Rugby: An Agile Process Model Based on Continuous Delivery

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

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

Bernd Bruegge
TU München
Munich, Germany
bruegge@in.tum.de

Martin O. Wagner
TU München
Munich, Germany
wagmarti@in.tum.de

ABSTRACT

In this paper we introduce Rugby, an agile process model that includes workflows for the continuous delivery of software. It allows part-timers to work in a project-based organization with multiple projects for the rapid delivery of prototypes and products.

We show how continuous delivery improves the development process in two ways: First, Rugby improves the interaction between developers and customers with a continuous feedback mechanism. Second, Rugby improves the coordination and communication with stakeholders and across multiple teams in project-based organizations with event based releases.

We have evaluated Rugby in two large university software engineering capstone courses with up to 100 participants working in 10 simultaneous projects with industry partners in 2012 and 2013. We describe the metrics used in the evaluation. First results indicate that Rugby increases the frequency and quality of the interaction between developers and customers leading to improved results in the delivered products.

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

Agile Methods, Release Management, Feedback, Continuous Delivery, Continuous Integration, Version Control System, Executable Prototypes, Communication Models, User Involvement, Software Evolution

1. INTRODUCTION

With the incorporation of agile methods, software engineering projects have gained more flexibility, efficiency and speed. Scrum, for example, is based on the creation of a potentially shippable product increment at the end of each sprint. Scrum focuses on small teams with full-time availability of the developers working face-to-face. Another feature is the early recognition of risks. Daily Scrum meetings allow the participants to report open issues such as impediments throughout the project. [20]

In some software development organizations, developers are involved in more than one project. For example, application domain experts usually participate in several projects simultaneously. Also students, who are participating in software engineering courses, cannot work full-time on the project, as they have to take other courses as well.

Classical software lifecycles require that all requirements are completely identified upfront and cannot be changed after the requirements elicitation phase. Agile methods have softened but not eliminated this constraint. For example in Scrum the requirements identified in a sprint backlog cannot change for the duration of the sprint and product increments can only be released at the end of a sprint. The sprint lifecycle in Scrum consisting of sprint planning, development and sprint review is rather rigid because it is time based. The rationale is that Scrum intends to protect the developers from frequent disturbances by the product owner.

However, we believe - especially during requirements elicitation - that this constraint based on time is overprotective. Requirements in the early phases of a project are often imprecise and volatile, requiring interaction as soon as issues arise. Even during a sprint it should be possible to disambiguate or refine requirements. Postponing these activities to the end of a sprint would only delay the development process. A developer should be able to create and publish increments even during a sprint to receive feedback.

Feedback is an important source for requirements after the delivery and is required for software evolution. With

the emergence of continuous delivery mechanisms and tools, incorporating feedback into the development process has become possible. Rugby presents a lightweight methodology to develop and release rapid prototypes and to learn from feedback comments in rapid parallel cycles[1].

This paper is organized as follows. Section 2 describes the environment of Rugby, in particular the interplay between development, integration, delivery and target environment to enable the incorporation of feedback from customers and users. Section 3 presents Rugby's process model. Rugby takes elements from the Unified Process [14]. In particular Rugby includes continuous delivery and user feedback as additional workflows in the software lifecycle model. Section 4 explains in detail how Rugby integrates continuous delivery to allow event-based releases of increments. It defines the requirements for the continuous delivery workflow and shows how the workflow fulfills these requirements. Section 5 describes the case study, where we have used Rugby in two large project courses and presents initial results.

2. RUGBY ENVIRONMENT

In this section we describe the environment and organization of Rugby. Rugby is designed to be used in project-based organizations with multiple projects.

Fig. 1 shows a typical project team in Rugby. It consists of up to eight developers, a team leader and a project leader. The project team is self-organizing, cross-functional and therefore responsible for all aspects of development and delivery of software.

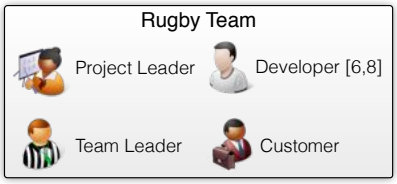

Figure 1: Rugby Team

The project leader and the team leader fulfill a role similar to a scrum master while being in a master-apprentice relationship. While the project manager is already experienced, the team leader is an experienced developer. Thus, he is familiar with the infrastructure and the organizational aspects of Rugby.

