Introduction of Continuous Delivery in Multi-Customer Project Courses

Stephan Krusche
TU München
Munich, Germany
krusche@in.tum.de

Lukas Alperowitz
TU München
Munich, Germany
alperowi@in.tum.de

ABSTRACT

Continuous delivery is a set of practices and principles to release software faster and more frequently. While it helps to bridge the gap between developers and operations for software in production, it can also improve the communication between developers and customers in the development phase, i.e. before software is in production. It shortens the feedback cycle and developers ideally use it right from the beginning of a software development project.

In this paper we describe the implementation of a customized continuous delivery workflow and its benefits in a multi-customer project course in summer 2013. Our workflow focuses on the ability to deliver software with only a few clicks to the customer in order to obtain feedback as early as possible. This helps developers to validate their understanding about requirements, which is especially helpful in agile projects where requirements might change often. We describe how we integrated this workflow and the role of the release manager into our project-based organization and how we introduced it using different teaching methods.

Within three months 90 students worked in 10 different projects with real customers from industry and delivered 490 releases. After the project course we evaluated our approach in an online questionnaire and in personal interviews. Our findings and observations show that participating students understood and applied the concepts and are convinced about the benefits of continuous delivery.

Categories and Subject Descriptors

K.6.3 [**Management Of Computing And Information Systems**]: Software Management—*Software development, Software process*; D.2.9 [**Software Engineering**]: Management—*Life cycle, Programming teams, Software configuration management, Software process models*

General Terms

Management

.

Keywords

Continuous Delivery, Continuous Integration, Version Control System, Executable Prototypes, Release Management, Feedback, User Involvement, DevOps

1. INTRODUCTION

In 2010, Jez Humble and David Farley defined the term *Continuous Delivery* (CD). They describe it as a set of practices and principles to release software faster and more frequently. CD extends the workflows and techniques for build- and test automation already know as continuous integration. While continuous integration focuses on the automation of the build process on a central server, continuous delivery extends this approach to all workflows needed for the test and deployment of a new build. CD aims to simplify the release of software and allows shorter feedback cycles between developers and customers. Among these benefits, CD enables teams to continuously track the current build state of the software, it reduces integration and configuration errors, lowers stress when dealing with releases and increases the deployment flexibility. [13]

On the other hand implementing CD is a resource-intensive process. If a software development team wants to use the concepts of CD, it needs to understand and maintain the corresponding tools and it needs to build a deployment pipeline that fits to the requirements of the project.

Even if applying CD might seem too sophisticated in an university setting, we incorporate CD into our software development project course right from the beginning. We see the chance that not only software developers benefit from these principles, but also the results of the projects improve. In 2012 we taught a multi-customer course with 11 companies and 80 students in which we started to apply CD techniques [5].

We structured the course using a project-based organization. As we were not experienced enough with CD at this time, we introduced the integration and delivery tools too late. Our approach how we introduced the workflows to students was also not mature. Students in the 2012 course experienced some techniques like manually releasing an automatically created build, but did not release the software regularly and in a repeatable manner. Most of the deliveries happened at the end of the course close to the client acceptance test [6].

In the 2013 project course we learned from these shortcomings and introduced CD right from the beginning. The application of CD can be quite complex, especially for beginners who need time to understand and apply it in their own project. We cannot introduce all the complexity in the beginning, but need to teach the basic techniques. This enables students to create an initial empty release and to get early in touch with the CD workflows and tools.

In this paper we like to present our approach how to introduce CD to students in a course structure similar to a project-based organization. Thereby we like to address the following questions:

- Understanding (Q1): Do the developers understand the concepts of continuous delivery?
- Teaching Methods (Q2): How can the organization distribute the required knowledge about continuous delivery into all project teams?
- Usage (Q3): How often do developers use the continuous delivery workflow during the project?
- Benefits (Q4): How can the organization convince the developers about the benefits of continuous delivery?

The paper is organized as follows. In section 2 we present our continuous delivery workflow with integrated tools that allow developers to release a software with only a few clicks to the customer. We also show how these tools are connected to each other to create a deployment pipeline. We distinguish between internal and external releases and focus on the improved communication within the team and with the customer.

We think each development team needs a person who is responsible for the activities concerning continuous delivery. We call this person *release manager*. He is responsible to deliver a new version of the software to the customer, whenever the team decides to release it. We describe the role and the responsibilities of the release manager in section 3 where we introduce the organization of our project course. We also explain the role of the *release coordinator* who instructs a cross-project team that consists of all release managers.

Section 4 describes different methods we use to teach and introduce continuous delivery in our course. Our approaches vary from lectures to video tutorials and cross-project teams with specific responsibilities.

In August 2013, after the final presentations of the project course, we evaluated our teaching and introduction approach in an online questionnaire and in personal interviews. We describe the results and the observations of the evaluation of our approach in section 5.

2. WORKFLOW

In this section we describe the deployment pipeline that we use in our multi-customer courses. We first describe the tools we use and continue with an explanation of the workflow.

Humble and Farley model all steps which are needed to deploy and deliver an application to the production environment as a stage-gate process. A deployment pipeline consists of multiple stages as shown in fig. 1.

