East Meets West: Global Software Engineering Course in Japan and Germany

Daniel Moritz Marutschke
College of Information Science and Engineering
Ritsumeikan University
Kusatsu, Shiga, Japan
moritz@fc.ritsumei.ac.jp

Victor V. Kryssanov
College of Information Science and Engineering
Ritsumeikan University
Kusatsu, Shiga, Japan
kvvictor@is.ritsumei.ac.jp

Patricia Brockmann
Computer Science Department
Technical University Nuremberg Georg Simon Ohm
Nuremberg, Germany
patricia.brockmann@th-nuernberg.de

ABSTRACT

Global software engineering poses unique challenges to distributed teams and tasks. Engineering education should reflect these real-world scenarios as closely as possible. Due to budgetary constraints and the complexity of conducting global software engineering education, students have limited opportunities to gain international experience. A global software engineering class conducted by the Ritsumeikan University in Japan and the Technical University of Nuremberg in Germany is described. A problem-based learning approach provided students experience working in international software development teams.

CCS CONCEPTS

• **Social and professional topics → Software engineering education**; • **Software and its engineering → Collaboration in software development**; *Software creation and management*; *Software design techniques*;

KEYWORDS

ACM proceedings, global, software, engineering, education, experience, intercultural

ACM Reference Format:
Daniel Moritz Marutschke, Victor V. Kryssanov, and Patricia Brockmann. 2018. East Meets West: Global Software Engineering Course in Japan and Germany. In *ICGSE '18: ICGSE '18: 13th IEEE/ACM International Conference on Global Software Engineering , May 27–29, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3196369.3196390

1 INTRODUCTION

Professional interactions are becoming more international, global software engineering posing unique challenges to distributed teams and tasks. Due to budgetary constraints and the complexity of conducting global software engineering education, students have limited access to such classes. Studies show that engineering education needs experience and problem-based learning approaches [1–4, 6, 9, 18, 19].

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
ICGSE '18, May 27–29, 2018, Gothenburg, Sweden
© 2018 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-5717-3/18/05.
https://doi.org/10.1145/3196369.3196390

This provides an important practical knowledge for future engineers. The learning environment should be as close to the real-world scenario as possible, in order to expose the students to all of the CDIO stages (Conceive-Design-Implement-Operate) of a modern product lifecycle [4–6].

The paper is written from the perspective of the instructors. We will describe our experiences from both German and Japanese sides. First, in section 2, we give an overview of the goals set by us, with subsections regarding learning goals and project organization. Following section 3, we provide our insights from questionnaire findings and our personal observations detailing both the Japanese and the German side, and overall discussion respectively. Lastly, we give concluding remarks in section 4.

2 PROJECT OVERVIEW

This section describes our joint course's objectives and its structure to obtain those goals. Point of views are given by the instructors of both German and Japanese universities, personal insight, considerations on how to address problems and to highlight important features.

2.1 Learning Goals

The goal of this project is to help students learn the skills necessary for their future roles as global software engineers. [16, 17]

(1) Conceptual: Understand key problems in distributed software system development
(2) Tools: Tools for distributed collaboration, such as cloud platforms, video conferencing software, agile tools, such as Jira
(3) Technical: Universally understood concepts, such as UML, good programming practices, modern practices of software engineering
(4) Organizational: Management methods for distributed project groups, distributed agile methods
(5) Intercultural: Communication with project members from different countries
(6) Ethics: Team communication, information exchange, respect

The skills above enable the students to model, develop, and document a modest-sized software product while working in an international virtual team. They should be able to apply fundamental computer science concepts and modern IT technologies and tools to design and implement software systems.

2.2 Project Organization

This paper gives an insight into a global software engineering class on the master's degree level conducted by the Technical University Nuremberg Georg Simon Ohm and the Ritsumeikan University. Two teams were geographically distributed in Germany and Japan. Seven students in total (German and Columbian) participated at the Technical University Nuremberg, seven students in total (Chinese, Korean, Vietnamese, South-African) at Ritsumeikan University.