One task of the team leader is to organize the first team meeting and to ensure that the team organizes all following team meetings in a structured way. In the first team meeting, he takes the role of the primary facilitator and introduces the other two important roles in a meeting, the minute taker and the timekeeper [2]. In the following meetings, these roles rotate between the developers so that they also take responsibility in the meeting organization.

The job of the team leader is then to make sure, that the developers organize the team meetings appropriately. If e.g. the timekeeper does not interrupt, if the team members discuss too long on an unimportant point of the agenda, the team leader need to interfere and remind the timekeeper about his job. During the project, the team leaders learn essential management skills by observing the behavior and actions taken by the project leader. Another important task of the team leader is problem solving and the communication of problems to the project leader and the program management (see fig. 3).

The customer has a similar role as the product owner. Typically there are different types of customers in software engineering projects. If the customer of a project does not have enough knowledge in the application domain or is not able to make decisions, the project leader helps him. Also if the customer is not available due to time reasons or a large physical distance, the project leader takes the role of a proxy customer [2].

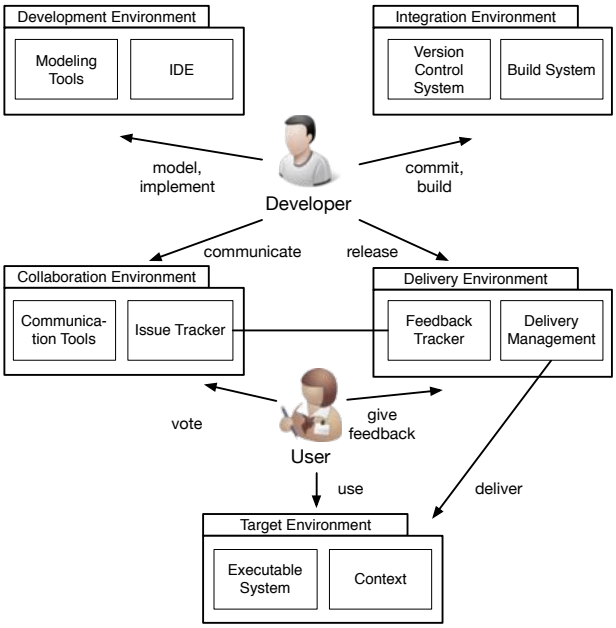

Figure 2: Rugby's Eco-System (adapted from [4])

Figure 2 shows the eco-system of Rugby. We divided the eco-system into five environments. A developer interacts with the collaboration, development, integration and delivery environment, a user interacts with the collaboration, delivery and target environment. The focus in Rugby is particularly on the collaboration and delivery environments because they bridge the communication gap between developers and users. A user is notified from the delivery environment if a new release is available and can then use the software in his target environment. Feedback of the user is stored in the delivery environment and then forwarded into the collaboration environment, e.g. as feature request. A user can also vote certain features in the collaboration environment.

Figure 3 shows the project-based organization of Rugby. Each development team is represented as a vertical bar, e.g. Project 1. Additionally multiple cross-project teams are formed in Rugby to further support certain expertise in the development teams. One of these teams is led by

[1]In the sport of Rugby, a Scrum is a method to restart play after a foul or when the ball has gone out of play. In that sense, Scrum handles only the exceptions of the game. The use case of a Rugby player passing the ball laterally to another player running in parallel formation is a better metaphor for describing the continuous interaction between developers, customers and users.

the release coordinator, who is responsible for release and feedback management of all projects. Release management includes all activities concerning version control, continuous integration and continuous delivery. The release management team is shown as horizontal box in fig. 3 and consists of one team member of each development team, the release manager.