During its lifecycle a build moves from the integration stage (1) through various testing stages (2) to production (3). In each stage the build is checked against certain quality criteria. If these are fulfilled, the build is promoted to the next stage. Humble also asks for a central repository (A) to store build artifacts from the integration stage. This allows the successor stages to use the same build, e.g. for testing and deployment. [13]

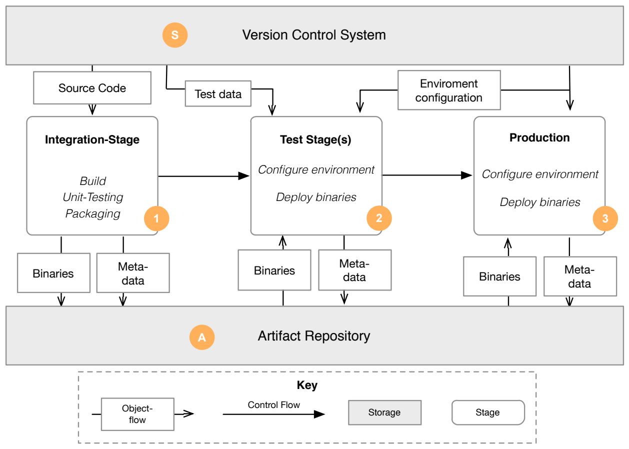

Figure 1: Deployment Pipeline (adapted from [13])

We use this deployment pipeline as base for our continuous delivery workflow. Fig. 2 shows the integrated workflow.

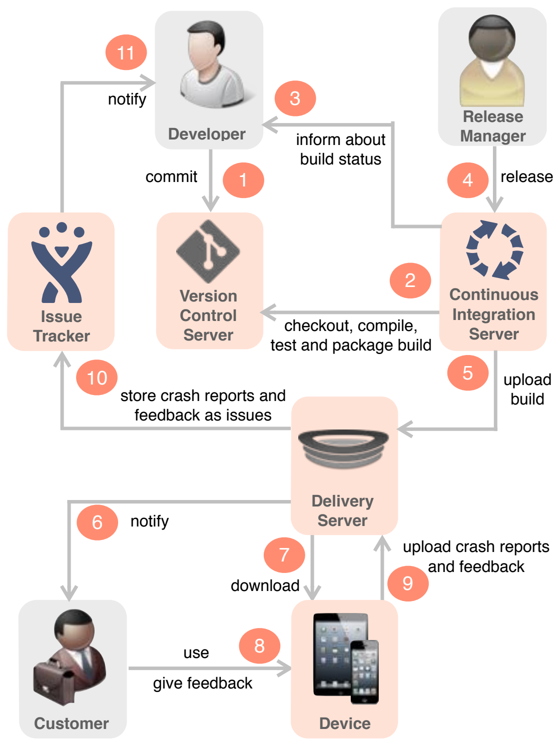
11
notify
Developer
3
inform about build status
Release Manager
commit
1
4
release
Issue Tracker
Version Control Server
2
Continuous Integration Server
checkout, compile, test and package build
10
store crash reports and feedback as issues
5
upload build
Delivery Server
6
notify
7
download
upload crash reports and feedback
9
Customer
use
8
give feedback
Device

Figure 2: Continuous delivery workflow

To implement it we use an *Issue Tracker* to manage the project progress and a *Version Control Server* with support for branches to store source code and configuration data (S). To checkout, build, test and package the source we use a central *Continuous Integration Server* which also fulfills the role of a repository for build artifacts (A). To deliver a build as a release to customers we use a separate *Delivery Server*. This improves the usability for the customer when downloading and installing new builds and allows him to give feedback directly in the software in a structured way.

The delivery workflow starts with a developer committing source code to the version control (VC) server (1). The continuous integration (CI) server regularly asks the VC server for code changes. On each new commit, the CI server checks out the source code, compiles it, runs the unit tests, packages the result (2) and informs the authors of the commit about the build status (3). Together with the development team the release manager decides (4) whether a *potentially releasable*[1] build is uploaded to the delivery server (5). The delivery server now notifies the customer about a new release (6). The customer can then download the release (7). While the customer is using the release (8) crashes are automatically reported to the delivery server. Additionally he can use an embedded mechanism to give structured feedback (9). The delivery server forwards the collected feedback and crash reports to the issue tracker (10) which notifies the developer e.g. about the new issue (11).

Because of visualization purposes the workflow in fig. 2 only shows a limited amount of interaction with the version control server. In fact a developer has more possibilities than just committing source code. Therefore we extend the interaction with the version control server and allow the use of branches. We used a simplified version of the gitflow branching model [8] shown in fig. 3.

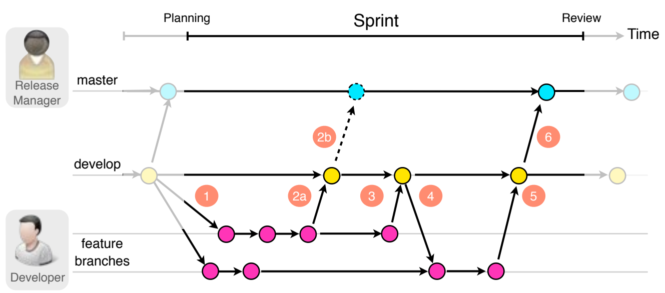

Figure 3: Simplified version of the gitflow branching model (adapted from [8])

In a typical iteration developers work on multiple features and create a feature branch whenever they start to develop a new feature (1). They commit their changes to according feature branch. If a developer wants to share his work with other team members (e.g. when he finishes a feature) he merges his changes into the develop branch (2a, 3, 5). The release manager can decide to revert the merge depending on the code quality of the produced changes. If he accepts the request, the other developers can pull these changes and apply them to their feature branch (4). The team can also decide to merge the changes to the master branch which is the branch for external releases (2b) to obtain feedback from the customers during the sprint.

At the end of the iteration, before the review with the customer, the team needs to release the current state of the software by merging the changes of the develop branch into the master branch (6). All merges to the master branch are maintained by the release manager only.

