Instructional Design and Assessment Strategies for Teaching Global Software Development: A Framework

Daniela Damian
Dept of Computer Science
University of Victoria, Canada
1 250 721 7225
danielad@cs.uvic.ca

Allyson Hadwin
Dept of Educational. Psychology. &LS
University of Victoria, Canada
1 250 721 6347
hadwin@uvic.ca

Ban Al-Ani
Dept of Software Engineering
University of Technology, Sydney, Australia
61 2 9514 1848
alani@it.uts.edu.au

ABSTRACT

In the context of increasing pressure to adopt global approaches to software development, the importance of teaching skills for geographically distributed software development (GSD) becomes essential. This paper reports the experience of teaching a course to prepare graduates for software engineering (SE) in global customer-developer teams, and which was taught in three-University collaboration (Canada, Australia and Italy). The course emphasized the learning of requirements management activities in frequent synchronous computer-mediated client-developer relationships and created a GSD environment with significant time zone and language differences. We describe our instructional approach and assessment strategies within a GSD instructional design framework which integrates (a) required GSD skills and strategies for aligning classroom projects with contemporary and authentic GSD conditions, (b) strategies for assessment of learning of GSD skills and (c) examples from our GSD course.

Categories and Subject Descriptors
H.5.3

General Terms
Design, Human Factors

1 INTRODUCTION

Software engineering education must adapt to meet the changing demands of the software engineering industry. As companies turn to outsourcing as a business model, there is a dramatically increasing trend towards distributed software development. High travel costs, the local availability of skilled technical staff as well as possibilities for around the clock development increase the demand for distributed software engineering efforts.

Processes that appear to be significantly hampered by geographical separation of team members include requirements management and design [9]. It is their communication-intensive and iterative nature that causes problems in the larger organizational and project management global contexts. The growing complexity of projects and inter-organizational relationships make the complete-spec approach in global projects infeasible. Failure to achieve a common understanding of features, combined with reduced trust and the inability to resolve conflicts results in budget and schedule overruns and in damaged client-supplier relationships [9]. Consequently, it is imperative that university curricula emphasize activities of requirements engineering for better management of distributed stakeholders' expectations, as well as project management activities such as planning and estimation in very dynamic development environments [12].

Other courses in the area of Distributed Software Engineering have given variable attention to the activities of requirements management as central to teaching students skills of expectation management in distributed stakeholder groups. Course approaches are primarily in the direction of one-time complete-spec process models, without a particular emphasis on learning to develop requirements iteratively and handle changes throughout the development lifecycle (e.g. [18,10]). Students have either performed requirements elicitations at one site by studying existing software packages, or through face to face visits made at one of the project sites. The most related experience in enabling distributed requirements elicitation and negotiation in distributed settings is reported in [2], where students playing the roles of analysts in Brazil determined the requirements for an information system with users at a US university during a 30 day online discussion. From an educational perspective, key questions are (1) How to create the appropriate GSD learning environments and (2) How to asses that learning of GSD skills so that continuous improvement of our teaching effectiveness is possible. While reports of lessons learned in creating GSD learning environments exist (e.g. [10]), methods for GSD learning assessment are still an area for research.

This paper reports experiences in the design, teaching and evaluation of a course intended to prepare our graduates for geographically distributed software engineering. The course was delivered between Jan-May 2005 in the collaboration of three geographically distributed universities. The course was innovative in (a) emphasizing the learning of communication-intensive requirements management activities in frequent synchronous computer-mediated client-developer relationships and (b) creating a GSD environment with significant time zone and language differences. We describe the course design and lessons learned in a wider context of teaching and assessing learning of GSD skills. Based on research in the education, SE and GSD domains, we describe a GSD instructional design framework that captures (a) emergent GSD skills that our graduates need to acquire (Section 2) and of which teaching presents significant challenges in our traditional curricula, (b) our instructional strategies intended to promote the development of these skills (Section 3) and (c) strategies for assessment of GSD learning and examples from our GSD course. We discuss findings from our assessment of learning (Section 4) and conclude in Section 5 by drawing some recommendations for researchers and course designers with similar educational goals.

