free hand drawing and text editing. In short, GroupUML combines the aspects of a shared whiteboard with a UML diagram editor. The key feature of the tool is that it allows all team members to view what is currently being done. Additionally, a team member can immediately see what has been discussed if she joins the meeting later, because chat messages between the users are logged and kept persistently with the diagram. Even better is that those chat messages can be linked to certain elements in the working area so that a user can easily attribute a message to a portion of the diagram. We will evaluate this project and see if it can be integrated into our environment. If that is the case, we will further pursue the issue.

Phase 4: Coding and unit testing. ECLIPSE is already excellently equipped for coding purposes in JAVA. Other languages are not so well supported, yet, but this is hopefully going to change in the future. For testing (and one way to specify), JUNIT is shipped with ECLIPSE, which our students are required to use. Independent of the programming language, a team interface is integrated with ECLIPSE which allows for CVS communication, so this collaborative issue is already solved elegantly in ECLIPSE. To improve this phase, external tools like JML (a specification language for JAVA) will be provided. We also plan to evaluate the effect of code checking plug-ins like CheckStyle or PMD, debugging aids like delta debugging [1], or others. Of course, discovered bugs are managed in our setting with the CodeBeamer's bug tracker.

[8] `http://www.borland.com/together/eclipse/`
[9] `http://www.nomagic.com/`

Phase 5: Delivery and grading. At last, after the students have delivered the software together with the required documents, the teacher has the task to evaluate and grade the students. Criteria for the evaluation can be many fold: (1) cooperation skills, (2) quality of the design and the implementation, (3) accordance of the final version to the documented requirements and design, and (4) needed time. The key requirement here is traceability. In a traditional software engineering project, where the group is given a task and delivers the final software system, it is almost impossible to grade the students individually. This is different with the presented infrastructure: The teacher is able to extract the needed data from the persistent communication in CURE (chat protocols, mail archives, all versions of the wiki pages), from the CodeBeamer trackers, and from the documents and source code stored in the CVS archive. Furthermore, she can use reverse engineering tools to compare the architecture of the delivered software with the original design. A plug-in specifically dedicated to this approach is JRefleX [10] which provides collaboration analysis (how has the team worked together) and evolution analysis (how has the architecture changed over time). We plan to evaluate JRefleX in our context.

4: Experiences and Future Work

During the evaluation of the various plug-ins we have experienced that most plug-ins provide good solutions to single problems. However, in a setting like ours, it is not enough that support in ECLIPSE exists, but that the various plug-ins are more integrated and aware of each other. For example, the various instant messaging plug-ins are standalone solutions and are not interchangeable. This would require them to be based on a framework like Koi[10]. There is a similar situation in cooperation and project management solutions. Each solution comes with its own discussion forums, calendar, document management, etc.—but it is not possible to integrate them with other forums or calendars. This results in a situation where our infrastructure used for software engineering projects offers at least four(!) discussion forums: (1) traditional news groups, (2) a discussion forum in the general learning platform, (3) the mail and chat solution within CURE, and (4) the discussion forums within CodeBeamer. This situation is confusing and must be solved by a more general, integrated solution.

Up to now, CURE has once been used alone and is currently being used together with CodeBeamer to support students in a practical software engineering course. During the first practical course, 6 groups of 5 to 7 students designed and implemented a synchronous collaborative game. CURE has mainly been used to communicate and to document relevant design decisions. All groups agreed that CURE facilitated group communication but they also admitted to have failed in documenting specific tasks and responsibilities as good as they could have. Thus, the task assignments were not completely transparent throughout the group which lead to misunderstandings during the course. It turned out, however, that the group with the highest amount of communication in CURE produced the best results. In the second practical course, we have currently 3 groups with 7 students each working at extensions of the CURE platform. Those extensions will allow for synchronous collaborative learning. Learning from our experiences during the first course, we provided CodeBeamer, introduced it to the students and explained how important a clear task assignment is. The groups could then decide if they wanted to use CodeBeamer or not. One of them actually decided to use CodeBeamer while the other groups wanted to document their task assignments solely in CURE. The group using CodeBeamer used it in the very beginning of the course and made the fastest advancements in comparison to the other two groups, however, if this progress was made

[10]`http://www.eclipse.org/koi/`

because of the use of CodeBeamer has yet to be determined by a final evaluation. After the initial phase, the coordination of that group was handled in CURE. The group members argued that for the initial task assignments, CodeBeamer was valuable, however, after those tasks were clear, a weekly team chat along with the data stored in the CVS repository sufficed to coordinate the remaining tasks.