While this use of branches complicates the collaboration, because developers now need to consider multiple branches and the according merge procedures, it also increases the flexibility. Developers can easily create feature branch releases and deliver them to the customer if the continuous integration and delivery server support this. This allows to get fast feedback if a developer e.g. does not understand all requirements behind a feature or if he is unsure about his concrete implementation.

If we combine the workflow described in fig. 2 with the branching model shown in fig. 3 and apply them to the deployment pipeline in fig. 1, we obtain the deployment pipeline for internal and external releases shown in fig. 4.

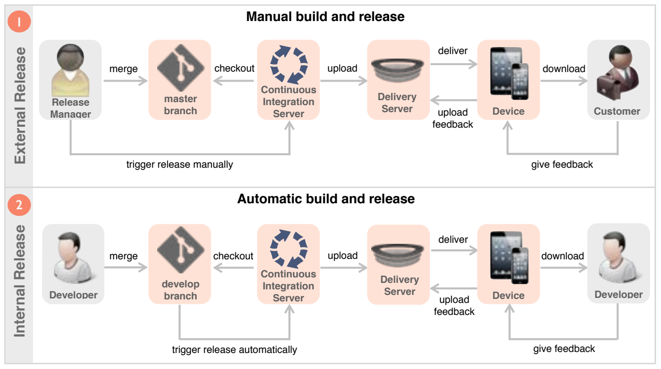

Figure 4: Deployment pipeline with internal and external releases

We distinguish between internal and external releases. An internal release is automatically triggered if a developers merges a feature branch with the develop branch. The continuous integration server automatically builds and tests the code and finally uploads the build to the delivery server. The build is tagged as an internal release so that only developers are notified and can download it (2). If the team decides to deliver a release to the customer, the release manager merges the changes to the master branch. The continuous integration server then automatically builds the release. If the build and all test stages are passed, the release manager manually triggers the upload to the delivery server. The customer receives a notification and can download and test the new version of the software (1).

3. PROJECT COURSE

In this section we describe the structure and organization of the multi-customer project course we conducted in summer 2013. We started teaching similar multi-customer courses in 2008. The course consists of around 10 software engineering projects with real industry partners which expect a working application at the end of the project. Typically 75% of the students in our course are graduates while 25% are still undergraduates. All students are studying computer science. We challenge the students to work together with the customers from industry to solve a real problem within a real deadline in one semester, i.e. typically in three months. This setup is particularly ambitious and requires a high effort for the students as well as for the teaching assistants who prepare and organize the course. However, this commitment leads to the effect that students experience real communication in a real project team with up to eight developers, a coach and a project manager.

[1]All successfully tested builds are potentially releasable [1].

We taught the same course in 2012 and described several aspects of it in [6]. In this paper we focus on the aspects that influence the continuous workflow. The lifecycle model of the course is shown as a horizontal timeline in fig. 5. It shows different activities that the participants apply in the project course like e.g. release management which includes configuration management, continuous integration and CD. The average effort of these activities during a certain phase is visualized for each activity. The figure also includes important events shown as diamonds and time based releases at the end of each sprint as well as some examples for event-based releases within the sprints, both shown as packages.

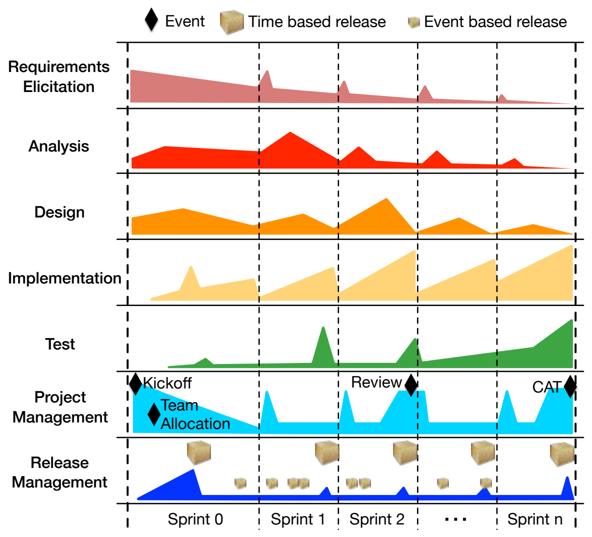

Figure 5: Lifecycle model of our project course (adapted from [6])

The course starts in April with a course-wide kickoff meeting where everybody gets to know each other. In this meeting the customers present their project ideas and try to convince the students for their project. After the team allocation the students start to work on the project. The first milestone is an initial empty release built with the continuous integration server. The teams develop the software in an agile manner using the concept of sprints following an adapted version of Scrum [18]. We think that agile software development and CD nicely fit together for such a project course [11].

Continuous delivery allows the teams to deliver a potentially shippable product increment at any time in the project. This is particularly useful at the end of the sprint (time-based), when they want to deliver this increment to the customer, but it also helps to obtain feedback during a sprint (event-based). We believe that CD bridges not only the gap between developer and operation teams as described with the term DevOps in [12] and [14], but also the communication gap between developers and customers. It enables the idea of continuous user involvement as described in [16] and [17] early in the development process.

Additionally to the first empty release, the students build up the team and gain the required knowledge depending on their project in sprint 0, which lasts about three to four weeks. Additionally they focus on the requirements analysis and on the top-level design of their application. The following sprints last two to three weeks. Each sprint leads to a potentially shippable product increment, a quite mature executable prototype. While we except each team to deliver not less than one release at the end of each sprint, we motivate the students to deliver the current version of the software to the customer whenever they want to obtain feedback (event-based).