2 A GSD INSTRUCTIONAL DESIGN FRAMEWORK

Training students for GSD inherits the challenges of teaching SE. For example, expectations management and project risk are inherently difficult skills to teach as they require years to acquire [5]. The challenge in teaching GSD is enabling students to recognize how the problems of remote collaboration and cultural differences make the performance of these activities even more difficult. Here we outline a set of emerging areas of competencies that our curricula needs to emphasize when training students for GSD, in addition to the fundamental skills of a SE ([11]).

International teamwork. The nature of distributed projects entails that SE expertise in architecture, design and process engineering needs to be leveraged in the *extended software engineering organization* comprised of business units from different locations [12]. Teamwork approaches to reaching an understanding of project goals, expectations and constraints are difficult to teach because (a) collaboration, negotiation skills or contract writing across diverse language and cultural groups are not part of the conventional curriculum, (b) replaying the complexity of real stakeholder teams is hard to achieve in educational environments [11] and (c) the majority of our comp. science students are typically introverts as much as our faculty are generally not prepared to teach collaboration skills [8].

Iterative development in remote client-developer relationships. Given the heightened risk of misunderstandings in distributed communication, students need to learn leadership skills in RE activities including communication with remote customers [12]. The teaching of these skills is difficult as working with real customers is challenging when industrial customers are not often willing to sacrifice software quality in student projects [1]. The need to involve remote industrial customers requires additional effort in managing these expectations.

Living with ambiguity/uncertainty in remote teams. Industrial software projects deal with ill-defined problems and a complex set of technical, organizational and economic constraints. Removing inherent ambiguities and uncertainties by applying requirements engineering and risk management techniques becomes challenging when the communication with remote problem owners relies on computer-mediated means that introduce delays and misunderstandings [9]. Teaching students these skills is difficult as most often our SE curricula includes toy- or well-defined problems [11], and studies show that students find it difficult to recognize ambiguity in software specifications [4].

Distributed project management. Additional resource estimation as well as technical and project planning is required for successful coordination of work in distributed heterogenous environments [12]. Teaching planning skills is not easy however given the current methods' reliance on experience from previous similar projects and application of methods of estimation. The absence of real life economic and organizational constraints in educational projects makes it difficult for students to engage in realistic cost-benefit analysis situations [11].

Computer mediated project communication. While computer-mediated communication (CMC) tools have been far from recommended for rich interaction a decade ago, their use in distributed groups is becoming widespread both in organizational as well as educational settings [3,12]. As travel to distributed locations is often not possible, computer-mediated communication remains one of the best tools for remote teams to meet efficiently and frequently. Students' learning of strengths and weaknesses of computer mediated tools for various collaborative purposes is critical [12,15]. It is also desired that students try out these tools in educational environment, where we can exploit opportunities often unavailable in real settings, e.g. remote synchronous shared design and prototyping sessions, as students are carriers of innovation into the business world [11]. The difficulty is in setting up appropriate communication infrastructures at remote locations and to maintain technical support throughout the GSD projects.

Given these challenges, the teaching of software development in distributed environments raises important research questions:

- Is it possible to create a GSD educational environment that overcomes some of these challenges?
- How do we assess the learning of GSD competencies?

These GSD competencies (learning outcomes) are outlined in Table 2. The set of instructional strategies we adopted to enable the teaching of these competencies are described next.

3 OUR INSTRUCTIONAL STRATEGIES

Our overarching instructional approach was to create a GSD environment that involves students in an authentic GSD task. This entailed a constructivist *problem-based learning* (PBL) approach wherein course curriculum involved authentic problems, working in groups to solve problems, problems that initiate free inquiry by students [22]. In our course (full course material at [25]), students worked in an international software team with the inherent GSD characteristics of geographical distance, different cultures and (possibly) languages. In addition to participating in the distributed projects, students at each university were supported with regular face to face instruction.

Developing International Teamwork skills

The course was offered in collaboration between three Universities: University of Victoria (Canada), University of Technology, Sydney (Australia) and University of Bari, (Italy), in Spring 2005. Given the difference in academic calendars at the three universities, a period of seven overlapping school weeks (Mar.7-Apr. 22), was dedicated to cross-University student project work. The projects were structured as outsourcing projects in which work was allocated to a distributed software group in a different organization. The project outcome was a software requirements specification (SRS) as a negotiated software contract between the software group and outsourcing company. Teamwork was critical in completing the software project as the software group had to frequently interact with the clients to understand the required software features.