Teams were intentionally designed to be heterogeneous. Each team was made up of half of its members from Japan (who are mostly non-native English speakers from mainly Asian countries), the other half from Germany (all non-native speakers of English). This requires considerably more international communication than if each team is co-located at the same site [15]. On the Japanese side, we let the students form their own groups with two important conditions: 1) friends should not be in the same group and 2) nationalities must be evenly distributed.

Both teams had the chance to meet with a real client who had the need for a customer loyalty system operating an Irish pub in Osaka, Japan. The German as well as the Japanese side had instructors with extensive experience in teaching global software engineering[1] [13, 14]. In the past, the course in Japan had collaborations with other Japanese universities, Switzerland, and Russia. Over the years, the class was composed of students from different countries, always with a majority from outside of Japan and mostly—but not exclusively—south eastern countries. As in most international teams, English was used as a common-ground language.

The one semester structure of the course was as follows: Most classes were introduced by a short lecture in Japan, joined by the German group via Skype. The rest of the time was used for team discussions and feedback with the instructors or with the client. Classes were conducted over 15 weeks (two semester-hours in Japan, four in Germany), with 12 overlapping weeks (joined by both sides). The first half of the semester was dedicated to the design phase, the second half of the semester was for prototyping.

3 RESULTS AND DISCUSSION

This section describes the experiences from both the German and Japanese viewpoint, insights gained from the project and discussion of the outcome.

3.1 Experiences from the Point of View of the Japanese Team

We purposefully structured the course to follow the CDIO stages—Conceive, Design, Implement, Operate. This is to help the students finish with either a working prototype or a concept demonstration. To enforce the real-world implications, we dedicate one class to have either an industry specialist give a speech and Q&A session relevant to the semester topic or, as in this case, to interact directly with a real client.

[1]The course on the Japanese side is well established and has evolved over eight years of undergraduate software engineering courses and four years of advanced topics in global software engineering, and eight years on the German side (see published papers). The syllabi are accessible via http://www.ritsumei.ac.jp/acd/ac/kyomu/gaku/onlinesyllabus.htm and https://www.th-nuernberg.de/fakultaeten/in/studium/masterstudiengang-wirtschaftsinformatik/ respectively.

Guided by these CDIO principals, the semester is divided into two parts, the design phase and the prototyping phase. We cover the following topics including: software lifecycle and its models; quality management, process improvement techniques in virtual teams and distributed projects; modern practices and future trends in software development; socio-technical systems, outsourcing and global software development; advanced techniques of requirement elicitation; software project management, modern approaches to management, risks and risk management in software development projects; advanced techniques of software development, i.e., software reuse, reference architecture, open source software; software testing and validation, modern approaches to software testing and certification; software product documentation, software documenting tools, Unified Modeling Language (UML).

Our observation from this particular and also previous classes is that students who have difficulties in communicating with other team members and those who, by some reason, were not ready to properly perform their duties always dropped from the class in the first two or three weeks. At the same time, the remaining students performed so far at least up to our expectations.

It is extremely important that the instructors clearly explain, what the required and expected results are for the end of the course, but also an estimated amount of time, that each student would spent on a weekly basis. This is explained in the first or second week of the course.

Reliably one of the main obstacles is the communication in the class' common language (English). This holds true for both the local teams as well as the virtual teams. We mention this to the students several times during the first orientation classes, yet at the end of the semester, two students still reported some hinderance related to the language barrier and felt better prepared for future projects.

Feedback regarding the German team members was positive throughout. Main difficulties arose within the Japan-based teams, which might relate to the German students having more lecture time allocated to the Global Software Engineering course. Another strain on the teams posed the heterogeneity of the Japan-based teams, where the German side was mostly homogeneous in terms of common language (German in contrast to four different ones on the Japanese side, including one native English speaker). We regularly talk to the students related to their team dynamics and encourage mastering heterogeneous team leading.

The two teams' projects diverged in terms of design and implementation after their first requirements elicitation. Team A had the following problem statement and objectives:

(1) **Problem Statement:** The Irish pub does not have a means to facilitate their customer loyalty support program.
(2) **Objective:** Design a customer loyalty support solution to fulfill the Irish pub's program requirements.

Team B stated the following:

(1) **Irish Pub:** The Irish pub is located in Osaka. Customers are about half Japanese, half foreigners.
(2) **Two Branches:** There two branches in Osaka. Also there might be more branches in the future.
(3) **Need of Customer Loyalty System:** The owner needs to build a customer loyalty system, as to encourage customers to come to the pub.

