Global Software Engineering in a Global Classroom

Paul Schmiedmayer
paul.schmiedmayer@tum.de
Technical University of Munich
Munich, Germany

Robert Chatley
rbc@imperial.ac.uk
Imperial College London
United Kingdom

Jan Philip Bernius
janphilip.bernius@tum.de
Technical University of Munich
Munich, Germany

Stephan Krusche
krusche@in.tum.de
Technical University of Munich
Munich, Germany

Konstantin Chaika
kvchaika@etu.ru
St. Petersburg State Electrotechnical University "LETI"
Saint Petersburg, Russia

Kirill Krinkin
kirill@krinkin.com
St. Petersburg State Electrotechnical University "LETI"
Saint Petersburg, Russia

Bernd Bruegge
bernd.bruegge@tum.de
Technical University of Munich
Munich, Germany

ABSTRACT

Due to globalization, many software projects have become large-scale and distributed tasks that require software engineers to learn and apply techniques for distributed requirements analysis, modeling, development, and deployment. Globally-distributed projects require special skills in communication across different locations and time zones in all stages of the project. There has been advancement in teaching these concepts at universities, but adapting global software engineering in a curriculum is still in infancy.

The main reasons are the effort and coordination required by teachers to set up the project, manage distributed development and enable distributed delivery. It becomes even more difficult when teaching distributed software engineering involving Internet of Things (IoT) applications. The situation has changed with recent advances in continuous deployment and cloud platform services that make globally-distributed projects more feasible, teachable, and learnable, even for short-term projects. However, no experience report in education research describes a truly distributed global setup in continuous software engineering for IoT applications.

This paper describes a ten-day project involving three universities in different countries with 21 students located across the world to substantiate this claim. It provides teachers with recommendations for conducting a global software engineering course in a global setting. Recommendations include access for all students to (remote) hardware, stable network infrastructure in all locations, the use of a central development platform for continuous integration and deployment, and the application of distributed pair deployment.

ICSE-SEET '22, May 21–29, 2022, Pittsburgh, PA, USA

CCS CONCEPTS

• **Applied computing** → **Collaborative learning**; *Distance learning*; • **Social and professional topics** → **Software engineering education**.

KEYWORDS

Teaching, Continuous Deployment, Continuous Software Engineering, Open Source, Internet of Things, Distributed, Remote

ACM Reference Format:
Paul Schmiedmayer, Robert Chatley, Jan Philip Bernius, Stephan Krusche, Konstantin Chaika, Kirill Krinkin, and Bernd Bruegge. 2022. Global Software Engineering in a Global Classroom. In *44th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET '22), May 21–29, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3510456.3514163

1 INTRODUCTION

Globally-distributed projects have become the norm for large software systems in industry [4, 13]. Teams are often spread across offices and countries. Particularly in light of the COVID-19 pandemic, remote work has become common [30]. Advances in digital communications technology continue to make collaboration easier even when workers are not colocated. Remote-first work has continued following the pandemic. As a result, global software engineering is likely to remain long-term.

The software engineering education community recognizes the need to provide instruction that prepares graduates to work in the software industry [7, 40]. In order to teach relevant and practical skills, teachers need to deliver students with experiences that mirror what happens in the "real world" [7, 8, 10]. Nevertheless, creating a course featuring an authentic global software engineering experience is a significant challenge [7, 9, 22, 23, 37]. The university experience has traditionally centered on bringing students together (often internationally) in one place to study and work together. However, the conditions that we want the students to experience while working on a global software engineering project are different.

Instructors are faced with determining how to make students with little or no experience aware of global software engineering challenges and equip them with skills to deal with them. There are two main options. Teachers can simulate a global software project, e.g., in a classroom setting, as reported by Li et al., maintaining the organizational effort relatively low by avoiding the problems of proper distribution [23]. The alternative is to arrange genuine global software engineering projects ranging from distributed requirements engineering to software delivery, including all the organizational challenges of a distributed organization and infrastructure [9].

Recent advances in Software as a Service (SaaS) products have made it possible to set up distributed development infrastructures with minimal overhead and cost in a reasonable amount of time [5, 28]. Combining this with online, collaborative tools supporting continuous software engineering methods for development, delivery, and deployment provide the foundation to organize and execute truly distributed project courses, producing nontrivial software applications in a short time frame [15].

In particular, continuous software engineering methods have become common among companies delivering SaaS, using techniques such as continuous integration, automated testing, and the application of deployment pipelines to ensure that system updating is a rigorous, reliable and repeatable process [5]. They have also appeared in the curriculum of many software engineering courses [17–19, 21]. In fact, continuous software engineering has become an essential practice for software engineers and everyone developing software-enabled products of all types such as smart-home technology, embedded devices, automotive technology, or smart cities [25].

An additional challenge is offering a globally-distributed project that includes software and hardware components. Developing software to control or interact in a global setting with the Internet of Things (IoT) devices in a distributed setting requires students to develop software for and test it on physically distributed hardware.

We believe that these kinds of applications can be taught in a global classroom. Therefore, we offered a ten-day project involving three universities in different countries, Imperial College London (Imperial), the Technical University Munich (TUM), and St. Petersburg Electrotechnical University (LETI). There was a group of 21 students from these institutions, located across the world in five different time zones. The system under development consisted of three subsystems, each with software and hardware components:

(1) Urban infrastructure for autonomous cars
(2) A fleet of drones to support aviation-based transportation
(3) A traffic lights manager based on smart light bulbs

The paper is organized as follows. Section 2 describes related work in the fields of teaching global software engineering, continuous software engineering, IoT applications, Platform as a Service (PaaS) and Software as a Service (SaaS). Section 3 defines our teaching concept, particularly team constitution, creating communities, agile methods, and the infrastructure setup for a global project in an academic setting. Section 4 describes the details of the course instantiation, including teaching the course during the COVID-19 pandemic. Section 5 concludes with a set of recommendations for conducting a global IoT-based software engineering course.

2 RELATED WORK

Globally-distributed software projects and the complexity of coordinating communication global software projects are decades-old research fields [13, 16]. Similarly, teaching global software engineering in a university context is a well documented teaching format incorporating several challenges [4, 6, 9, 12, 22, 23, 26, 34, 37].