There was a total of 32 students at all three sites: 12 graduate Canadian students worked with 10 graduate Italian students, and 2 grad and 8 undergrad Australian students. The students were assigned to 6 international project teams, each involving two countries. There were three distinct projects, each with two instances. Project A (A1 and A2 in Table 1) was to design a *Global software development system* to facilitate GSD collaboration. In project B (B1 and B2) the students designed the interface for a "iMedia" software to allow users to purchase movies online, organize and play their movies. Finally, project C (C1 and C2) involved the design of a real estate system.

Developing computer mediated communication skills

A wide range of tools for collaboration were made available in the global teams, both for synchronous and asynchronous interaction. The client-developer interaction was supported by (1) weekly one-hour long scheduled videoconference project meetings, which utilized tools such as videoconferencing (Access Grid [24] and Polycom) and shared application via VNC [14]. The main objective if this activity was to minimize the misunderstandings that typically occur between clients and software engineers. A unique characteristic of this course, students experienced frequent synchronous meetings with the remote clients for building the shared understanding of features through synchronous requirements elicitation, negotiation and prototype demo sessions. The requirements inspection as performed to clarify ambiguous and incomplete requirements in the specification was carried entirely online using IBIS, an Internet based inspection tool [19]. Outside class interaction was supported by Skype for audio conferencing, MSN IM for text chats, and email. Each project created a website, while document management was done in CVS repositories.

Developing Iterative development and Remote client-developer communication skills. Each group of developers worked with a group of clients from another country. Each student was assigned to a single group for the entire duration of the course. Note the distinction between *group* and *team* is made, to refer to the members of a group belong to the same country and the international team respectively. The group allocation to project teams was such that each group belonged to two projects and two different global teams, playing the role of a client for one project and the role of a developer for the other project (with the partner group always located in a different country).

Table 1: Client and developer groups and allocation to international project teams

Country	Group (number of students)	Project A		Project B		Project C	
		A1	A2	B1	B2	C1	C2
Canada	Gr1 (4)	Client (C)					D
	Gr2 (4)		D	C			
	Gr3 (4)				D	C	
Australia	Gr4 (5)	Developer (D)			C		
	Gr5 (5)		C			D	
Italy	Gr6cl (7)						C
	Gr6dev (3)			D			

To enable an iterative cycle in the short 7-week timeframe, the emphasis was placed on the upstream activities of requirements and design. The shared understanding of required software features was developed through a series of scheduled activities of requirements elicitation, analysis, inspection, negotiation, prototype design and evaluation. The success of each software project relied on frequent client-developer communication.

At each step in the frequent client-developer interaction neither the developers nor the clients had full understanding of the problem to be addressed or the ability to fully remove the uncertainties around the possible technical solutions or resources needed for their accomplishment.

Developing skills in Distributed Project Management

Table 2. A framework for teaching and assessing learning of GSD skills

GSD skills (learning outcomes)	Our approach (instructional strategies)	Strategies for assessment of learning (and *evidence discussed in this paper*: first 2 rows)
International teamwork i.e. work in distributed teams to solve large problems *and* collaborative development of shared understanding of project goals and constraints	Cross-university project teams No single group from one country could solve the problem alone	Degree of projects completion & characteristics of project outcome (s/w spec.); *see Table 3* Students' evaluation of shared understanding: was it achieved, how they conceptualized it, challenges in achieving it; *see Table 4* Degree of learning community in local and distributed teams; *see Section 4*
Computer mediated communication i.e. use of available tools for remote collaboration and evaluation with respect to their strengths and weaknesses	The remote groups use a wide range of tools both for synchronous and asynchronous project interaction	Students' perceived effectiveness of tools for project collaborative tasks and in particular achieving shared understanding in the remote client-developer relationship; *see Tables 5 and 6*
Iterative development in remote client-developer relationships i.e. management of client expectations, developers working closely with market researchers and other project stakeholders throughout the project lifecycle	Student plays dual role, client and developer for two different projects. Each project has remote client-developer structure Project's scope is defined through an iterative process reliant on continuous client-dev. comm.	Students' perceptions of working in a particular role and with a distributed group, as well as assessment of their remote group's performance Trajectory (trend) of issues identified by clients or developers at each project milestone: post elicitation, inspection, negotiation and validation of requirements.
Living with ambiguity/uncertainty in remote teams i.e. tackling ill-defined software problems, when communication with problem owners is computer-mediated	The client group initially defines the problem, followed by ambiguities in specifications and uncertainties about technical solutions being discussed during requirements elicitation, inspection, negotiation and validation.	Tracking of those issues that refer to ambiguous requirements and its trend throughout the iterative development process; number of ambiguous issues resolved at each project milestone
Distributed project management i.e. resource estimation and planning in distributed heterogenous working environments	Self managed teams, team coordinates the work on required project deliverables Each project team negotiates project scope based on estimation of problem understanding and available resources	Students' perceived effectiveness of coordination strategies Number of issues resolved at each client meeting and perceived effectiveness of the negotiations given the project characteristics and CMC tools