The examples indicated early on, that Team A was looking into a future-oriented approach, whereas Team B was following the reliability of proven technology. Team A presented a working prototype programmed in Python based on the Microsoft Azure's platform and used biometrics (face recognition) to identify customers without additional devices or inputs. Team B demonstrated a simple and reliable (and low cost) barcode-based system running on smartphones.

3.1.1 Difficulties. In the design phase of the project, communication with the client and understanding her wishes and needs proved to be a significant difficulty. Although we assisted the students, the main obstacle was on the one hand explaining and talking to the client on a non-technical plane and on the other hand translating (understanding) the client's propositions into the technical domain.

Students also underestimated on multiple occasions the system's obtrusiveness on the client side, having too many steps involved, taking too long to verify, distracting the bartender, or having unwanted technical implementations such as connecting the system with the cash register. We had to intervene multiple times to point out to students when the number of steps exceeding the allowed or desired number.

The implementation phase showed two main problems. One was the teams being stuck in a "status-quo" of technology, i.e., their mindset needed adjustment to explore more possibilities. Another one was the drifting into marketing strategies for the client rather than focussing on technical aspects of the project. The latter problem reappeared in a different form in the project's final presentation, where an idea to include gamification went too far into business management.

One aspect that was discussed but not followed up at the final project presentation was security, e.g., a user's malicious intent to crack a system's code or hack the client's database.

We had several general difficulties during the 12 weeks, including one network outage (reconnecting after a ten minute break), video-chat quality (partially resolved by sending lecture material previous to each class), and noise pollution while both teams were discussing with their team-counterpart in Germany (resolved by allocating two different rooms).

3.1.2 Future Propositions. Although we try to minimize problems involving the structure of the course, i.e., technical difficulties, scheduling, and team discrepancies, we find some difficulties worth going through.

This particular class, with partners from Germany did not bring any new experience in terms of teaching.

As is the nature of problem based courses, many aspects have to be worked through by the students and experienced to be valuable for their future development. We go into pitfalls of previous projects with the students in the beginning of the project. However many of the same issues present themselves during the semester, which we found is a good way for the students to connect the theoretical lecture with actually facing a problem.

3.2 Experiences from the Point of View of the German Team

As the instructor at the Technical University of Nuremberg, my role was to organize, coach and assign grades for the German students. The preparation phase for this course started six months in advance. In May of 2017, one of the instructors from Japan (Moritz Marutschke) spent one week visiting our university in Nuremberg. In my opinion, meeting one another personally before starting the cooperation project greatly increased rapport and helped to establish trust.

Previous global software engineering courses conducted in cooperation with other universities in Mexico and Mongolia [13, 14] tried to limit the complexity in communication by allowing students to form homogenous sub-teams in each country.

The course described in this paper, however, was much more ambitious. We as instructors assigned the students to work in heterogenous, cross-site teams.

Instead of traditional, instructor-based lectures, I required my master's degree students to self-organize and to work independently. These qualities reflect a high regard for individualism in German society, as opposed to collectivism in Japanese society [7].

During the initial orientation lecture, I introduced my students to distributed software engineering methods. Most of them were already familiar with agile methods, such as Scrum. When the topic of intercultural project groups came up, it became apparent that few of them actually had any experience working together with team members from other countries. Some students were somewhat apprehensive about working remotely with a project group they had never met. German students tend to value known, clear situations and can feel quite uncomfortable in ambiguous, uncertain environments [7]. The students initially focused on how to handle the geographic and time zone differences, rather than potential difficulties in intercultural communication. Many assumed that students in Japan would work and think in a manner similar to students in Germany. My lecture material on cultural dimensions [7] was not taken as seriously as the technical aspects of software engineering. I think that one explanation may be that students feel more comfortable concentrating on problems which they feel they can assert some control over. The problem of time zone differences can be explicitly minimized by planning a video conference for 8:00 a.m. in Germany and 4 p.m. in Japan. The problem of intercultural differences is too ambiguous and thus more difficult to address.