After two third of the course we synchronize all course participants in the design review. In this course-wide meeting each team presents its current development state to all other teams and to all customers. At the end of the course, all teams present and demonstrate their final application in another course-wide meeting, the client acceptance test.

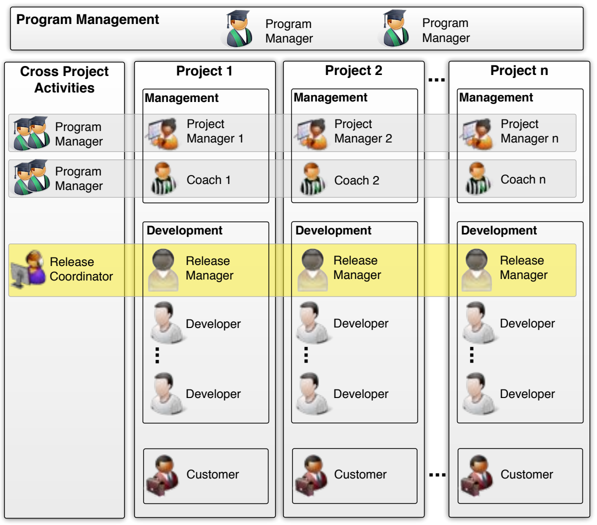

Figure 6: Organization (adapted from [6])

Figure 6 shows the project-based organization in the course which is organized by two program managers, who are experienced teachings assistants. Each project team is represented as a vertical bar and includes a project manager and a coach, the development team as well as a customer. The project manager is a teaching assistant from the university who is responsible for the team success. The role of the project manager is comparable to the role of a scrum master [18]. The coach is an experienced student, who took the course in the year before. He learns agile project management by observing the project manager and is responsible for meeting management and task organization. He helps the development team with detailed design questions like e.g. the usage of architectural styles or design patterns. Six to eight students form the development team which is self-organizing, cross-functional and responsible for the development and the delivery of the software. The customer is an employee of the company and takes the role of the product owner. If it happens that the customer is not available, the project leader overtakes the role of a proxy customers [4].

Additionally we setup multiple cross-project teams in the course, each responsible for a certain topic. The cross-project teams consist of a developer from each project team. In two separate weekly meetings the project managers and the coaches (including the release coordinator) report the status of their projects to the program management and discuss important issues. The program management uses these meetings to distribute important information into the teams. The release coordinator, a teaching assistant who is experienced with CD, leads the release management team, shown as horizontal box in fig. 6. The team members of this cross-project team are responsible for all activities concerning version control, continuous integration and continuous delivery in their team. They meet biweekly to share knowledge about tools and workflows, to synchronize their understanding about CD and to discuss and resolve potential issues with workflows and tools.

We use the following tools to implement the workflow shown in fig. 2:

- JIRA [2] as an *Issue Tracker*
- Stash [2] as *Version Control Server* with git [20] as distributed version control
- Bamboo [2] as a *Continuous Integration Server*
- HockeyApp [3] as a *Delivery Server*

We assessed different tools using the questions mentioned by Swartout in [19]. We also chose the mentioned tools because they are free of charge for universities, they integrate tightly to each other, they are powerful but easy to use and they fulfill all the requirements from the workflow shown in fig. 2.

4. TEACHING METHODS

In this section we explain our approaches for transferring the knowledge about continuous from the program management and the release coordinator into the teams. We mainly use three ways that we describe in the following.

4.1 Cross-Project Team

The release management team meets the first time in the mid of sprint 0 after all projects and tools have been setup. The first meeting of the team is important. After becoming acquainted with each other, the release coordinator explains the team goals and the responsibilities of the release managers.

He then gives a tutorial about the first steps concerning version control, continuous integration and continuous delivery using an example project he prepared in advance. In the tutorial he shows all important steps of the workflow so that the release manager can prepare the first empty release within the following week. He also explains the simplified version of gitflow as shown in fig. 3 and how to create branches in git. Additionally the coordinator assigns tasks (e.g. in the issue tracker) to all release managers to track their progress until the next team meeting. It is important, that he also controls whether the release managers fulfill these tasks, e.g. by talking to the coaches or project managers. If he notices that some of the release managers do not understand a concept correctly, he talks to the release manager in person to clarify any open issues.

Additionally the release coordinator is the main contact person for the release managers if problems or questions arise. On the one hand he needs the technical expertise to find solutions to common problems when applying CD, e.g. how to fix broken build scripts or how to repair misconfigured build steps. On the other hand he needs skills to communicate with release managers, to teach CD techniques and to track the status of all projects.

The release coordinator introduces more and more steps and concepts of our deployment pipeline shown in fig. 4 in the following regular meetings (e.g. every two weeks). He takes care that all release managers understand and apply the steps of the pipeline correctly and use them in their teams. He also introduces the idea of a wallboard that shows the current integration and delivery state of all branches within a project or all projects within the whole project course. An example of such a wallboard is shown in fig. 7. We observed that especially the wallboard visualization motivates the students to always have a potentially releasable version of their software in the version control system.

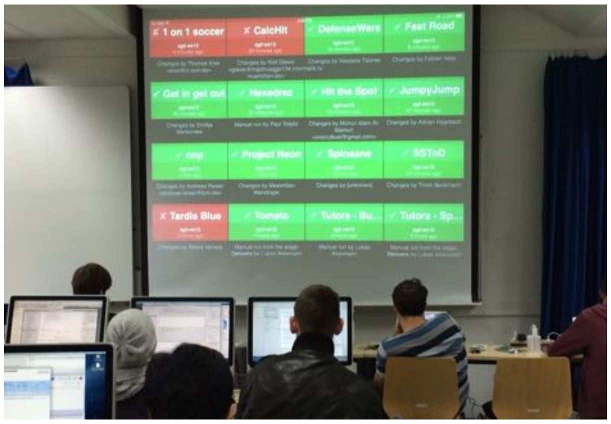

Figure 7: Wallboard showing whether the current version of the software is releasable or not.