Beecham et al. inspire the need for global software engineering education and summarize 19 global software engineering education-related challenges and proposed solutions [4]. Bruegge et al. describe the experience of organizing three courses on distributed software engineering with students from Carnegie Mellon University (CMU) in Pittsburgh, USA, and the Technical University Munich (TUM) for real clients located at a third site; they also express the challenges and advantages of distributed software projects including video communication [6]. Stroulia et al. [37] and Sievi-Korte et al. [34] describe distributed courses across universities, including open-source projects and agile project management [34, 37]. Similarly, Matthes et al. provide insights into teaching global software engineering across five different universities in France, Mexico, Germany (2x), and Chile [26]. Gloor et al. reported on six years of teaching distributed courses, highlighting lessons learned like balanced teams, team commitment, frequent deadlines, clear information flows, and the need for global trust and cultural understanding in distributed teams [12]. Damian et al. highlight the challenges of teaching distributed software engineering between the University of Victoria in Canada and Aalto University in Finland, underlining several challenges, including the difference in time zones [9].

Research by Lescher et al. and Li et al. emphasizes possibilities of using exercises in classroom and seminar settings to simulate global software engineering projects and how students obtain information about global software engineering using classroom exercises showcasing communication overhead, impediments, and delay in global software projects [22, 23]. Anderson and Ramalingam highlight how similar principles can also be applied to other domains such as construction management, providing similar insights like comparative research about global software engineering [2].

Krusche et al. focus on a similar teaching approach to that proposed in this paper, where creativity is fostered by allowing opportunities and mistakes [20]. They base their teaching approach on chaordic learning theory using cha-ord, a mixture between chaos and order [14]. The teaching concept described in this paper also relies on related work about pair programming [42] and ensemble programming (originally called mob programming, a term avoided for its negative connotations) [44], facilitating communication and active knowledge exchange between students and within teams [42, 44].

Several publications are concerned with a subset of topics discussed in the course defined in this paper about continuous software engineering for IoT applications. Kuusinen and Albertsen depict a course that focuses on continuous software engineering, including setting up a CI pipeline and interacting with well-known source control standards such as git [21]. Mäenpää et al., and research by Silvis-Cividjian, draw attention to courses teaching development with IoT devices and related hardware technologies while aiming for self-learning and development-focused approaches [29, 35].

GitHub is used as a platform to teach contributions about open source projects, contribute to open source projects, and potentially create new projects, which is described in research by Tan et al. and Feliciano et. al [11, 39]. The research lists practical benefits such as gaining industry-relevant experience and encouraging student collaboration as well as challenges associated with the platform and working on projects in public [11]. Li proposes to use free PaaS offerings for education projects which is part of the teaching concept described in this paper [24]. It enables students to work on a project with a setup as close as possible to real projects. Research by Stray et al. demonstrates the application of SaaS-based asynchronous communication tools used in the case study in section 4, representing the applicability of these tools to global team collaboration [36]. The course defined in this paper applies different lessons learned and insights from teaching continuous software engineering, IoT applications, and using PaaS or SaaS offerings to create a new teaching format.

Similar to many other courses and industry projects, the course design described in this paper is also affected by the COVID-19 pandemic. Ipek Ozkaya highlights the COVID-19-related challenges and opportunities for software engineering, including the flexibility of hybrid work and the problem of retaining organizational memory [30]. Lessons learned from other courses being reworked due to COVID-19, such as the work by Schmiedmayer et al., explaining how to conduct online sessions, remote supervision, and real-time feedback for students in the setting of an ongoing pandemic [32].

3 TEACHING CONCEPT

Software engineering research comprises a global community with an increasing number of students studying the topic and more and more professionals finding jobs in the technology sector resulting in an increased need for high-quality software engineering education [27]. As universities aiming to provide a high-quality educational experience, we want to provide opportunities for our students to work with other members of this international community, exchange ideas, and complete collaborative projects to develop their engineering skills, possibly resulting in new research opportunities.

Bringing students from different universities together is essential: the idea of joint student schools or summer schools has existed for a long time, and many have been completed with great success [20]. However, the ways of working in the world of software engineering are changing. Our work was initiated during the COVID-19 pandemic, which in many cases pushed companies and universities toward remote work. Due to regulations and travel restrictions, a remote setting was the only possible way for conducting a global course.

Therefore, our aim in developing a new teaching format was to update our notion of summer school and bring students together as part of an international virtual community to work on common themes, share their experience and skills and to contribute to a common goal. Although running an inter-university collaborative course in a virtual setting might be seen as a compromise, we believe that this mirrors the interaction patterns of modern software engineering in many ways. Hence, running a collaborative course in a distributed and virtual style reflects the environment that many software engineers work in today. As a result, such a course helps develop critical skills related to effective communication and collaboration in a virtual setting using digital tools.

3.1 Team Constitution

It is vital to construct teams with diverse members to implement a successful distributed project, particularly with students from different institutions and a uniform spread of demographics and home institutions. Additionally, we observe that it is often natural for students to group with others from the same institution, country, or who are "like them", even in a colocated course. There may also be differences in prior knowledge, skills, or experience with students from different universities based on what they have covered in their home university curriculum.

There is value in creating mixed and balanced teams [41]; for this reason, our recommendation is to preallocate groups rather than allow self-organization when forming teams. At the same time, it is essential for the overall success of the course that the students are engaged and excited about the topic they work on, when preallocating. Therefore, we suggest giving students options regarding the technical focus of the work and allocating teams around these themes, taking the students' preferences into account as much as possible while reserving the right to give second choices where it makes for a better split across teams.