During the design and prototype phase of the class, I was pleased to see that the students in Germany quickly realized that they had underestimated the cultural differences between Germany and East Asia. As expected, the German students were accustomed to working in a very task-oriented way. Instead of taking time to get to know the remote half of their team in Japan, they plunged right in and focused immediately on technical aspects, such as which cloud platform and which collaboration tools would be used. They quickly started assigning tasks and delegating activities, without checking whether the other students in Japan understood what was expected of them. When they felt that the off-site half of the team had not performed according to plan, they pointed out any discrepancies directly, often to the point of bluntness. This direct style of communication can be viewed as quite impolite in collectivist

East Asian societies, which value group harmony [7]. Although I had talked in length about the tradition of "saving face" during the introductory lectures on intercultural aspects, this was not taken seriously during the stressful phase of the project. Getting a good grade was paramount.

In keeping with the principles of Problem-Based-Learning [12], my role as an instructor prevented me from directly interfering with this independent discovery process. The hypothesis of this teaching method is that students learn best by making their own mistakes. One of the hardest things for me as an instructor was to refrain from giving commands about how they should solve a problem. When I saw them making mistakes, such as failing to invest adequate time in team-building, my impulse was to stop and correct them immediately. I found it quite difficult not to lapse into a lecture that their blunt communication could be viewed as insulting.

At the end of the project, I led the German students in a review and retrospective discussion. The moderation method I used was an adapted version of the 4Ls method, developed for Agile project management by Ellen Gottesdiener [8]. Each student was asked to fill out four sticky notes, one in each color.

(1) **Like** (green): What did you like about this project?
(2) **Lack** (red): What did you miss / What went wrong?
(3) **Long for** (yellow): What would you do differently next time?
(4) **Learn** (blue): What did you learn during this project?

I have found this modified version of the 4Ls method to be quite effective at the end of each global software engineering course. Before the review and retrospective, I had the impression that many students didn't really appreciate what they got out of the course. This moderation method helps my students to structure their thoughts on what they have just experienced. After the review and retrospective, they were better to identify the things they had done right, the mistakes they had made and what they would do differently next time. What I found most valuable about the 4Ls method is that it helped students recognize what they had actually learned during this project.

An application of the 4Ls method from my point of view as the instructor in Germany:

(1) **Like** (green): The semester schedules of both participating universities allowed for a 12 week overlap. This allowed for an adequate amount of time for students to work together on a real-world project. I greatly appreciated the chance to work together on this course with the colleagues from Japan. The communication and coordination with my two colleagues in Japan before, during and after the course made this cooperation very effective. Frequent e-mail contact and video conferences between the instructors made it possible to see how student's on each side were progressing and to solve any problems which arose during the course.
(2) **Lack** (red): The German students found the lectures reviewing basic concepts of software engineering unnecessary, because their bachelor's degree programs were all similar. Could this common time have been used more effectively? Although students were assigned to cross-site groups, they rapidly developed an "Us" (on-site) vs. "Them" (off-site) mentality within their groups. It was easier to blame the other half of the team than to take personal responsibility for the entire team as a unit. Skype proved inadequate as a video conferencing system. During presentations, it was often difficult to understand the speaker without a headset.
(3) **Long for** (yellow): We should discuss whether some of the lecture time could be better spent on video conferencing between the student groups. I plan to explore more possibilities to increase trust between the two halves of cross-site teams. Would explicit exercises to further team-building at the beginning of the course lead to more trust between team members and thus better cooperation? Next time, we need an adequate video conferencing system on the German side.
(4) **Learn** (blue): As an instructor, I have learned that my students learn best by making their own mistakes. Telling them about intercultural differences is not nearly as effective as letting them encounter these differences themselves.

In summary, I observed that the students went through a number of learning phases during this project. Students initially displayed a high level of optimism and confidence in their abilities to organize an international software project. During the main phase of the project, they experienced first-hand the difficulties which can occur when working with team members from very different cultures. After the final review and retrospective, my students were able to reflect on both their positive and negative experiences to recognize what they have learned. As an instructor, I have learned that no amount of lecture material can teach these skills as effectively as the experience gained by making your own mistakes.