It is important to use different concepts to build up knowledge in the release management team. We also assign smaller challenges between two meetings to the cross-project team members. For instance we ask them to collect and present two or three best practices for continuous integration, e.g. like described in [10] and [9]. Another possibility is to ask the students to describe how their project team members use the CD workflow. This facilitates that the release managers take responsibility for their role, internalize the knowledge and distribute it to their team.

4.2 Lecture

Besides the practical parts, i.e. development of a real application, the project course also consists of a weekly course-wide meeting. The program management uses this fixed time slot for the important events like kickoff, icebreaker, design review and client acceptance test (also see fig. 5). In between these events the slot is used to teach important software and usability engineering concepts to the students in mini lecture[2], e.g. meeting management, agile methods and prototyping. The release coordinator uses one of the slots to give a presentation about release management. He should prepare at least one week to create the material, the examples and the slides for the presentation.

This presentation is divided into three parts:

1. *Git as a distributed version control system* explains the main concept of git, like e.g. branching and merging,

[2]These mini lectures last about 30 minutes.

and introduces a simplified version of git-flow [8] as our preferred branching model (see fig. 3).

2. *Bamboo as a continuous integration server* shows how to create a deployment pipeline with multiple stages in Bamboo and how to deal with branches

3. *HockeyApp as a delivery server* explains the differences between internal and external releases and how this can be configured within our workflow.

The main goal of the presentation is to create a common understanding about continuous delivery. To deal with the complexity, we teach the workflow shown in fig. 2 in these three steps.

Nevertheless the presentation has also other purposes. It shows the theoretic concepts behind the practical benefits of CD and motivates all course participants, not only the release managers of the cross-project team, to apply the workflow with the given tools. The presentation also has an experiential learning aspect. Students who already made concrete experiences with CD workflows and tools, e.g. in earlier projects or while creating the initial release, reflect about their observations and generalize the concepts used in the CD tools [15].

Additionally the release coordinator demonstrates live tutorials with the example project, shows the current wallboard (see e.g. fig. 7) and answers questions from the audience.

4.3 Tutorials

The third teaching method we used were tutorials. We published them as step for step manuals with detailed instructions and pictures into our central wiki knowledge management system and distributed the links via email. The advantage of this method is that the team members are able to read these manuals at home and to post questions or comments if they do not understand a concept. We also uploaded short video tutorials that we screen captured during the course-wide presentation so that students could watch them any time if they want to know how a specific step in the workflow is realized with one of the given tools.

5. EVALUATION

In the following section we describe the results of our evaluation and the observations in the 2013 project course [7]. After the client acceptance test we conducted an online questionnaire with all participants. We invited all students of the course including the release managers and coaches to take part in an online survey. The questionnaire consisted of 16 questions, took about 15 minutes and was not mandatory for the students. As we did the survey during the exam phase, we gave the students four weeks after the final project presentations to complete it. After two weeks we sent out a reminder e-mail. We received 41 valid responses out of 90 students, about 30% of the answers were given by undergraduate and about 70% by graduate students.

In the questionnaire we covered the following four topics. First we asked the participant about their prior experience with CD and evaluated whether they understood the main concepts (Q1). Second we evaluated our teaching methods, i.e. whether the students used our learning resources described in section 4 and how much knowledge they gained through each of them (Q2). Third we asked how often they used the workflows for version control, continuous integration and continuous delivery (Q3). Fourth we asked if they understood the main goals and benefits of continuous delivery (Q4). We then compare the deployment pipeline of the 2012 project course with the one from 2013 to support the results our survey. Finally we report some interesting observations we made during the course and in personal interview after the course and discuss the limitations of our evaluation.

5.1 Understanding

We first asked the participants about their **prior** knowledge with version control, continuous integration and continuous delivery. Fig. 8 shows, that 70% of the participants had less than three months experience on these topics while 30% of the participants had more than three months experience.

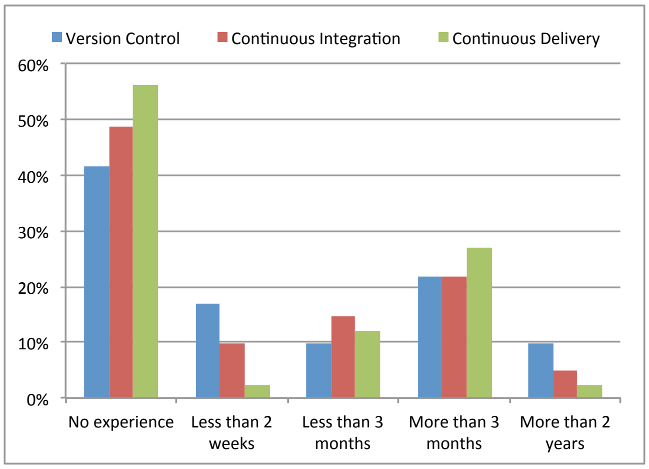

Figure 8: How much prior experience did you have in ...?

In the next question we asked the participants for a self-assessment of their knowledge **after** the course. Fig 9 shows, that more than 90% of the participants think they could explain the concepts we taught to a friend.