3.2 Creating Community

Gloor et al. noted that one of the keys to teaching a successful global project course is creating global trust [12]. However, this does not always happen organically and is even harder to build when participants have never met in person. Our aim is to allow students to develop the skills to communicate and build trust as part of the learning experience. Thus, we need to create conditions and structures conducive to collaboration and making sense of a new community. We recommend three various types of structured activities to foster collaboration in different ways at different times:

First contact: There is a long tradition of using "ice breakers" – short, fun, collaborative activities – to encourage participants to introduce themselves to one another and to break down the initial social friction of not having worked together before. In a colocated setting, popular icebreakers often involve collaborating to complete a fun challenge together, e.g., the Marshmallow Challenge [38], where teams have to build the tallest free-standing structure in a limited time. In a virtual world, it is helpful to recreate the dynamics of an icebreaker exercise. If any physical items are involved, participants may try and solve the problem on their own, without collaborating, as there is no physical interaction with shared objects. As a result, we recommend an interaction around a shared digital artifact, where one person's actions affect the experience of the others. We endorse keeping such activities short and ensuring they are accessible. Not everyone may have access to the same resources; therefore, activities completed via a web browser are better than those involving dedicated hardware.

Broadband collaboration: Actively working together on a single problem as a pair or group and constantly verbalizing and exchanging ideas is ideal for broadband collaboration. This has

long been realized in software development through pair programming [42] or ensemble programming [44]. Pair and ensemble programming sessions can be conducted effectively online using supporting technology and provide a valuable way for teammates to solve a particular problem together and share knowledge. This is particularly the case when one teammate has a specific skill or competency and can pair program together with a less experienced member to pass on their knowledge. It is especially advantageous when the "junior" member of the pair has control of the screen and keyboard.

Structures for Regular Synchronization: In a virtual setting, it is easy for participants to disengage from the rest of the group, and it is difficult for organizers to notice when this occurs. When we are all together in a room, it is easy to see someone sitting on their own, struggling, or going off in an unuseful direction. Facilitators can pick up these signals organically in a physical space with a relatively small number of people. It is best to introduce more structure in a virtual setting with regular synchronization points to ensure that everyone is on track, knows what they are doing, and has the support they need. As a software engineering community, regular ceremonies associated with practices like Scrum effectively support synchronization.

3.3 Agile Methods

Agile methods are a natural fit in an environment where we want to balance self-organization and freedom to explore a technical area with structures that encourage collaboration, shared learning, and cross-team synchronization toward a common goal. While we do not recommend following any particular agile method dogmatically, we endorse adopting the general principles of lightweight structures and ceremonies that foster collaboration and help keep everyone on track while allowing changes of direction and priorities to make the best use of limited time and resources.

In the context of an intensive course carried out over a small number of consecutive days, we would recommend at least daily synchronization points. Agile methods such as Scrum [33], or XP [3] advocate using short "standup" meetings to report on what advances have been made since the last synchronization, what if anything is blocking further progress, and planning what to do next to have the best possible day. Although in virtual meetings, it may not be natural to physically stand up – traditionally a measure to help to keep such meetings from dragging on too long – all the other parts of a standup meeting can be recreated online. There are also advantages in software projects, such as the ease of sharing a screen to demonstrate new work, which can sometimes be challenging to do seamlessly in a physical meeting.

Agile methods emphasize rapid iteration, focusing on learning and demonstrating results. This fits well with a course that does not extend over a long period and is exploratory, investigating problems in a given research area rather than following a tutorial to teach a tried and tested method.

3.4 Infrastructure Setup

Conducting a distributed course on project-based software engineering requires an infrastructure arrangement to fulfill several requirements.

First, asynchronous and synchronous communication tools are essential for all participants to communicate in a distributed setting. Students and instructors need to collaborate on source code using widely adopted distributed version control tools, e.g., git[1]. Automatic testing and deployment of the source code using continuous integration and continuous deployment provide participants valuable feedback and enables automated workflows for collaborating. Providing a platform to integrate software components into subsystems, run integration tests, and automatically deploy the system is crucial in reducing the manual overhead of conducting such a course. Exploring IoT devices or continuous deployment setups involving multiple hardware components requires participants to access hardware that cannot be distributed to all participants. Accessing these components in a distributed course requires tools to remotely access, control, and observe hardware components. A reliable way to access and control remote hardware is essential if a deployment or code execution fails in the target environment. In addition, the infrastructure needs to provide tools and means to enable participants to share their status in a distributed meeting and encourage the sharing of rapidly changing knowledge. Synchronous meeting setups should also enable pair programming or meeting in smaller groups to discuss problems like exploring failures in the continuous deployment pipelines.

University-specific services are often limited to its students and instructors or require time-consuming registration processes for students from other universities. This is not feasible in a short-duration project course. Therefore, we recommend using PaaS and SaaS solutions that do not require access to enrolment information or local resources at a specific university. PaaS or SaaS solutions such as AWS, Google Cloud, Microsoft Azure, and GitHub offer low entry costs while allowing the design of a scalable architecture and tailoring the courses to varying sizes [28]. These providers often make limited free tiers available, particularly for open-source development projects. Developing the course as a set of open-source projects enables better visibility of the results for all participants, permits reuse across different courses and research projects, and allows the infrastructure setup to take advantage of the added benefits offered by PaaS or SaaS providers. Showcasing advantages and possibilities of open source development to participants educates them about a mindset which encourages the reuse and sharing of the course results.

Using PaaS or SaaS products enables instructors to reuse knowledge about widely used tools as a part of the course setup to provide the students with a superior onboarding experience. Infrastructure access needs to be tested by the instructors before the course starts to ensure students have access and can navigate the platforms. Sharing tasks before the course enables participants to become familiar with the course content, facilitating a shallow learning curve.

4 CASE STUDY: JASS 2021

This section describes the Joined Advanced Student School 2021 (JASS 2021) as a distributed course on project-based software engineering. JASS 2021 shows how the teaching concept and infrastructure setup mentioned above are instantiated in a cross-university course conducted by Imperial, TUM, and LETI. JASS courses have

[1] https://git-scm.com

been conducted for over a decade, initially funded as an industry-university collaboration and transformed into a collaboration between TUM and LETI since 2006. These collaborations were conducted in-person, requiring one group of researchers and students to travel to the respective city of the collaborating university. As a result of the COVID-19 pandemic, we developed the global teaching concept described in section 3, allowing us to add a third university while adhering to the COVID-19 restrictions in March and April 2021.