Table 1: Comparison of opinions of Japanese and German teams: Most important for global software engineering (*smaller being more important*)

Factor	Japan	Germany
Geographical Distance	4.4	5.0
Time Zone	3.3	4.7
Language Difference	3.6	3.9
Proficiency in shared language	3.1	4.0
Cultural difference	5.3	2.4
Familiarity between team members	3.6	5.1
Trust between team members	2.6	1.8

3.3 Discussion of Results

At the end of the semester, we asked our students to fill out a survey on their opinions about which factors are most important for global software engineering. The results are presented in the Table 1.

From the results of this small survey, we can conclude that students in both Japan and Germany felt that trust between team members was the most important factor in global software development projects. This confirms that results reported by Hussain, et al. [11].

The questionnaire asked students to rank the factors in order of importance, both values for each factor are averaged. Smaller values indicate more importance with a minimum of 1, larger values indicating less importance with a maximum of 7. The setup done with Skype as the video-conferencing system reflects trust

establishment described in a 2016 paper by Hussain and Blincoe [11]. To underline the importance of mutual respect and trust in the teams and process, we spend some time in the beginning of the semester to stress these points.

Both teams regarded the geographical distance as one of the lower factors. Although some of the open comments of the questionnaire said they found it frustrating not being able to "just talk in the same room" with their counterparts abroad, cloud systems, video or audio chat, email, and other communication methods make the physical presence less pertinent.

Language differences were rated medium to medium-low on both sides, with a higher discrepancy for proficiency in shared language, where the German side scored almost one scale-point lower. The biggest variation showed for cultural difference, where the Japanese side put significantly less importance than the German side. We think this might reflect the heterogeneous group structure within an international course on the Japanese side and the homogeneity on the German side. For familiarity between team members, the Japanese side favored medium, where the German side scored about one and a half scale-points lower.

4 CONCLUSIONS

Software engineering education needing a well organized curriculum and problem based education approach, global software engineering is known to add a few layers of complexity to the mix. Combined with findings from previous research and as reflected by questionnaire results, the teams benefit from first-hand experience.

We have found that regular discussion with the instructors proved to be very important. As in software engineering education in general, there are plenty of possibilities to diverge from a project goal. We regularly pointed out the importance of using good programming practices, UML, flowcharts, diagrams, and other universally understood concepts.

Given the multicultural environment and varying undergraduate educational backgrounds, the main task for the instructors on the Japanese side has been to provide counseling on an individual basis to team members who struggle to integrate or work on a team. Another focus of the instructors activity was to enforce the teams strictly following the course schedule and abiding all the deadlines. Given the natural complexity of multicultural communication, the most difficult task for the instructors is to reconcile individual perception of the team members in the beginning of the project, but also to ensure a fair distribution of project work to team members.

We experienced the difficulty of using free video conferencing software, where we needed two instructors on the Japanese side to coordinate the classes with short lectures and time for discussion. Due to technical limitations, the video-conferencing was set up via Skype instead of a more sophisticated IP-based system, which made the video camera location and tracking more difficult (done manually).

Finally, trust between project members was identified as the most important factor for the success of global software projects. We have found that trust between the instructors is vital for the success of collaborative course in global software engineering. We have seen trust exercises and making it a conscious subject matter in the beginning of a GSE course continually held the performance on a high level.

ACKNOWLEDGMENTS

The authors would like to express their thanks to Jutta Eckstein for her constructive suggestions on improving this paper. This work was partially supported by a research grant from the Technical University of Nuremberg, for the research project "DiverSenta".

REFERENCES

[1] A. Cajander T. Clear A. K. Peters, W. Hussain and M. Daniels. 2015. Preparing the Global Software Engineer. In *IEEE In Proceeding of 10th International Conference on Global Software Engineering (ICGSE 2015)*. 61–70.

[2] S. dos Santos A. Rodriges. 2016. A framework for applying problem-based learning to Computing Education. In *In Proceedings of IEEE Frontiers in Education Conference (FIE 2016)*. IEEE Press.

[3] John Findlay Avinda Weerakoon, Nathan Dunbar. 2014. Integrating Multi-Disciplinary Engineering Projects with English on a Study-Abroad Program. In *Proceedings of the 10th International CDIO Conference*. Universitat Politècnica de Catalunya, Barcelona, Spain.