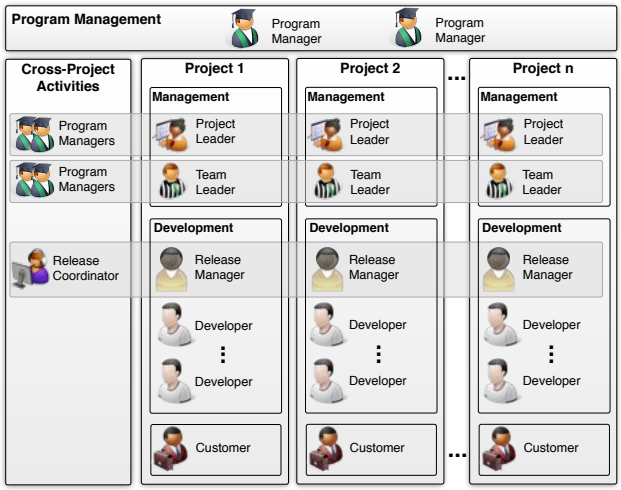

Figure 3: Rugby's Organization (adapted from [4])

Cross-project teams meet weekly or biweekly to build up and share knowledge, to synchronize their understanding about tools and workflows and to resolve potential issues. While the cross-project teams are the main resource for team members to gain knowledge on e.g. continuous delivery practices, there should also be other resources like workshops or tutorials to learn the most important aspects about release and feedback management. In the beginning of the project, tailored tutorials show the developers how to use the tools. During the projects, team members reflect over the actual tool usage in retrospective meetings to improve upon common mistakes and to build best practices. With these experiential learning techniques, a culture of continuous improvements and continuous learning within the teams should be established [13].

3. RUGBY PROCESS MODEL

In this section, we describe Rugby and its workflows as well as the rationale behind it. Based on Takeuchi's and Nonaka's paper from 1986 [21] we use the term Rugby to describe a lightweight process model based on Scrum [20] that is influenced by the Unified Process [14]. In Rugby self-organizing teams develop software in project-based organizations using the concept of sprints following Scrum.

Agile software development and continuous delivery nicely fit together. [8] Continuous delivery bridges not only the gap between developers and operations like described in [10] and [12], but also the gap between developers and customers and users. It enables the idea of continuous user involvement as described in [16] and [18] and fits nicely into the ideas behind the agile manifesto where working software and customer collaboration are more important than comprehensive documentation and contract negotiation. [1]

According to Fowler part-time developers are common in industry projects [7]. If too many part-time developers with different schedules work in a team, daily scrum meetings are not feasible. Therefore Rugby proposes weekly meetings.

Rugby focuses on innovation projects where problem statements are formulated as visionary scenarios and where requirements and technologies can change during the project [4]. In innovative projects, customers typically want developers to explore multiple ideas before they decide how their vague requirements will be implemented. The sawtooth model, an adoption of the V-model, also addresses incomplete and ambiguous requirements [19] and we reuse some of its concepts in Rugby.

During the sprint planning meeting, the team baselines the visionary scenarios for the upcoming sprint so that the sprint backlog includes a defined set of requirements. This means that the customer chooses requirements to work on and specifies them detailed enough that the developers can start working. However, he does not need to fully describe them, and he can still challenge the developers to come up with their own ideas of how to realize a vague requirement and turn it into a potentially shippable product increment which we also call executable prototype.

Implementing a visionary scenario during the sprint might raise new questions for the team members, which they could not have thought of in the sprint planning meeting. Presenting a first mockup for the visualization of a user interface could also lead to a requirement change because the customer might have different expectations that he was not able to express in words during the sprint planning meeting. Work that could be done during the same sprint would shift to the sprint planning meeting of the next sprint if customer collaboration and changing requirements during a sprint would be disallowed.

In difference to Scrum, Rugby allows that requirements are further discussed and negotiated within the sprint. Event-based releases using the continuous delivery workflow presented in section 4 help the team to illustrate the current realization of a requirement and to obtain feedback whether the team is on the right track. Therefore the team does not have to wait until the end of a sprint to deliver software to the customer and can save time and increase the quality of the product increment that is delivered at the end of the sprint.

Another emphasis of Rugby is the organization of multiple projects in an agile manner. There are approaches for scrum of scrum meetings (e.g. used in distributed agile development, see [9] and [17]), but synchronizing multiple projects with different problem statements can become very time-consuming. In Rugby, developers and the management use executable prototypes to report about development status and to discuss important issues. We think this can significantly improve the quality of the communication and decrease the time for unnecessary discussion.