4.1 Topics and Themes

The core theme of the JASS 2021 instance was Continuous Software Engineering in combination with IoT smart devices. The goal was to provide students with projects that empower them to create a collaborating IoT system developed as an open-source project and deployed to the distributed IoT environments using continuous software engineering. We provided students insights into open-source research projects conducted at the participating universities, such as the Apodini[2] framework to develop evolvable web services, a research project at TUM [31].

We identified three sub-projects within this project scope using different hardware types that can be easily applied in a university course setting. The first type of devices were education-focused Tello drones produced by DJI and Ryzen[3] that provide a software SDK to control the drones using mobile devices to simulate air-based traffic in a smart city. The second type was DuckieBots[4], Raspberry Pi-based model cars initially developed by the Massachusetts Institute of Technology, representing autonomous vehicles in the smart city environment. The third group was smart lights[5], taking the place of traffic lights that can be controlled in a local WiFi network. The project goal was to develop a smart city scenario with the three device types interacting in an intelligent infrastructure setup. The deployment of this scenario should be automated using continuous software engineering, enabling the students to learn about the various device groups, how these groups can interact, and how to set up and maintain a continuous delivery setup for smart IoT devices. Setting up this continuous software engineering pipeline enabled students to work collaboratively together on a project involving hardware distributed across three sites.

After establishing a continuous software engineering workflow, one of the main challenges was demonstrating the interoperability of the IoT-based heterogeneous software subsystems. As often the case in global software projects, these subsystems were heterogeneous in structure, designed by separate sub-teams, and used different operating systems and communication protocols. Thus, the project illustrated the need for combined cross-platform interoperability of these subsystems, forming a smart infrastructure setup. In the case of a smart environment, interoperability refers to the ability of systems, applications, and services to exchange information and work reliably and predictably together [1, 43]. As part of the course, students had the task of addressing two scenarios concerning the interoperability problem.

The first task was defining the interaction between various agents and subsystems including the DuckieBot notifying the traffic lights about its position and updating the autonomous model cars with traffic light states in the smart environment. Drones should examine the traffic situation and communicate this to the other subsystems, and the traffic lights provide interaction points for the drones to control the traffic in the smart environment based on their observations.

The second task was to ensure that continuous code deployment in the IoT environment did not break any running operations in the different subsystems. A mechanism was implemented to notify and ensure that each agent's software was safely updated allowing them to switch to an update process in a safe state.

4.2 Team Distribution

Twenty-one students from three universities participated in JASS 2021. Almost all students partook in computer science-related study programs and already participated in introductory courses in their bachelor's or master's curriculum. Instructors preselected the students from their respective universities to guarantee familiarity with foundational concepts of the course. Due to COVID travel restrictions, we began gathering team preferences from all participants in February 2021 after students With students from different universities and in different time zones were selected to participate in the course. Based on the initial preferences, we distributed the teams at the beginning of March, two weeks before the start of the course. This allowed the participants to get to know each other and become familiar with the course tools, as mentioned in the Infrastructure Setup subsection.

The team distribution was made after imposing several constraints aimed at producing well-balanced and diverse teams. The first criterion was that the students from each university should be spread equally across the different sub-teams to enable cross-university collaboration. We ensured that the self-identified gender distribution was similar in each team. Each group consisted of students already familiar with a topic and students who could learn from their peers. Based on these criteria, we formed three teams containing seven students, with at least two students from each university and one student who self-identified as female. We could distribute 16 students based on their first choices. Three students were allocated their second choice, and only two students were assigned to their last choice Before finalizing the team distribution, we reached out to these students to ask if they were comfortable joining their third choice.

Each team was supervised by at least two doctoral candidates of multiple participating universities responsible for providing input and helping organize the teams' internal task distributions and pair programming arrangements. Professors from all three participating universities oversaw the complete project, delegating tasks to the doctoral candidates and the teams and shepherding the project's general direction.

[2]The Apodini framework can be found on GitHub at https://github.com/Apodini/Apodini

[3]More information about Tello drones can be found at https://www.ryzerobotics.com/tello-edu

[4]Information about the Duckietown project can be found at https://www.duckietown.org

[5]We have been using smart lights manufactured by LIFX as they offer an easy-to-implement UDP-based communication protocol https://lan.developer.lifx.com/docs

4.3 Case Study Setup

The course infrastructure setup began in February when we created a Slack[6] workspace for the course organizers to communicate with and invite students once they were selected for the course. The Slack workspace provided a university-independent and free cloud-hosted solution to communicate asynchronously with all course participants. Synchronized meetings were conducted using a Zoom[7] license from one of the participating universities, allowing the organizers to invite all participants to join virtual meetings.

In addition to synchronous and asynchronous communication using cloud-hosted SaaS solutions, we decided to use GitHub as the central source code and knowledge-sharing platform. GitHub provided us with a free way to host source code and made it easy to invite students and reuse their existing knowledge, e.g., open-source contributions. GitHub wikis, README files, or dedicated GitHub repositories were used to share information with participants and document the setup of the individual components. Issues and pull requests were used to structure the development effort. We used GitHub Actions as an integrated and free CI/CD service which is usually billed per minute if you exceed a specific quota[8]. Developing the components as open-source components or making them public during the course enabled us to use the unlimited free GitHub Actions minutes to build and deploy software to the hardware devices. Each subteam was working in their separate repositories and created a specialized CI/CD workflow allowing them to customize the deployments to the needs of the hardware components.

We followed two approaches concerning the hardware IoT devices. Firstly we sent some IoT hardware such as smart light bulbs and small indoor drones to the students' homes. Therefore, individual students could access a single IoT device at home for testing purposes. We used the postal service or allowed students to pick up and drop off the device per COVID-19 regulations.

In addition to providing a subset of students with single IoT devices to develop and test at home, we also had larger setups at two of the three universities. We prepared a room at each university that included a complete setup of the smart city simulation. The Munich setup included roads, two DuckieBots, places to start and land a Tello drone, and six traffic lights. Two Raspberry Pis controlled traffic lights simulating intersections, permitting communication in the smart city setup. The Saint Petersburg setup included a more sophisticated road map and six DuckieBots. The site featured several cameras mounted to the ceiling; Figure 1 displays the setup from different angles. Each site had a student or instructor present to supervise the hardware and help in case of deployment issues. All participants could observe the setup using live video streams (Figure 2) showing the situation from different angles and Zoom conferences with the student or instructor present at the sites.