5: Conclusions

We have presented how we use ECLIPSE as the key environment for software engineering projects in distant teaching. It mainly consists of the integration of two major components: the standard cooperation platform of the FernUniversität, CURE, and the project management solution CodeBeamer. CURE has been integrated by a newly developed plug-in that enables the usage of wiki, mail, chat, and group calendar within ECLIPSE. CodeBeamer comes with a plug-in that integrates the various trackers into ECLIPSE.

First experiences have shown CURE and CodeBeamer to support distributed software development by improving the collaboration of the team. Together with additional plug-ins, this platform will ease the (distant teaching) software engineering project in all phases; students will be able to collaborate more intensively, manage their project more easily, and deliver higher quality software. Because of the heavy use of persistent communication, trackers, and version repositories, it will be easier for the teachers to grade their students individually.

The presented integrated development environment with CURE and CodeBeamer is currently being used for the first time. We have reported first experiences with the IDE. A final evaluation of the collaboration of participating groups will show if students need more tools and if they accept the provided ones.

References

[1] Philipp Bouillon, Martin Burger, and Andreas Zeller. Automated debugging in eclipse. In *Proceedings of the 2003 OOPSLA Workshop on Eclipse Technology eXchange*, pages 1–5, 2003.

[2] Naoufel Boulila, Allen H. Dutoit, and Bernd Brügge. Scoop: A framework for supporting synchronous collaborative object-oriented software design process. In *Proceedings of the 2004 ASE Workshop on Cooperative Support for Distributed Software Engineering Processes*, pages 39–53, 2004.

[3] Li-Te Cheng, Susanne Hupfer, Steven Ross, and John Patterson. Jazzing up eclipse with collaborative tools. In *Proceedings of the 2003 OOPSLA Workshop on Eclipse Technology eXchange*, pages 45–49, 2003.

[4] Michael Gnatz, Leonid Kof, Franz Prilmeier, and Tilman Seifert. A practical approach of teaching software engineering. In *Conference on Software Engineering Education and Training*, pages 120–128, 2003.

[5] S. Greenberg and M. Roseman. Using a room metaphor to ease transitions in groupware. In M. Ackermann, V. Pipek, and V. Wulf, editors, *Beyond Knowledge Management: Sharing Expertise*, Cambridge, MA, 2002. MIT Press.

[6] J. M. Haake, T. Schümmer, M. Bourimi, and B. Landgraf. Supporting flexible collaborative distance learning in the CURE platform. In *Proceedings of the Hawaii International Conference On System Sciences (HICSS-37)*, 2004.

[7] Bo Leuf and Ward Cunningham. *The WIKI way*. Addison-Wesley, Boston, MA, USA, 2001.

[8] H. Pfister, C. Schuckmann, J. Beck-Wilson, and M. Wessner. The metaphor of virtual rooms in the cooperative learning environment CLear. In N. Streitz, S. Konomi, and H. Burkhardt, editors, *Cooperative Buildings*, LNCS 1370, pages 107–113. Springer-Verlag Berlin Heidelberg, 1998.

[9] Daniel A. Tietze. *A Framework for Developing Component-based Co-operative Applications*. PhD thesis, Technische Universität Darmstadt, 2001.

[10] Kenny Wong, Warren Blanchet, Ying Liu, Curtis Schofield, Eleni Stroulia, and Zhenchang Xing. Jreflex: Towards supporting small student software teams. In *Proceedings of the 2003 OOPSLA Workshop on Eclipse Technology eXchange*, pages 50–54, 2003.