Developing a Pedagogical Infrastructure for Teaching Globally Distributed Software Development

Ed Keenan  
DePaul University  
243 South Wabash  
Chicago, IL, USA  
ekeenan2@cs.depaul.edu

Adam Steele  
DePaul University  
243 South Wabash  
Chicago, IL, USA  
asteele@cs.depaul.edu

ABSTRACT
Teaching students about Globally Distributed Software Development (GDSD) is becoming increasingly important as a significant percentage of current projects are being developed by globally distributed teams. We discuss the pedagogical infrastructure used to teach GDSD at DePaul University and its partner institutions. We cover the educational, technical and institutional challenges that need to be resolved in order to successfully partner with globally distributed teaching institutions in order to teach GDSD in a realistic environment.

Keywords
Globally distributed software development, industry experience, pedagogical resources, educational patterns, collaborative education

1. INTRODUCTION
In this paper we discuss the challenges of teaching Globally Distributed Software Development (GDSD) in a university environment. We show how our approach using teaching patterns exposes the students to situations that are motivated by real GDSD experiences[4]. We first cover the detail of the pattern, the teaching methodology and the pedagogical results we achieved from each pattern. We then discuss how a technology infrastructure using communication and Software Configuration Management (SCM) tools can mitigate some of the common problems encountered by GDSD teams. The technology infrastructure is important because it allows the students to communicate and share artifacts, while allowing the teaching staff visibility into the functioning of the teams. We finish off with a discussion of some of the institutional challenges that DePaul has faced with its partner institutions in getting a GDSD education course off the ground.

INDUSTRY OBSERVATIONS
Developing a Pedagogical Infrastructure for Teaching Globally Distributed Software Development

2. TEACHING GDSD AT DEPAUL
At DePaul University’s School of Computing and Digital Media (CDM) we have developed a GDSD course targeting masters level students in software engineering. This course studied the challenges and solutions for developing software in globally distributed teams. Some of the topics we covered were the strategic issues related to allocation of tasks, communication issues that arise due to distance, time zone differences, infrastructure support, geographical dispersion, lack of information communication and coordination complexity. The students also studied cultural and technical issues related to information and artifact sharing, architectural design, and finally knowledge management issues. As such, this course is similar to a number of other GDSD courses[1,3,7].

The course was taught at DePaul University and at a collaborating university located in Mexico through an in class video recording system. Students at both universities had access to course source materials and course video playback. Students had a minimum background of C++ or Java and data structures.

Students were divided into teams of four or five students from each university. Each team formed a simulated software company that interacted with the other teams. The teams in the 2011 iteration of the course consisted of three teams from Mexico City and four teams from DePaul (Chicago). The Chicago teams consisted of a mixture of in class students and online students geographically dispersed across the US and in Europe.

The teaching patterns that we used provide vehicles for students to simulate real-world interactions found in software development. These teaching patterns have been developed based on the industry trends and behavior common to developing software in a global environment. The details of their development and use can be found in [5].

We first introduced a pattern (remote testing) which requires minimal coupling between the teams which allowed the students to get acclimated to the tools and the environment which was used throughout the course. In order to motivate the patterns that we used in the GDSD course we discuss some of the observations that the two co-authors have seen in over 25 years of industry experience in the gaming, avionics, defense and control system industries.

3. INDUSTRY OBSERVATIONS
Frequently a company’s resources and schedules prevent sufficient testing on their completed software system. The need to have Unit, Regression, and System testing is critical, but due to internal priorities and pressures, the design team isn’t able to complete these tasks, therefore the company will try to outsource...
the testing to a third party. Often there is a lack of high level and code level documentation for the system; and, furthermore, the design team doesn’t have time to support and monitor the remote team’s testing efforts. Therefore, the remote team needs to be able to understand the legacy code, implicit design assumptions and domain issues without external support. We explore these issues in the Remote Testing pedagogical pattern.

Another common situation in industry is the desire to extend a design team’s capacity without giving up any architectural control. In practice, however, this is very inefficient because the design team spends a lot of time trying to create a standalone design that is easy to understand. The design team provides the customer requirements, UML diagrams, use cases, collaboration diagrams, pseudo-code, APIs, etc. to provide a fully specified design. Unfortunately, creating a project this much detail, requires the design team to essentially implement the design in order to understand the solution which consumes significant local time and resources (which are usually in short supply). We show the challenges and tradeoffs that arise from this approach in the Subordinate Role pedagogical pattern.