4.4 Course Schedule

The JASS 2021 course was a five-day course spread over eight calendar days. The course was scheduled around Easter, which meant a four-day weekend due to public holidays on Friday and Monday in several participating countries with no requirement to be present during these days. We had students joining from UTC+0 in the United Kingdom to UTC+7 in China and Indonesia. Therefore, we created a schedule that allowed for reasonable working hours and active collaboration across different time zones.

The first day of the course started with an introductory presentation by the course instructors, an ice-breaker, and tutorials about drones, smart lights, autonomous driving, as well as web service

Figure 1: Cameras positioned above the model smart town enable participants to observe the situation from different angles. The smart town setup uses DuckieBot project-based roads and markers, that help DuckieBots navigate the road network.

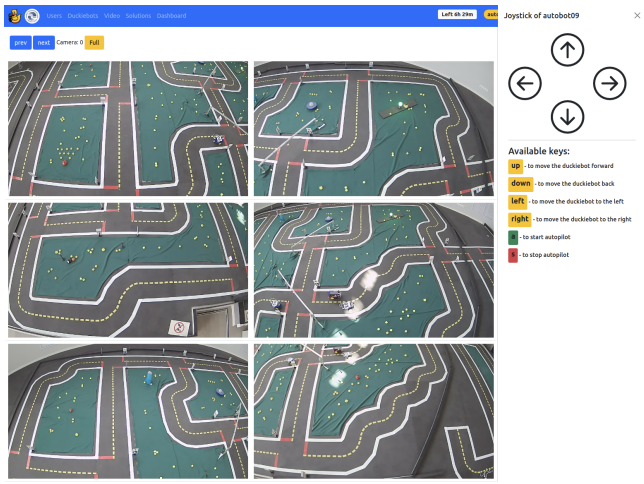

Figure 2: The video streams from the Saint Petersburg laboratory can be observed on a website. The website offers different camera angles to observe all are areas of the smart city environment. Students can manually control a DuckieBot using controls available on the web page.

[6]https://slack.com
[7]https://zoom.us
[8]More information about GitHub Action billing limits can be found at https://docs.github.com/en/actions/learn-github-actions/usage-limits-billing-and-administration

development using Swift and Apodini. The first three lectures were thirty-minute introductory lectures about the topics, followed by three one-hour lectures. The in-depth lecture included code examples and exercises to apply the learned content and use them as starting points for the challenges. The ice-breaker consisted of an interactive online quiz enabling participants to get to know each other despite the physical distance.

The project was conducted as a single sprint aiming to create a first product increment as a shippable project increment at the end of the course. We used agile methodologies to organize the teams' daily schedule and development organization, as discussed in section 3. Every day started with a global meeting involving everyone in a video call for 20 minutes, beginning at 09:00 am UK time. This meeting provided a forum for daily synchronization in terms of status, impediments, and promises. After this meeting, subsystem-teams moved on with another meeting to discuss issues in-depth as part of the team standup meetings. While we did not define a fixed end of the working day, the formal meetings and most discussions in Zoom rooms and Slack channels ended at around 6:00 pm UK time. Nevertheless, several students and instructors continued developing and discussing course-related work in the evening or conducting virtual get-togethers.

We used pair programming and encouraged the paring of students from different universities, taking advantage of the screen sharing and remote control features in Zoom meetings. Moreover, we encouraged students to get to know each other across universities by pairing them up with regular rotations. We extended the pair programming concept to *pair deployment*. In pair deployment, the driver at one site modifies the software and starts a deployment. The navigator at the other site observes the automatic deployment using continuous integration and delivery and is the observer at the other site. In traditional pair programming, the navigator is in the observer position while the driver is typing. The navigator is in the observer position in pair deployment while the driver executes remote commands.

We used pair deployment for testing and for distributed demonstrations. The presentation on the last day of the course featured multiple demonstrations involving pair deployments. Figure 3 shows the pair deployment navigators in a demo of the traffic light subsystem in city A. The navigators use Zoom on a laptop and an external webcam to stream the results of the pair deployment to the remote pair programming drivers in city B.

4.5 Outcome

This setup and the usage of the concepts of continuous software engineering were the key enablers to conduct a global software engineering course involving IoT devices. Altogether the students developed three subsystems using CI/CD pipelines for deployment to the target environments.

The drone team implemented a GitHub Action-based CI/CD pipeline to deploy a service monitoring intersections of the smart city using drones. However, updates were not allowed to be installed while the drone was flying to ensure flight safety. The drone flight is controlled from a Raspberry Pi, which connects to the Tello Drone via WiFi. The drone lifts off, flies above the city, and locates the intersection based on a request. After taking a picture of the traffic

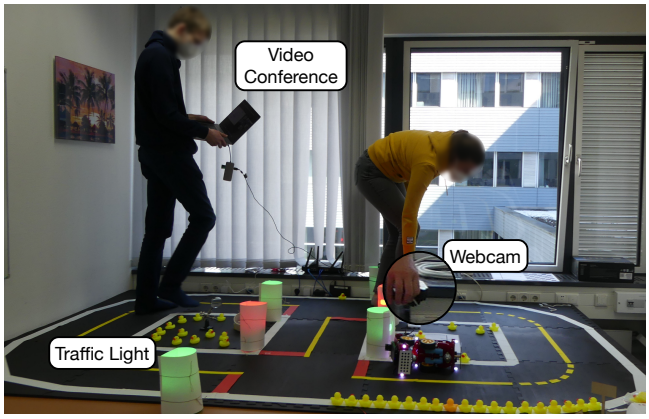

Figure 3: A world view of the traffic light demo involving pair deployment at the Munich-based smart city setup involving LIFX-based traffic lights and DuckieBots.

situation, the drone flies back to the takeoff point and lands. A web service is offered over an HTTP and JSON-based API. Clients can request the picture, and the request is successfully answered with a picture captured by the drone.