Each team self-managed its communication between clients and developers and the coordination of its deliverables on a weekly basis. Each weekly synchronous session had to be planned in advance and appropriate issues brought to the team's agenda for discussion and follow up. Estimating the required resources based on clients' problem description was critical in developers' ability to design a system that fit their current expertise and time in the course. A negotiation session was held mid-development in order to agree the project scope in cooperation with the clients.

4 FINDINGS

Is it possible to create a GSD educational environment that overcomes some of these challenges? Table 2 shows the instructional strategies we used to address those challenges. We posit that the true test of this question is the resulting challenges encountered, experiences accrued and competencies gained. As a result we focus on our second research question "How do we assess the learning of GSD competencies?" Table 2 describes specific strategies for assessing that the adopted instructional strategies were effective. Due to space limitations here we describe in detail our findings with respect to assessment of learning of the first two skills as in Table 2. The specific research questions that we pursued are:

(1) What were the students' understandings of aspects of working in an international team such as achieving shared understanding, and under what conditions was this achieved? and

(2) What were the students' understandings of using CMC tools to mediate distributed client-developer communication and which elements of the software process or team interaction shaped this understanding?

Learning to design software in international teams

In order to assess students' learning of skills of international teamwork, we observed whether they achieved the project goal (i.e. completed the project assigned), and asked students to reflect on the achievement of shared understanding. We also examined whether students reported learning about aspects of cultural diversity in international teams. Findings regarding the length and quality of final project outcomes are listed in Table 3 and demonstrate that students were successful in achieving these goals.

Shared understanding in software design

Shared understanding is what one would expect as a result of successful teamwork. We examined whether students perceived they reached shared understanding of software features and what indicated to them that shared understanding had been met. Each response was condensed to include each key point. This was done by an independent coder who was not an instructor in the course, and not invested in seeing particular kinds of responses (author 2). Key points were organized into themes for clients and developers. Four common themes emerged. Clients and developers both discussed the following themes in their written comments: (a) whether they felt they had achieved understanding (answer yes or no usually), (b) How they knew they had achieved understanding, (c) How they had achieved shared understanding, (d) Why there were problems achieving shared understanding. For two themes (how they knew, and how shared understanding was achieved) we have included responses that were repeated multiple times across individual respondents. For the challenge theme, we have included all statements about challenge and problems. Clients and developers overwhelming indicated that shared understanding had been achieved (23/26 of client responses; 18/20 of developer responses). Table 4 summarizes the things that indicated to students that shared understanding was reached, as well as problems in reaching it from the clients and developers' perspective respectively.

Table 3: Summary of project outcomes

Projects	A1	A2	B1	B2	C1	C2
Assessment of SRS	100%	80%	NA*	83%	96%	87%
Length of SRS (in pages)	81	19	14	41	72	29
Function Points count	153	133	142	153	171	154

*The Italian SRS was not marked since students volunteered in the course; Assessment of SRS was performed by TAs not involved in this research

Sense of connectedness and learning community

Students responded to a classroom community scale adapted from [21]. The classroom community scale measures students' overall sense of community in a course. It is comprised of two subscales: (a) feelings of connectedness, and (b) feelings regarding the use of interaction within the community to construct understanding and satisfy learning goals. Students responded to the questionnaire with respect to their local community groups, and their distributed community project teams separately. A higher the score indicates a higher the sense of community.