Often a large software projects cannot be completed by one team, and time constraints and/or resource burdens force the project to be divided up so that different globally distributed teams are responsible for separate parts of the design. This improves the parallelism of development and ideally allows individual teams to focus on their areas of competence. However, two major challenges for this process are the appropriate partitioning of the project, and the later integration of the separate components[10]. We look at partitioning and integration of a large project in the Partitioning pedagogical pattern.

The practice of adding a remote team to augment the design team occurs when local resources are not available or only needed for a short period of time. A remote team is contracted for a fixed duration, and is treated as an extension of the design team. When a design team engages a remote team one of the challenges the design team is faced with is maximal utilization of both team’s resources (this is especially critical if the remote team is working on a fixed fee basis). We explore this case with the Continuous Development pedagogical pattern.

4. PEDAGOGICAL PATTERNS

1.1 Remote Testing

**Pattern:** In this pattern a design team delegates the testing to a remote team that isn’t the original designer and implementer of the software system.

**Methodology:** The student teams taking the role of the remote team were given freeware source code. The code provided compiled successfully and had sample code demonstrating correct use of the system (at DePaul we used a game physics engine provided as a companion to a textbook). The documentation provided was very minimal and existed only in the comment fields of the source code. The pattern was broken into two phases:

*Analysis phase:* Student teams had to understand code that they didn’t write. Since the code was provided and the original author wasn’t available, external questioning and support was not possible. This forced the students to understand the code by themselves. Sample tasks such as compiling the system, stepping through unknown code, writing sample code to exercise the APIs and diagramming the components all aided in the understanding of the system as a whole. After analyzing the system the students developed a plan for implementing the testing of the system.

*Testing phase:* Students subdivided the system into sections and distributed them amongst the team. Each member of the team developed and implemented tests for their respective sections. All of these tests were collected into a unified test suite.

**Results:** Students gained firsthand experience as to what documentation should be included as part any design. They learned that testing after a design has been implemented is extremely difficult and testing should be implemented incrementally as a design develops. Fundamentally, they learned that handing testing responsibilities to an external team is problematic, and may take much longer to execute than if the original team had done the testing. In addition, complete understanding of a design from another team may not be possible without that original team’s participation and support.

More interestingly, the students learning the remote testing pattern gained important insights into how the designer’s role impacts the remote team. Namely, it is very difficult for a design team to rapidly develop an application and expect full-spectrum testing without the iterative and collaborative participation of the remote team.

1.2 Subordinate Role

**Pattern:** In this pattern the design team delegates the implementation to the subordinate remote team, and the remote team implements and tests the supplied design. The design team oversees the remote team during this process.

**Methodology:** To teach the subordinate role pattern student teams took on the roles of both the design team and the remote team.

During the first iteration, the student teams played the role of the design teams, by creating the design artifacts. They did this by creating a detailed design with UML diagrams and other related material.
In the second iteration, the design was handed off to another student team. The student team acted as the remote team by implementing and testing the provided design materials as well as supporting their own design by communicating with the other team.

With this approach the teams gained the experience of implementing a supplied design, while understanding the challenges of supporting a design that another team was implementing at the same time.

**Results:** Students gained experience preparing a design for a remote team’s implementation. However, this experience brought home the fact that in many cases the cost of preparing and monitoring the design exceeded any benefits that were derived by delegating the implementation effort to the remote team.

As a result of this exercise, many students said that if faced with a similar situation they would have the remote team design and implement their solutions with approval and oversight from the design team, reducing upfront design and instead reacting to issues during the implementation.

### 1.3 Partitioning

**Pattern:** In this pattern the design team breaks a large project into discrete components which are each assigned to remote teams who are responsible for their respective design and implementation. Once all the components are implemented and debugged, they are merged into a solution.

**Methodology:** For this pattern the student design teams were given the specification of a large application that they needed to collectively design and implement. In the first phase the teams did a high-level design and subdivided the project amongst themselves. During this phase the teams implemented rapid prototypes to better understand the overall design, and based on this knowledge, teams then completed a single high-level design and subdivided the design and tasks among the teams.

In the second phase the teams designed and implemented their respective components according to the high-level plan. In this phase teams coordinated the migration and merging of the different components through a SCM tool. Solving merge issues and debugging the other teams contributions are some of the challenges that were faced by the students in this project. Their ability to correctly use synchronous and asynchronous communication tools and identify project dependencies are essential to the overall success of this project[8].

**Results:** Experiencing this pattern taught the students the importance of collaboration, handling dependencies, partitioning projects, version control, communication, and the challenges of debugging merged code in a global environment. Because many of the problems that the individual teams experienced are amplified in this environment, there is an even more critical need for orthogonal component design to reduce the inter-component dependency and to allow for independent testing and merging[2].