[4] Rick Sellens Lynann Clapham Brian M. Frank, David S. Strong. 2012. Progress with The Professional Spine: A Four-Year Engineering Design and Practice Sequence. In *Proceedings of the 8th International CDIO Conference*. Queensland University of Technology, Brisbane.

[5] Edward F Crawley and Doris R. Brodeur Malmqvist, William A. Lucas. 2011. The CDIO Syllabus v2.0 An Updated Statement of Goals for Engineering Education. In *Proceedings of the 7th International CDIO Conference*. Technical University of Denmark, Copenhagen, 47–83.

[6] Jiali Lin Dazhi Jiang. 2012. Project-Based Learning with Step-Up Method —Take CDIO Abilities Cultivation in Computer Specialty for Example. In *Proceedings of the 8th International CDIO Conference*. Queensland University of Technology.

[7] G. J. Hofstede G. Hofstede and M. Minkov. 2010. *Cultures and Organizations: Software of the Mind*. McGraw-Hill.

[8] E. Gottesdiener. 2010. The 4L's, A Retrospective Technique. (2010). https://www.ebgconsulting.com/blog/the-4ls-a-retrospective-technique/

[9] Arnold Plotkin Grigory Rechistov. 2014. Computer Engineering Educational Projects of Mipt-Intel Laboratory in the Context of CDIO. In *Proceedings of the 10th International CDIO Conference*. Universitat Polit'ecnica de Catalunya.

[10] E.T. Hall and M.R. Hall. 1987. *Hidden Differences: Doing Business with the Japanese*. Anchor Press / Doubleday.

[11] Waqar Hussain and Kelly Blincoe. 2016. Establishing Trust and Relationships through Video Conferencing in Virtual Collaborations: An Experience Report on a Global Software Engineering Course. In *IEEE Press, In Proceedings of Inaugural Workshop on Global Software Engineering Education (GSE-Ed'16)*. 49–54.

[12] Y. Delaney I. Richardson. 2009. Problem Based Learning in the Software Engineering Classroom. In *In Proceedings of 22nd Conference on Software Engineering Education and Training*. IEEE.

[13] A. Kress J. Staufer P. Brockmann J.M. Olivares-Ceja, B. Gutierrez. 2017. Project-Based Learning in an International Classroom to Teach Global Software Engineering. In *In Proceedings of International Conference on Education and New Learning Technologies (EDULEARN17)*. IATED.

[14] P. Brockmann G. Ayurzana M. Ende, R. Lömmermann. 2013. A Virtual, Global Classroom to Teach Global Software Engineering: A Joint Mongolian-German Team-Teaching Project. In *In Proceedings of International Conference on E-Learning and E-Technologies in Education (ICEEE2013)*. IEEE.

[15] C. Laasenius D. Damien J. Sheoran F. Harrison P. Chhabra A. Yussuf V. Isotao M. Paasivara, K. Blincoe. 2015. Learning Global Agile Software Engineering Using Same-Site and Cross-Site Teams. In *In Proceedings of 37th International Conference on Software Engineering (ICSE 15)*, Vol. 2. IEEE Press, 285–294.

[16] Y. Shastri R. Hoda, M. Babar. 2016. Socio-Cultural Challenges in Global Software Engineering Education. *IEEE Press, In IEEE Transactions on Education* Issue 99 (2016).

[17] J. Barr M. Daniels M. Oudshoorn J. Noll. 2017 S. Beecham, T. Clear. 2017. Preparing Tomorrow's Software Engineers for Work in a Global Environment. *IEEE Press, In IEEE Software* 34, 1 (2017), 9–12.

[18] Stefan Schneider, Richard Torkar, and Tony Gorschek. 2013. Solutions in global software engineering: A systematic literature review. *International Journal of Information Management* 33, 1 (Feb. 2013), 119–132.

[19] J. Barr M. Daniels R. McDermott M. Oudshoorn A. Savickaite J. Noll. T. Clear, S. Beecham. 2015. Challenges and Recommendations for the Design and Conduct of Global Software Engineering Courses: A Systematic Review. In *ACM In Proceedings of the 2015 ITiCSE on Working Group Reports (ICTICSE-WGR15)*, Vol. New York. 1 – 39.