Students reported high perceptions of a classroom community (Scores more than 50 indicate positive perceptions of community) on both the learning and connectedness subscales for both local groups (M=76.09, s=13.89) and distributed teams (M=64.06, s=13.83). However, there were statistically significant differences between reports of overall community (*t(31)*=6.357, *p<.001, d=1.01)* community connectedness (*t*(31)=5.34*, p*<.001, *d=.80)* and learning community (*t*(31)=5.71*, p*<.001, *d*=1.06) for local groups compared to distributed teams.. Not surprisingly, students reported stronger perceptions of classroom community in their local groups compared to their distributed teams.

In their discussions about cultural diversity, students described their notions of culture as broadening to include things beyond the cross cultural differences they anticipated. Students recognized cultural differences and similarities within co-located teams not just remote teams. Students also described the importance of professional culture, school culture, undergraduate and graduate

Table 4: Students' evaluation of shared understanding and the problems encountered

How they knew	How shared understanding was achieved		Why there were problems in reaching shared understanding	
	Client responses	Developer responses		
• Prototype demo was successful • SRS were fulfilled • no major unresolved issue	• prototyping process • requirements engineering process • through communication tools • Assigned schedule for requirements engineering imposed by teachers	• requirements engineering (especially after negotiation) • continually revised understandings using communication tools • IBIS	• developers had no background in GSD • developers misunderstood a key requirement • we communicated our intent and features but not WHY they were important • some communication difficulties	• sending wrong drafts to remote group • client not involved enough, didn't really care to discuss or resolve issues • client too accommodating (probably not realistic) "whatever is best for you" • not enough communication • not enough time together for first release of [spec.], needed a second release and [issue] discovery cycle • demanding clients…wanted things not in the SRS

culture on the development of shared goals and trust in this course project. For example, students were confronted with the challenges of having different academic goal orientations at the graduate level (to learn the GSD process and tools) than students at the undergraduate level (to get a good grade). For a couple of students this was seen as more challenging than language differences. For others, sharing "student culture" provided a common place for overcoming other cultural differences.

Learning about CMC tools in GSD

In order to assess learning with the CMC tools in the context of mediating the interaction of global teams, students were asked to rate the *effectiveness* of tools on a nominal scale ('detrimental to the task', 'not very valuable in supporting the task', 'somewhat valuable in supporting the task', and 'essential in supporting the task'). We calculated the frequency of responses indicating that a tool was "essential" in supporting (a) specific activities for achieving shared understanding (CMR=communication of missing requirements, MSR=mapping software requirements to user needs, RA=removing ambiguity in software requirements, RI=resolving inconsistencies in the SRS, and RM=resolving misunderstandings in user needs or software requirements), and (b) particular group processes involved in remote communications (GI=generation of ideas, PT=planning, PS=problem solving, DM=decision making, CRC=cognitive conflict resolution and ACR=affective conflict resolution) [20]. Students also responded to an open ended question asking them to comment on the usefulness and effectiveness of computer-mediated tools for group communication. Data is summarized in Table 5 and 6.

In Table 5 (showing responses of total 32 students), we highlighted the tools with highest three scores in each row (if greater or equal to 10), to give some indication of tool ranking for certain tasks. Face-to-face medium (F2F) was also included for a comprehensive list of communication media. Videoconferencing (VC) was reported as most essential (consistently higher than F2F) in supporting both the shared understanding and the group processes, with the highest scores for removing ambiguities, communicating missing requirements and decision making (confirming claims of 'social cues" theories [23] that the richer the communication medium, the more it is perceived as appropriate to supporting interactive group tasks). Other tools that appear with an interesting trend are email (only ranked third for most tasks, not surprisingly found as most essential for planning tasks [23]). IBIS was rated by clients as next useful tool for removing of ambiguities after VC and F2F, and equal in usefulness to F2F for removing requirements inconsistencies and misunderstandings; from the developers' perspective, IBIS was (almost) equal in usefulness to F2F in removing inconsistencies, next to VC.

The qualitative responses in Table 6 provide additional information on students' understanding of tool usage for the different project activities and group processes. While in general, CMC tools were effective for (a) keeping record of decisions and discussions and (b) communicating and interacting remotely, this data indicates that students perceived each tool to be useful for different things.