This pattern reinforced critical lessons about how global software teams and individuals need to successfully communicate and coordinate through wikis and other tools in order to mitigate the risks of failure which in this pattern, as in real-world global software projects, are primarily management rather than technical.

### 1.4 Continuous Development

**Pattern:** In this pattern the design team supplements their resources with those of the remote team. The remote team is contracted for a fixed fee, and is treated as an extension of the design team. Tasks are distributed to the design team or the remote team on the basis of available resources.
c) The remote team needs to have a queue of tasks waiting to be implemented

In the problems remote team was given a task that could be implemented very quickly with advance tools or existing systems. The remote team was then without a next task, and was underutilized. The design team was graded on utilization of the remote team.

**Results:** Understanding this pattern teaches the importance of continuous monitoring of the remote teams effort by the design team to minimize the delay dependencies between the design team and the remote team, and to make sure the capabilities of the remote team are matched with task requirements in order to fully utilize the remote team.

### 5. PEDAGOGICAL RESOURCES

#### 1.5 Communication Tools

Wiki pages (Google Sites) were used to setup the class communication links. This was used by the students from all the locations to find links to the class forums (Google Groups), documents, assignments, diagrams, links and material for the class. This wiki was also a centralized location for teams’ pages and the students’ individual blogs.

![Figure 6. Class Wiki.](image)

The team’s wiki pages contained all the communication links, documents, schedules, task tracking, and discussion boards for their project development. Additional communication tools such as emails, team forums, Skype, Doodle, Dabbleboard, Facebook, and Twitter were also used to coordinate between team members. The students’ individual blogs also recorded the technical and managerial issues of the project, and their reflections on GDSD.

![Figure 7. Student Blogs.](image)

#### 1.6 Development Tools

Software designs, documentations and diagrams were developed with Word, Excel and Visio. All documents were stored in web friendly formats such as the PDF, GIF and PNG formats to maximize accessibility.

Software tools, integrated development environments (IDE), analysis tools, documentation generation tools, were either supplied through Microsoft’s MSDNAA, or were used in their trial periods during the academic term. The primary development tools were the MS Visual Express Editions, Understand, Doxygen, CppUnit, Perforce and the Bugzilla SCM system.

Version control and effective code base aligning are necessary components in any software engineering project. Having the ability to efficiently merge and track changes in an active code base across distributed locations requires the basic features in a SCM tool, including atomic commits, versioning, differencing and email notification. The SCM tool should also have Codeline functionality such as forward/reverse branching, selected integration, code line merging and tracking. We also need Distributed Support and Security with support for proxy servers, remote depots, ancestry tracking and permissions control for users, directories and groups.

Perforce is a commercial SCM tool that meets these criteria, and it is used by many GDSD teams in industry because of its characteristics noted above. Proxy servers can be added to cache the server’s data locally to mask the delay of overseas transmissions or poor networking infrastructure in countries other than the USA. Remote depots allow multiple servers located locally at corresponding institutions to interconnect seamlessly with others.

One of the challenges faced by both industry and student teams is disjointed development through code branching. The functionality of Perforce allows concurrent development without breaking the mainline, because the isolated development can be successfully integrated back into the mainline is executed. Revision graphs allow visual tracking of codelines between branches and formal releases.
Both of the institutions’ administrations need to fully embrace and support the development of GDSD class such as the one we describe. There are many indirect support issues that are required to develop and sustain a course like the one described in this paper. For example, the host institution must supply and maintain a perforce server, a video class capture and playback system, and must have the appropriate legal agreements in place to support collaboration. In addition, both institutions and their personnel must have the commitment to work through the many unforeseen issues that undoubtedly occur.

7. CONCLUSION
At DePaul University’s School of CDM we have developed a pedagogical infrastructure that has allowed us to partner with global institutions to teach a number of GDSD courses. The challenges faced by the students in their classes, effectively mirror the challenges faced by GDSD projects in industry. We hope that the experience gained in our classes will allow our students to effectively up manage or lead groups that use remote development. The lessons learned in this class can apply to outsourcing, distributed development or in-sourcing.

8. ACKNOWLEDGMENTS
We wish to thank the administration of DePaul and DePaul College of CDM, in particular Deans David Miller and Lucia Dettori for supporting the development of the GDSD course, and Esther Quintero Associate Director of Global Initiatives who facilitated the interaction with the partner institution.

9. REFERENCES