The lifecycle model of Rugby is shown as timeline in fig. 4. It shows different activities that team members apply in parallel (adapted from the unified process) including release management and feedback. The average effort of these activities during a certain phase is visualized as the area in the horizontal bars. It also includes important milestones, like the kickoff of projects, a review to synchronize multiple projects and a client acceptance test (CAT) at the end of the projects. These milestones are shown as black diamonds.

In Sprint 0, which lasts two to four weeks depending on the staffing necessities, the teams focus on team building, gain the required technical knowledge depending on the problem statement and start with the requirements analysis phase. Here the teams also get familiar with release management techniques, in particular version control, continuous integration and continuous delivery, as well as with feedback management. The goal is to create an initial empty time based release to show that the release management and feedback capabilities are available. This can happen already in the mid of Sprint 0 as shown in fig. 4. Experienced teams may perform this initial sprint faster than newly established ones.

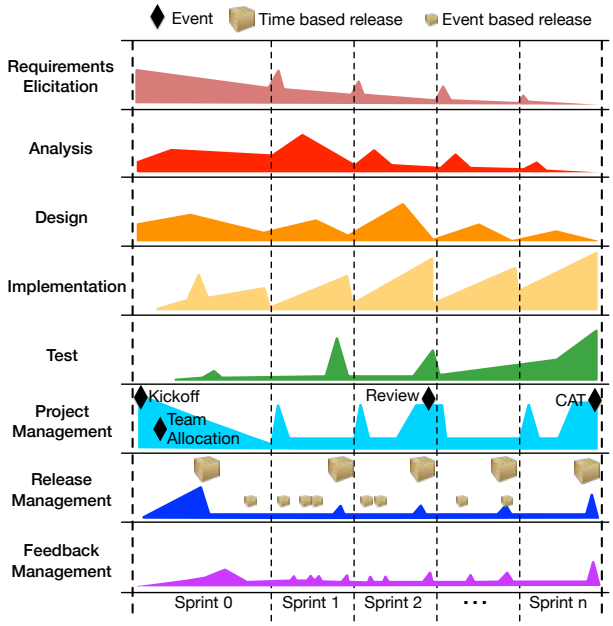

Figure 4: Rugby's Lifecycle model (adapted from [4])

The following working sprints lasts between two and four weeks[2], depending on the innovation of the project. Each of these sprints lead to a potentially shippable product increment. In Rugby, these releases are seen as communication models because they facilitate the communication between multiple project participants.

Especially in projects where user interfaces are an important part of the development, project participants can hardly discuss important issues without having an executable prototype running on a device in the target platform. While Rugby expects each team in the project-based organization to deliver at least one time-based release at the end of each sprint, it motivates the teams to release their software also event-based, i.e. whenever they want to obtain feedback or when a manager or the customer request it. In Rugby user feedback is an important source of elements for the backlog such as bug reports, design requests and feature requests. Rugby supports different kinds of issues that trigger the development. Depending on the issue type developers initiate a different workflow. Fig. 5 shows the feedback lifecycle[3] of Rugby in several usage scenarios.

Initially the user receives a potential product increment P0. While using it, he sends a couple of feedback items to the backlog. Each feedback item is categorized according to its type. The developers handle feature requests in the analysis workflow, design requests in the design workflow and bug reports in the implementation workflow. In the scenario shown in fig. 5, the first feedback includes two bug reports. The team resolved both bugs in the implementation workflow. After the implementation the developers commit their changes to the version control system and the build server automatically creates a new build including these changes.