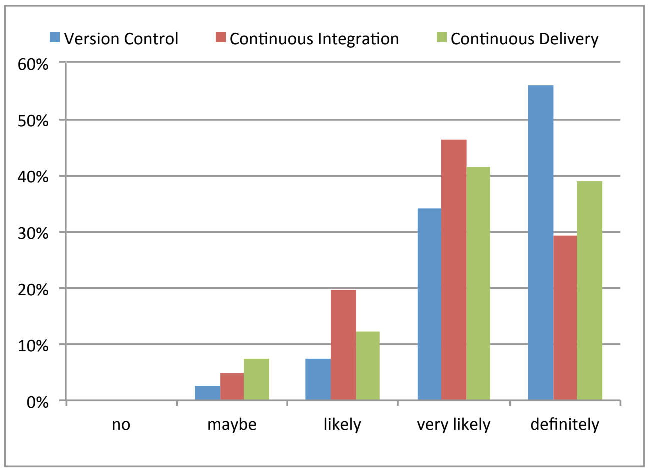

Figure 9: Are you able to explain the concept of ... to a friend?

5.2 Teaching Methods

In the next part of the questionnaire we asked how the participants gained knowledge about continuous delivery in the project course. Fig. 10 shows that the students used all teaching methods we introduced, but that they preferred to communicate within the cross-project team and to listen to the lecture.

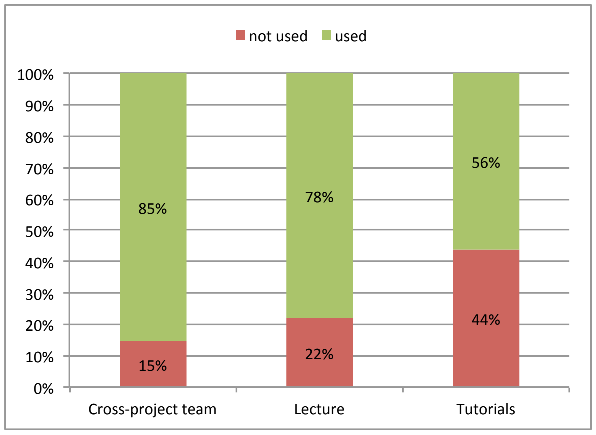

Figure 10: Did you consume the teaching method ...?

We additionally asked participants of the questionnaire how much knowledge they gained from the teaching methods they consumed. Fig. 11 shows that the student gained knowledge through all different teaching methods. However in their opinion the cross-project team communication added the most value. 70% of the students experienced a medium or high knowledge gain through the cross-project team.

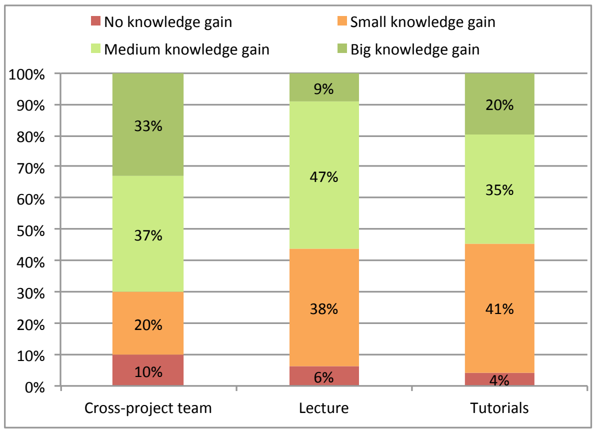

Figure 11: How much knowledge did you gain using...?

5.3 Usage

In the third part of the questionnaire we evaluated whether the students performed specific tasks within version control, continuous integration and continuous delivery. We asked if the students experienced some advanced version control tasks like e.g. solving a merge conflict in a git repository or creating an additional feature branch. For continuous integration we wanted to know if team members e.g. checked the status of a build or whether they started a build manually. To evaluate the continuous delivery usage, we asked e.g. if a student delivered a potentially releasable build to the customer or if he read a crash report.

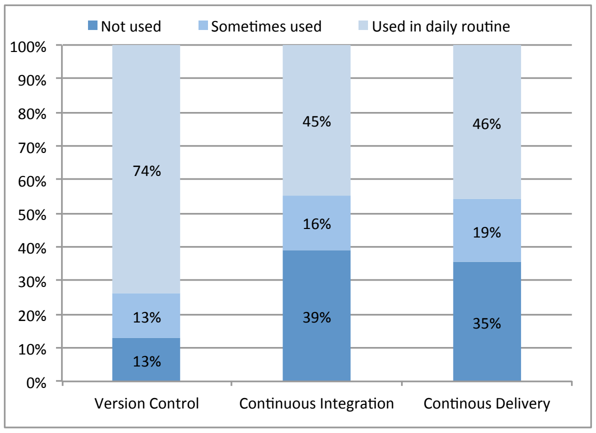

Figure 12: How often did you perform the task...?

We summarized the answers for the three parts of the deployment pipeline in fig. 12. It shows that version control tasks were applied most frequently and also continuous integration and continuous delivery was used in the daily routine by 45% of the students.

5.4 Benefits

Finally we evaluated whether students see benefits in version control, continuous integration and continuous delivery. We e.g. asked whether the students think that branches help in developing and delivering multiple prototypes at the same time. We also asked if they think that continuous integration leads to faster error detection and correction and if they are able to collect more and better feedback from customers when applying CD. We summarized the results of these questions in fig. 13. It shows that most of the students see benefits. On average only 20% do not perceive any advantages.