Table 5: Students' tool ranking based on usefulness during group processes in general or shared understanding

Processes/Tools		VC	F2F	Email	IBIS	Skype	VNC
Shared understanding (*client* responses)	CMR	**20**	**16**	**12**	8	4	4
	MSR	**16**	**15**	9	8	3	8
	RA	**20**	**19**	10	**14**	5	5
	RI	**17**	**12**	8	**12**	2	5
	RM	**17**	**14**	11	**13**	4	5
Shared understanding (*developer* responses)	CMR	**17**	**12**	10	9	4	4
	MSR	**11**	**12**	6	6	3	3
	RA	**16**	**14**	6	9	3	4
	RI	**14**	**11**	6	**10**	3	4
	RM	**16**	**12**	6	6	4	4
Group processes	GI	**18**	10	8	3	5	3
	PT	**15**	8	**21**	1	3	3
	PS	**19**	9	**11**	8	7	3
	DM	**22**	**10**	**10**	5	5	2
	CCR	**16**	9	**11**	7	5	2
	ACF	2	2	0	0	1	1

Self-reported tool *frequency* data for each group process (actual numbers not included due to space constraints), indicates that email was by far the most used tool for: planning, decision making and problem solving; followed by F2F which was almost equally used for all tasks, then VC for: decision-making, problem solving, planning and affective conflict resolution; and IBIS for: problem solving, decision making and generation of ideas.

5 CONCLUSIONS

In this paper we presented a framework of strategies for teaching and assessment of learning of GSD skills as exemplified in a three-University collaboration course. Due to space limitations we highlighted findings with respect to two dimensions in the framework: the learning of skills of international teamwork and use of CMC tools for remote communication, demonstrating that these learning outcomes had been met.

The students' specific comments about shared understanding demonstrate that they learned to recognize the presence or lack of shared understanding between clients and developers, as well as particular characteristics of the software development process and GSD environment that affected or contributed to it. Differences in responses from the students playing the roles of clients versus developers increases our confidence in the authenticity of students' experience in a particular role despite the seemingly simulated scenario [6]. The direct quotes below confirm this:

Table 6: Qualitative responses from students' evaluation of CMC tools with respect to project activities

Tools	Videoconferencing (VC)	Email	IBIS	Skype	VNC
Positive qualitative responses	• best synchronous tool • effective for use of time and meeting goal • seeing expressions of remote team, explain, and wait for people to think • required more at the beginning • reasonable tool for participation activities (brainstorming, planning) • personal	• best asynchronous tool because it is universal/ubiquitous • flexible • can attach all file formats • good for planning and scheduling • good when there is difficulty • send things in different time zones	• finding and recording problems in RS • allowed various ideas to be expressed • quick resolution of issues • little conflict during discrimination	• best for VoIP • good enough for synchronous comm. • re-creating face-to-face experience	• prototype demonstrations • important for coming to shared understandings

Client: "I feel that with a language barrier it is more important to rely on prototypes than to consider a requirements specification to be a fixed contract"

Client: "Shared understanding was reached. The communications methods, when combined, gave an effective medium in which to continually revise the meeting of requirements. The prototyping sessions were useful in that they allowed viewing of what each team envisaged the final application to look like, allowing best clearing up of and requirements mapping discrepancies."

Developer: "Yes - our clients were very demanding on some issues that we were reluctant to include in the SRS for them. I can see that most of these issues were valid, and that we understand their needs to a much greater extent now than we did at the start."

Given the difficulties of teaching teamwork skills, we believe this is a significant finding that demonstrates some validity in our strategies for learning to work in international teams.

Similarly, recent research in CMC [16] suggests that with current advances in technologies tools usage and their effects on group processes need to be understood in the context of their use rather than simply seeking a general tool-task classification. Our findings indicate that students were able to distinguish the affording characteristics of each tool and judge their appropriateness to particular group or task processes. Rather than one perfect tool, tools were found useful for different purposes. When tool effectiveness data is analyzed together with the frequency information, interesting trends appear: students were able to identify VC as more effective for most group interactive tasks (especially decision making) despite email being most frequently used for these tasks: although VC was "effective for use of time and meeting end goal", email was more frequently used because it was "flexible", "universal" and "ubiquitous." Email was found most essential for planning tasks though. Since this is the only GSD course we are aware of using this innovative approach to enabling synchronous client-developer meetings, we believe that the students' overwhelming appreciation for videoconferencing is a significant result and warrants future research.