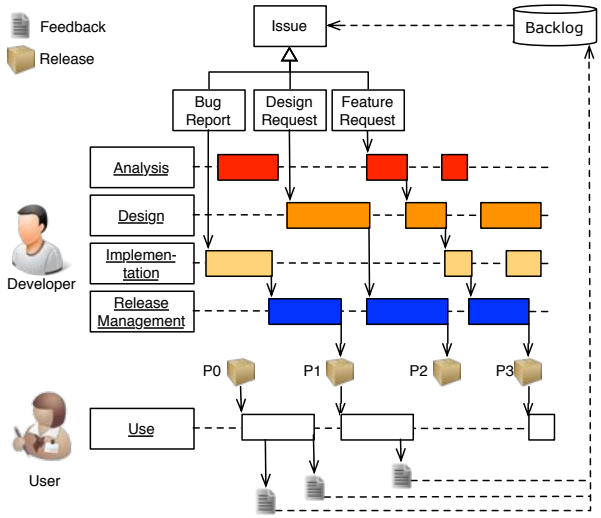

Figure 5: Rugby's Feedback Lifecycle

As part of the release management workflow, the release manager of the team decides whether and to whom the build should be delivered. In the scenario in fig. 5, the team decides to release and deliver this build as product increment P1. During a sprint, such a delivery is an event based release in Rugby. They can contain concrete questions of developers, e.g. in the release notes. Now the user can directly validate whether the reported feedback was resolved correctly.

P1 e.g. includes release notes about the two resolved bugs, so that the user can directly see that the team was able to resolve those. While using P1 the user finds another issue and produces another feedback request, which is categorized as a design request. This time the team handles the the feedback first in the design workflow, e.g. by adapting the system design of the software. After that it forwards the request to the implementation workflow so that the implementation can be adapted to the new design.. In Rugby multiple issues can be addressed in parallel. Some issues lead all the way to the release of a new product increment such as P1. Others lead only to internal releases such as P2. Some of the requests turn out to be not realizable and stay in the product backlog for a future sprint.

[2]Explorative projects usually have shorter sprints as requirements change more often and more feedback is required. Projects with mature requirements usually have longer sprints.

[3]Issues in the backlog have multiple sources like e.g. customer requirements. Fig. 5 focuses on the aspect how feedback is processed in Rugby and omits other sources.

4. EVENT-BASED DELIVERY IN RUGBY

In this section, we describe how Rugby uses continuous delivery to enable event-based releases. We first describe the deployment process to build and deliver new releases. We then explain the tools and continue with an overview of the workflow. We finally discuss how developers and managers use releases as models to facilitate communication between each other and with the customer. [4]

Humble models a deployment process as a stage-gate process, shown in fig. 6 [11]. During its lifecycle a build moves from the integration stage (1) through multiple testing stages (2) to a target environment (3). In each stage the build is checked against certain quality criteria. If these are fulfilled, the build is promoted to the next stage. Rugby calls a build which successfully went through all testing stages *releaseable*. A releasable build can be delivered to a target environment, e.g. production with no effort.

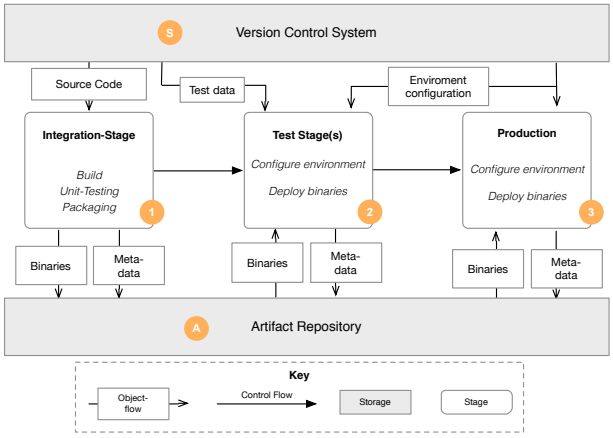

Figure 6: Rugby's Deployment Process (adapted from [11])

Rugby uses Humble's deployment process as a base for its continuous delivery workflow shown in fig. 7. To implement it, there is an *Issue Tracker* in the collaboration environment to manage the product backlog and a *Version Control Server* in the integration environment with support for branches to store source code and configuration data (S)[4].

To checkout, build, test and package the application there is a central *Continuous Integration Server* in the integration environment, which fulfills the role of the repository for build artifacts (A). To deliver a build to a target environment there is a separate *Delivery Server* in the delivery environment. The delivery server provides an easy to use solution for team members and users when downloading and installing a release into the target environment and allows users to give feedback to a certain release in a structured way.