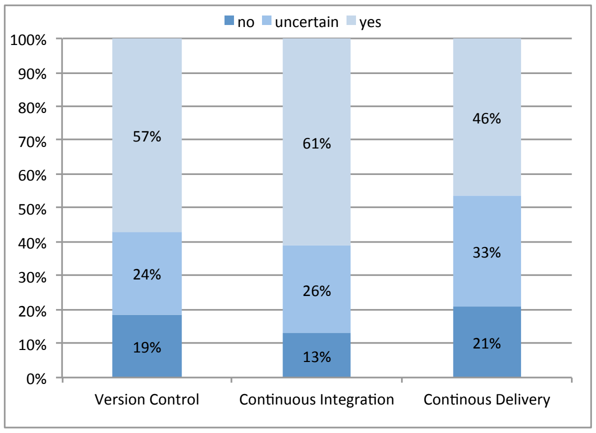

Figure 13: Do you see benefits when using ...?

We also asked the students whether they would like to use the concepts we introduced in a future project. As shown in fig. 14 70% of the participants would at least likely apply the concept taught in the course in their own projects.

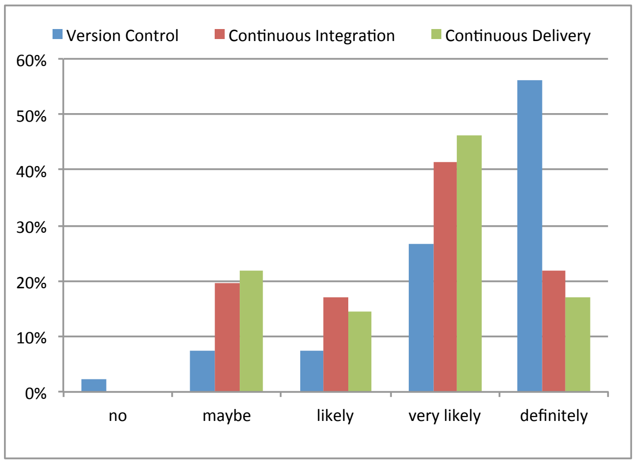

Figure 14: Would you use these concepts in a future project?

5.5 Usage Numbers

Before we continue to talk about the observations we made in the personal interviews, we like to show the number of commits, builds, releases and downloads and to compare them with the course in 2012. We did not use continuous integration and continuous delivery in multi-customer courses before 2012, so we cannot compare the numbers to earlier courses. The numbers for 2012 are shown in fig. 15, the numbers of 2013 in fig. 16.

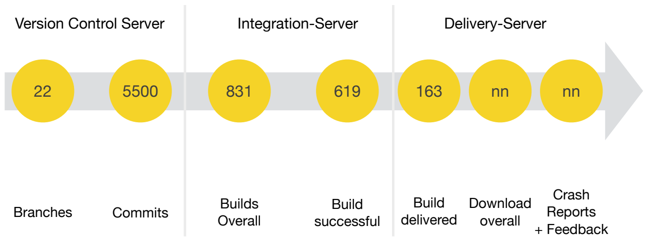

Figure 15: Build pipeline of 2012 in numbers

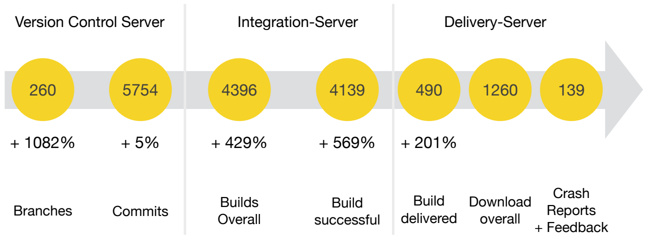

Figure 16: Build pipeline of 2013 in numbers

While in 2012 only a central version control server was available from the first day, in 2013 the infrastructure for continuous integration and continuous delivery was in place right from the beginning. While the amount of commits has not changed much, we recognize that the number of branches increased strongly as we introduced a branching model in 2013. In 2013 more than 75% commits led to a build in the continuous integration server while in 2012 only 15% of the commits led to a build. This is caused by the fact that the build server was available earlier in 2013 whereas in 2012 we introduced it after two third of the project course. Also most of the builds (in fact more than 94%) were successful in 2013. Consequently the absolute number of delivered builds to the costumer is three times higher in 2013 because the teams were able to deliver releases from the first day using an automated workflow. In 2012 we did not measure the number of downloads, crash reports and feedback. However, in 2013 we can see that each delivered build was downloaded 2.5 times on average and that crash reports and the built-in feedback feature were used to create 139 reports.

The usage of branches, the increase in delivered builds and the higher percentage of successful builds compared to 2012 underline the results of the evaluation.

5.6 Observations

Finally we report on observations we made during the course and we perceived in personal interviews with the release managers after the course. We interviewed the release managers in the final release management cross-project team meeting and asked them about their personal opinion and feedback.

One observation during the course was that some of the customer used the delivered prototypes to create screenshots of the user interface. They marked errors like e.g. misplaced user interface elements and sent these screenshots as well as additional improvements suggestions back to the developers which could then react to this feedback. We think that without our automated deployment pipeline this would not have been possible in such an easy way.

One of the teams even incorporated its own feedback gesture into the user interface of the application and received more than 10 in-app feedback reports. During the project course we observed different ways how teams collected their feedback. The students collected most of the feedback in customer meetings or via phone or online meetings. Structured approaches like the feedback mechanism in our delivery server were not used frequently. We think that these structured mechanism do not have the same quality as personal interaction because they are asynchronous and in written form. Also the teams had regular customer meetings and there was no need to rely on electronic feedback mechanisms. Such approaches make more sense if a high amount of test users is available or if personal meetings are not possible.

Another interesting observation was that some customers requested additional releases and were curious to know when the next release will happen.