The traffic light team built a system to manage intersections in a smart city and implemented different traffic control strategies to manage the traffic flow in a smart city. The system manages the states of traffic lights in a city represented via a digital street map. An HTTP and JSON-based API can be used to select different traffic control strategies. The team developed a continuous deployment process to allow new releases of the traffic light controller, allowing modification and extension of the switching strategies.

The autonomous driving team implemented a continuous deployment process for the DuckieBot software based on ROS. The remote setup, especially regarding Duckietown at LETI, created a necessity for an automated deployment process. Students cannot flash their software onto the target environment without physical access. Additionally, the CI pipeline uses GitHub Actions tests and integrates student code increments pushed into the project's git repository. The CD process deploys new code increments pushed into git using actions aimed at the execution environments at the different universities. Changes to the autonomous-driving software were also managed using GitHub Actions and SSH access. However, starting the cars required a manual command.

5 CONCLUSION & RECOMMENDATIONS

We conducted a one-week global software engineering course using IoT applications in a global and truly distributed setting. The goal was to understand the feasibility of global projects using continuous software engineering and IoT applications in an academic environment and provide recommendations and lessons learned for other instructors. The artifacts of this project, in particular the definitions of the CI/CD pipelines, can be found in the JASS-2021[9] GitHub organization.

[9]https://github.com/JASS-2021

At first glance, the infrastructure setup and preparation seem similar to the challenges of preparing an ordinary project course. However, the distribution and usage of hardware components are fundamentally different from in-person teaching. General computing hardware can be virtualized using commercial PaaS platforms or virtual machines installed at university sites. However, IoT hardware is distinct and cannot easily be virtualized. Instructors cannot rely on participants meeting in person and sharing a centralized hardware setup.

> **Recommendation 1:** Ensure all students have (remote) access to hardware.

Instructors must ensure that students either have hardware at their working location or remote access to enable concepts such as pair deployment. They need to send IoT devices to the participants or create custom build remote access for those devices located at the university sites. If they decide to send hardware to the participants, it needs to be shipped, or a pickup needs to be arranged weeks before the course. The remote setup must include ways for participants to observe the deployments at the sites.

> **Recommendation 2:** Provide stable network infrastructure in all locations.

Sophisticated network infrastructure has to be set up to allow the usage of continuous software engineering technologies to deploy software from outside of the university network safely. We recommend VPN-based access to infrastructure that should not be exposed directly for access from the internet. We also recommend testing the remote and parallel access to remote machines and video streams, providing access to remote locations as shown in Figure 1 and Figure 2.

> **Recommendation 3:** Use a central development platform that supports an easy setup of continuous integration and continuous deployment.

GitHub, Bitbucket and Gitlab are central platforms that enable discussions, decision knowledge, and code in distributed version control. Such platforms are the central location to look up information, in addition to chat and video conferences. GitHub Actions, Bitbucket Pipelines and GitLab CI/CD provide the foundation to teach continuous delivery and continuous deployment, particularly for globally-distributed projects.

In the case study, the students reported that GitHub Actions is easy to learn and allows for customizing pipelines to their needs. Many resources and examples available from the developer community helped the students find a suitable solution for their project. The free-tier for education usage gives a reasonable build time quota for teaching tasks for conducting a project that cannot be open-sourced. If the free quota is insufficient, converting to public repositories and open-sourcing the code could be a solution.

> **Recommendation 4:** Use PaaS and SaaS solutions to reduce the infrastructure setup effort for short-term courses.

Using existing PaaS and SaaS software solutions enables easy collaboration across different universities and reduces the infrastructure setup effort. It does not require compatibility between university systems. Many PaaS and SaaS solutions offer free tiers for limited usage, making them practical for short-term courses.

Nevertheless, free PaaS and SaaS solutions can lead to impractical storage and computation limits. Slack's free message limit to view and search 10.000 messages and pricing model makes this solution impractical for communication-intensive courses, particularly for long-term data analysis. It is also important to note that these platforms must fulfill data-privacy and security standards accepted at the participating universities. Instructors must ensure that students are not mandated to participate in the course and usage of these platforms. They must ensure that as little personal data as possible should be kept on these platforms. All participants must be able to delete any personal data following local data protection and privacy regulations.

> **Recommendation 5:** Apply distributed pair deployment.

Distributed pair deployment is a useful technique for testing and demonstrations involving multiple sites, particularly in the development of IoT devices. In pair deployment, the driver at one site modifies the software and starts a deployment. The navigator at the other site observes the automatic deployment using continuous integration and delivery and is the observer at the other site. In traditional pair programming, the navigator is in the observer position while the driver is typing. In pair deployment, the navigator is in the observer position while the driver executes remote commands. In order to enable successful pair deployment, there has to be on-site expertise at the laboratories in each of the participating universities throughout the course. At least one student or staff member was located at the laboratories to fulfill the role of the navigator at the local site per COVID-19 regulations. Pair deployment can be used to test non-IoT-based software outside the development environment. Future work can apply the pair deployment methodology to continuous software engineering to test and debug CI/CD setups.

In summary, we conclude that teaching continuous software engineering in a global project with IoT applications is teachable and can quickly be learned and applied by students.

ACKNOWLEDGMENTS

We want to thank all students who participated in JASS 2021; without you, this course would not have been a success! In addition, we want to thank the administrative staff at the three universities and JetBrains Research for supporting us in organizing JASS 2021. The project was in part funded by the Imperial College London – Technical University of Munich Education Fund.

REFERENCES

[1] 1991. IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. *IEEE Std 610* (1 1991), 1–217. https://doi.org/10.1109/IEEESTD.1991.106963

[2] Anne Anderson and Shobha Ramalingam. 2021. A socio-technical intervention in bim projects - An experimental study in global virtual teams. *Journal of Information Technology in Construction* 26 (7 2021), 489–504. https://doi.org/10.36680/j.itcon.2021.026

[3] Kent Beck. 2004. *Extreme Programming Explained: Embrace Change* (second edition ed.). Addison-Wesley Professional.