Recommendations for research and curricula designers

First, our findings suggest that different group tasks and processes are supported by specific types of communication and production tools. Since our study focused on a limited set of these group processes (e.g., generation of ideas, decision making) and shared understanding tasks (e.g. communicating missing requirements, resolving misunderstandings), we suggest that future work may examine how other global distributed software development tasks and processes are best supported. For example, qualitative findings suggest that particular processes and activities for achieving shared understanding are supported by specific selections of tools. This warrants more thorough examination as we have a very limited understanding of which group processes predominate in activities of achieving shared understanding. Second, introducing a perception of community and trust scale was revealing in this study. We suggest that research and instruction may benefit from administering this scale early and late in the process so instructors and students might consider the factors in their interaction and collaboration that influence perceptions of community.

Acknowledgments

We thank all students, in particular the Italian students who volunteered in this course for the experience of global software projects, and to Dr. Filippo Lanubile who guided them interaction. Thanks to Tish Scott for data management and coding.

References

1. Alzamil, Z. Towards an effective software engineering course project, *Proc. 27th Int'l Conf. on Soft. Eng*., 2005, 631-632
2. Audy, J., Evaristo, R. and Watson-Manheim, M.B., Distributed analysis: the last frontier?, in *Proc. of HICSS' 37,* 2004
3. Bernard, R. M., & Lundgren-Cayrol, K. (2001). Computer Conferencing: An Environment for Collaborative Project-Based Learning in Distance Education. *E. Res. & Evaluation, 7*(2/3).
4. Blaha, K., *et al*. Do students recognize ambiguity in software design? A multi-national, multi-institutional report", *Proc. 27th Int'l Conf. on Soft. Eng*., 2005, 615-616
5. Boehm, B. and Port, D. Educating Software Engineering students to manage risk, *Int'l Conf. on Soft. Eng*., 2001, 591-600.
6. Carver, J., *et al*. Issues using students in empirical studies in software engineering education. Proc. METRICS 2003, 239-49
7. Cognition and technology Group at Vanderbilt. Anchored instruction and its relationship to situated cognition. *Educational Researcher, 19,* 1990, 2-10.
8. Cushing, J., Cunningham, K., and Freeman, G., Towards best practices in software teamwork, *Journal of Computing Sciences in Colleges, Volume 19* (2), 72 – 81
9. Damian, D. and Zowghi, D. Requirements Engineering challenges in multi-site software development organizations**,** *Requirements Engineering Journal, 8, 149-160, 2003*
10. Favela, J. and Peña-Mora,F. An Experience in Collaborative Software Engineering Education" *IEEE Software*, 2001, 47-53.
11. Ghezzi, C. and Mandrioli, D. The challenges of SE education, *Proc. 27th Int'l Conf. on Soft. Eng*., 2005, 637-638
12. Hawthorne, M. and Perry, D. Software engineering education in the era of outsourcing, distributed development, and open source software: challenges and opportunities, *Proc. 27th Int'l Conf. on Soft. Eng*., 2005, 643 - 64.
13. Herbsleb and Moitra: Introduction to The Global View, special issue on GSD, *IEEE Software*, May 2001
14. http://www.realvnc.com
15. Hung, D., & Chen, D.-T. (2003). A Proposed Framework for the Design of a CMC Learning Environment: Facilitating the Emergence of Authenticity. *Ed. Media In'l, 40*(1/2), 7
16. Hine, C. *Virtual Ethnography*, 2000, Sage
17. IEEE SRS Std, Recommended practice for Software Requirements Specifications
18. Johnston, L., et al, Requirements Analysis in Distributed Software Engineering Education: An Experience Report, *Proc. of 6th Australian Workshop on Requirements Engineering*, 2001.
19. Lanubile, F. T. Mallardo, and F. Calefato. Tool Support for Geographically Dispersed Inspection Teams. *Software Process: Improvement and Practice, 8(4)*: 217-231, 2003.
20. Mcgrath, J.E. *Groups: Interaction and performance*, Prentice Hall, 1984
21. Rovai, A. P. Development of an instrument to measure classroom community. *Internet and Higher Education, 5,* 2002
22. Savery, J. R. & Duffy, T. M. (1995). Problem-based learning: An instructional model and its constructivist framework. *Educational Technology, 35,* 31-38.
23. Short, J. *et al*, *The social psychology of telecommunications*, 1976, Wiley
24. www.accessgrid.org
25. http://segal.cs.uvic.ca/csc576b