5.7 Limitations

With this evaluation we aimed to explore whether students learned the concepts we taught, which teaching methods work best and whether students acknowledge the benefits of continuous delivery. We see threats to the validity of our evaluation and want to discuss them briefly. We might have the problem of selection bias because not all participants of the course took part in the online questionnaire. Some teams used the concepts we taught more frequently than others, e.g. because of more experienced students or because of customer requests. To alleviate this threat we asked the students in which team they worked. As we have at least three responds from each team and as we also analyzed the results on a team basis and did not find deviations, we think that this threat is low. Additionally we observed the same results in the personal interviews and all interviewees agreed with our findings.

Another problem might be, that participants gave answers which do not reflect their work practice, because they knew that we like to publish the results. We guaranteed the participants the complete anonymity and addressed this threat by that. We know that we cannot generalize our findings for industry because we have a different setup in the university. But we think that we come quite close to a project-based organization with the structure of our course and believe that at least some of the results are also valid in industry.

6. CONCLUSION

In this paper we introduced a continuous delivery workflow based on a deployment pipeline by Humble and Farley. We explained how we integrated it into our multi-customer project course right from the beginning. We described three different teaching methods, a cross-project release management team, lectures and tutorials. These methods ensure that knowledge about required workflows and tools is distributed to all participants in our project-based organization. We evaluated the understanding, the teaching methods, the usage and the benefits of continuous delivery within our approach. For the evaluation we used an online questionnaire and personal interviews after the project course. We found that most of the students understood and used our workflows and tools, see their benefits and even want to use them in future projects.

One of the key success factors is the cross-project release management team. On the one hand a release manager is responsible for applying continuous delivery within each team, on the other hand the release coordinator is in charge of introducing and tracking all activities of all development teams. This helps to spread the knowledge and ensures that all teams use version control, continuous integration and continuous delivery in a proper way.

We think that the introduction of continuous delivery in project courses is worth the effort! In fact it increases the effort for the teaching assistants who organize the course, because they need to setup the tools and workflows, but it also leads to many benefits. The additional effort varies based on the teams experience with agile methods and continuous delivery. Using a cross functional team to introduce the workflows minimizes the additional work for the teaching assistants.

The introduction of continuous delivery simplifies and automates release management so that developers deliver their software earlier, more often and on a regular base. They are not only able to deliver time-based at the end of sprint, but also event-based during a sprint, i.e. whenever they need feedback. This improves the communication between developers and customers because of shorter feedback cycles. It might even lead to continuous user involvement, if there are enough test users like e.g. in a beta test.

We believe that some of the findings of our evaluation are also valid for project-based organizations in industry. We want to evaluate whether this is the case in an industry project in 2014.

7. ACKNOWLEDGMENTS

We want to thank all participants of the project courses, in particular the teaching assistants and coaches. We also thank the students who filled out our online survey and who talked to us in personal interviews.

8. REFERENCES

[1] Atlassian. Atlassian blog, 2012. `http://blogs.atlassian.com/2012/01/continuous-delivery-with-bamboo-stages`.

[2] Atlassian. Atlassian software suite, 2013. `http://www.atlassian.com/software`.

[3] Bit Stadium GmbH. Hockeyapp, 2013. `http://www.hockeyapp.net`.

[4] B. Bruegge and A. H. Dutoit. *Object Oriented Software Engineering Using UML, Patterns, and Java (Third Edition)*. Prentice Hall International, 2009.

[5] B. Bruegge, S. Krusche, and M. Wagner. iOS Praktikum, 2012. `http://www1.in.tum.de/ios12`.

[6] B. Bruegge, S. Krusche, and M. Wagner. Teaching Tornado: from communication models to releases. In *Proceedings of the 8th edition of the Educators' Symposium*, pages 5–12. ACM, 2012.

[7] B. Bruegge, S. Krusche, and M. Wagner. iOS Praktikum, 2013. `http://www1.in.tum.de/ios13`.

[8] V. Driessen. A successful git branching model, 2010. `http://nvie.com/posts/a-successful-git-branching-model`.

[9] P. M. Duvall, S. Matyas, and A. Glover. *Continuous integration: improving software quality and reducing risk*. Pearson Education, 2007.

[10] M. Fowler. Continuous Integration, 2006. `http://martinfowler.com/articles/continuousIntegration.html`.

[11] P. Gfader. Use scrum and continuous delivery to build the right thing, 2013. `https://www.scrum.org/Portals/0/Documents/Communityum.orgWhitepaper_ContinuousDelivery.pdf`.

[12] J. Humble. Devops: A software revolution in the making? *Cutter IT Journal*, 24(8), 2011.

[13] J. Humble and D. Farley. *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education, 2010.

[14] J. Humble and J. Molesky. Why enterprises must adopt devops to enable continuous delivery. *Cutter IT Journal*, 24(8):6, 2011.

[15] D. A. Kolb et al. *Experiential learning: Experience as the source of learning and development*, volume 1. Prentice-Hall Englewood Cliffs, NJ, 1984.

[16] W. Maalej, H.-J. Happel, and A. Rashid. When users become collaborators: towards continuous and context-aware user input. In *OOPSLA*, 2009.

[17] D. Pagano and B. Bruegge. User involvement in software evolution practice: a case study. In *Proceedings of the 2013 ICSE*, pages 953–962. IEEE Press, 2013.

[18] K. Schwaber and M. Beedle. *Agile software development with Scrum*. Prentice Hall PTR, 2002.

[19] P. Swartout. *Continuous Delivery and DevOps: A Quickstart Guide*. Packt Publishing Ltd, 2012.

[20] L. Torvalds. Git, 2013. `http://www.git-scm.com`.