[4] Sarah Beecham, Tony Clear, John Barr, Mats Daniels, Michael Oudshoorn, and John Noll. 2017. Preparing Tomorrow's Software Engineers for Work in a Global Environment. *IEEE Software* 34 (2017), 9–12. Issue 1. https://doi.org/10.1109/MS.2017.16

[5] Jan Bosch. 2014. Continuous Software Engineering: An Introduction. , 3-13 pages. https://doi.org/10.1007/978-3-319-11283-1_1

[6] B. Bruegge, A.H. Dutoit, R. Kobylinski, and G. Teubner. 2000. Transatlantic project courses in a university environment. *Proceedings Seventh Asia-Pacific Software Engeering Conference. APSEC 2000* 2000-Janua, 30–37. https://doi.org/10.1109/APSEC.2000.896680

[7] B. Bruegge, S. Krusche, and L. Alperowitz. 2015. Software engineering project courses with industrial clients. *ACM Transactions on Computinig Education* 15 (2015). Issue 4. https://doi.org/10.1145/2732155

[8] Robert Chatley and Tony Field. 2017. Lean Learning - Applying Lean Techniques to Improve Software Engineering Education. *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering Education and Training Track (ICSE-SEET)*, 117–126. https://doi.org/10.1109/ICSE-SEET.2017.5

[9] Daniela Damian, Casper Lassenius, Maria Paasivaara, Arber Borici, and Adrian Schröter. 2012. Teaching a globally distributed project course using Scrum practices. *2012 2nd International Workshop on Collaborative Teaching of Globally Distributed Software Development, CTGDSD 2012 - Proceedings*, 30–34. https://doi.org/10.1109/CTGDSD.2012.6226947

[10] Dora Dzvonyar, Stephan Krusche, and Lukas Alperowitz. 2014. Real Projects with Informal Models. In *Proceedings of the MODELS Educators Symposium co-located with the ACM/IEEE 17th International Conference on Model Driven Engineering Languages and Systems (MODELS 2014), Valencia, Spain, September 29, 2014 (CEUR Workshop Proceedings, Vol. 1346)*, Birgit Demuth and Dave R. Stikkolorum (Eds.). CEUR-WS.org, 39–45. http://ceur-ws.org/Vol-1346/edusymp2014_paper_5.pdf

[11] Joseph Feliciano, Margaret-Anne Storey, and Alexey Zagalsky. 2016. Student Experiences Using GitHub in Software Engineering Courses: A Case Study. *Proceedings of the 38th International Conference on Software Engineering Companion*, 422–431. https://doi.org/10.1145/2889160.2889195

[12] Peter Gloor, Maria Paasivaara, Casper Lassenius, Detlef Schoder, Kai Fischbach, and Christine Miller. 2011. Teaching a Global Project Course: Experiences and Lessons Learned. *Proceedings of the 2011 Community Building Workshop on Collaborative Teaching of Globally Distributed Software Development*, 1–5. https://doi.org/10.1145/1984665.1984666

[13] J.D. Herbsleb, D.J. Paulish, and M. Bass. 2005. Global Software Development at Siemens: Experience from Nine Projects. *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.* (2005), 524–533. https://doi.org/10.1109/ICSE.2005.1553598

[14] Dee W Hock. 1995. The chaordic organization: Out of control and into order. *World Business Academy Perspectives* 9 (1995), 5–18. Issue 1.

[15] Jez Humble and David Farley. 2010. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation.* 497 pages.

[16] Robert Kraut and Lynn Streeter. 1995. Coordination in Software Development. *Commun. ACM* 38 (1995), 69–81. Issue 3. https://doi.org/10.1145/203330.203345

[17] Stephan Krusche and Lukas Alperowitz. 2014. Introduction of continuous delivery in multi-customer project courses. In *36th International Conference on Software Engineering, ICSE '14, Companion Proceedings, Hyderabad, India, May 31 - June 07, 2014*, Pankaj Jalote, Lionel C. Briand, and André van der Hoek (Eds.). ACM, 335–343. https://doi.org/10.1145/2591062.2591163

[18] Stephan Krusche, Lukas Alperowitz, Bernd Bruegge, and Martin O. Wagner. 2014. Rugby: an agile process model based on continuous delivery. In *1st International Workshop on Rapid Continuous Software Engineering, RCoSE 2014, Hyderabad, India, June 3, 2014*, Matthias Tichy, Jan Bosch, Michael Goedicke, and Magnus Larsson (Eds.). ACM, 42–50. https://doi.org/10.1145/2593812.2593818

[19] Stephan Krusche and Bernd Bruegge. 2017. CSEPM - A Continuous Software Engineering Process Metamodel. In *3rd IEEE/ACM International Workshop on Rapid Continuous Software Engineering, RCoSE@ICSE 2017, Buenos Aires, Argentina, May 22, 2017*. IEEE, 2–8. https://doi.org/10.1109/RCoSE.2017.6

[20] Stephan Krusche, Bernd Bruegge, Irina Camilleri, Kirill Krinkin, Andreas Seitz, and Cecil Wöbker. 2017. Chaordic Learning: A Case Study. *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering Education and Training Track*, 87–96. https://doi.org/10.1109/ICSE-SEET.2017.21

[21] Kati Kuusinen and Sofus Albertsen. 2019. Industry-Academy Collaboration in Teaching DevOps and Continuous Delivery to Software Engineering Students: Towards Improved Industrial Relevance in Higher Education. *Proceedings of the 41st International Conference on Software Engineering: Software Engineering Education and Training*, 23–27. https://doi.org/10.1109/ICSE-SEET.2019.00011

[22] Christian Lescher, Yang Li, and Bernd Bruegge. 2014. Teaching Global Software Engineering: Interactive Exercises for the Classroom. *2014 IEEE 9th International Conference on Global Software Engineering*, 163–172. https://doi.org/10.1109/ICGSE.2014.14

[23] Yang Li, Stephan Krusche, Christian Lescher, and Bernd Bruegge. 2016. Teaching Global Software Engineering by Simulating a Global Project in the Classroom. *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, 187–192. https://doi.org/10.1145/2839509.2844618

[24] Zheng Li. 2020. Using Public and Free Platform-as-a-Service (PaaS) Based Lightweight Projects for Software Architecture Education. *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering Education and Training*, 1–11. https://doi.org/10.1145/3377814.3381704

[25] Ramon Lopez-Viana, Jessica Diaz, Vicente Hernandez Diaz, and Jose-Fernan Martinez. 2020. Continuous Delivery of Customized SaaS Edge Applications in Highly Distributed IoT Systems. *IEEE Internet of Things Journal* 7 (10 2020), 10189–10199. Issue 10. https://doi.org/10.1109/JIOT.2020.3009633

[26] Florian Matthes, Christian Neubert, Christopher Schulz, Christian Lescher, Jose Contreras, Robert Laurini, Beatrice Rumpler, David Sol, and Kai Warendorf. 2011. Teaching Global Software Engineering and International Project Management - Experiences and Lessons Learned from Four Academic Projects. *Proceedings of the 3rd International Conference on Computer Supported Education*, 5–15. https://doi.org/10.5220/0003272600050015

[27] Nancy R. Mead, David Garlan, and Mary Shaw. 2018. Half a Century of Software Engineering Education: The CMU Exemplar. *IEEE Software* 35 (9 2018), 25–31. Issue 5. https://doi.org/10.1109/MS.2018.290110743

[28] Peter Mell and Tim Grance. 2011. The NIST definition of cloud computing. https://doi.org/10.6028/NIST.SP.800-145

[29] Hanna Mäenpää, Samu Varjonen, Arto Hellas, Sasu Tarkoma, and Tomi Männistö. 2017. Assessing IOT Projects in University Education: A Framework for Problem-Based Learning. *Proceedings of the 39th International Conference on Software Engineering: Software Engineering and Education Track*, 37–46. https://doi.org/10.1109/ICSE-SEET.2017.6

[30] Ipek Ozkaya. 2021. The Future of Software Engineering Work. *IEEE Software* 38 (2021), 3–6. Issue 5. https://doi.org/10.1109/MS.2021.3089729

[31] Paul Schmiedmayer. 2020. Apodini: An Internal Domain Specific Language to Design Web Services. *Proceedings of the 21st International Middleware Conference Doctoral Symposium*, 47–49. https://doi.org/10.1145/3429351.3431751

[32] Paul Schmiedmayer, Lara Marie Reimer, Marko Jovanović, Dominic Henze, and Stephan Jonas. 2020. Transitioning to a Large-Scale Distributed Programming Course. *2020 IEEE 32nd Conference on Software Engineering Education and Training (CSEE&T)*, 1–6. https://doi.org/10.1109/CSEET49119.2020.9206239

[33] Ken Schwaber. 1995. SCRUM Development Process. *Proceedings of the OOP- SLA Workshop on Business Object Design and Implementation.*

[34] Outi Sievi-Korte, Kari Systä, and Rune Hjelsvold. 2015. Global vs. local — Experiences from a distributed software project course using agile methodologies. *2015 IEEE Frontiers in Education Conference (FIE)*, 1–8. https://doi.org/10.1109/FIE.2015.7344101

[35] Natalia Silvis-Cividjian. 2019. Teaching Internet of Things (IoT) Literacy: A Systems Engineering Approach. *Proceedings of the 41st International Conference on Software Engineering: Software Engineering Education and Training*, 50–61. https://doi.org/10.1109/ICSE-SEET.2019.00014

[36] Viktoria Stray, Nils Brede Moe, and Mehdi Noroozi. 2019. Slack Me If You Can! Using Enterprise Social Networking Tools in Virtual Agile Teams. *2019 ACM/IEEE 14th International Conference on Global Software Engineering (ICGSE)*, 111–121. https://doi.org/10.1109/ICGSE.2019.00031

[37] Eleni Stroulia, Ken Bauer, Michelle Craig, Karen Reid, and Greg Wilson. 2011. Teaching Distributed Software Engineering with UCOSP: The Undergraduate Capstone Open-Source Project. *Proceedings of the 2011 Community Building Workshop on Collaborative Teaching of Globally Distributed Software Development*, 20–25. https://doi.org/10.1145/1984665.1984670

[38] Noriko Suzuki, Haruka Shoda, Mamiko Sakata, and Kaori Inada. 2016. Essential Tips for Successful Collaboration–A Case Study of the "Marshmallow Challenge". In *International Conference on Human Interface and the Management of Information.* Springer, 81–89.

[39] Shin Hwei Tan, Chunfeng Hu, Ziqiang Li, Xiaowen Zhang, and Ying Zhou. 2021. GitHub-OSS Fixit: Fixing Bugs at Scale in a Software Engineering Course. *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, 1–10. https://doi.org/10.1109/ICSE-SEET52601.2021.00009

[40] Bastian Tenbergen and Nancy R. Mead. 2021. Adapting a Software Acquisition Curriculum to Instruct Supply Chain Risk Management in a Project-Based Software Development Course. *2021 Third International Workshop on Software Engineering Education for the Next Generation (SEENG)*, 36–40. https://doi.org/10.1109/SEENG53126.2021.00014

[41] Daan van Knippenberg, Wendy P van Ginkel, and Astrid C Homan. 2013. Diversity mindsets and the performance of diverse teams. *Organizational Behavior and Human Decision Processes* 121 (2013), 183–193. Issue 2. https://doi.org/10.1016/j.obhdp.2013.03.003

[42] Laurie Williams and Robert R Kessler. 2003. *Pair programming illuminated.* Addison-Wesley Professional.

[43] P. Young, N. Chaki, V. Berzins, and Luqi. 2003. Evaluation of middleware architectures in achieving system interoperability. *14th IEEE International Workshop on Rapid Systems Prototyping, 2003. Proceedings.*, 108–116. https://doi.org/10.1109/IWRSP.2003.1207037

[44] Woody Zuill and Kevin Meadows. 2016. Mob programming: A whole team approach. *Agile 2014 Conference, Orlando, Florida* 3.