Figure 7 shows the integrated continuous workflow together with its tools and transitions. The workflow starts each time a developer commits source code to the version control server, leading to a new build on the continuous integration server. If the build was built successfully and if it passed all test stages, the team can decide to upload it to the delivery server which then notifies users about a new

[4]Also compare fig. 2 for the different environments

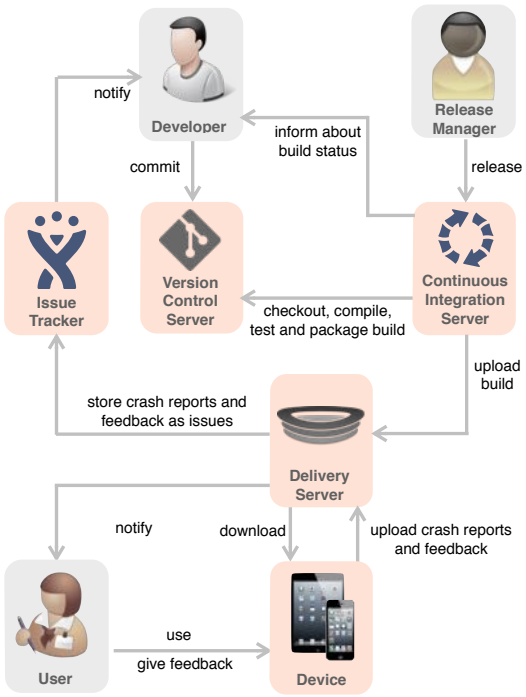

Figure 7: Rugby's Basic Continuous Delivery Workflow (adapted from [15])

release. Each release includes release notes, which are collected automatically by the continuous integration server[5] and can be edited in the manual release step if necessary.

The user can download the release and recognize easily, which features and bugs were resolved in the release. He can use an embedded mechanism to give feedback in a structured way. This feedback is collected on the delivery server and forwarded to the issue tracker which notifies the release manager about. For a detailed description of the workflow we refer to [15].

The workflow in fig. 7 only shows a limited amount of interactions of the developer with the version control server. In fact a developer has more possibilities than just committing source code. He can create branches to separate the work on a feature basis and merge these branches. Rugby uses a simplified version of the gitflow branching model [6] shown in fig. 8.

Developers use feature branches for the actual development work, a development branch for the integration of the feature branches and a master branch for time-based releases (e.g. at the end of the sprint) to the customer. Rugby's deployment process can automatically build and test all those three types of branches. For more details about the branching model we refer to [15]. Continuous delivery combined with this branching model helps a developer to automatically check if a new feature passes all test-stages and can be delivered to a target environment. It also enables developers to let users or customers validate the requirements of a feature, by producing an executable prototype as com-

[5]The continuous integration server asks the issue tracker to get all resolved issues since the last release. This connection is not shown in fig. 7 for simplicity reasons.

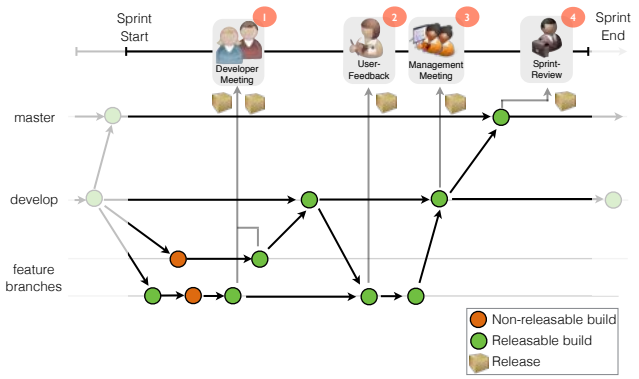

Figure 8: Rugby's Branching Model (adapted from [15])

munication model and sending it to the user or customer (event-based).

As described in section 3, Rugby proposes the use of executable prototypes created throughout the deployment process as communication models during the whole development process. The Rugby process model therefore allows developers to create releases from any branch as communication models. Fig. 8 also shows four different use cases of releases in Rugby. Releases from feature branches can be used in meetings to demonstrate the development status to all other team members (1).

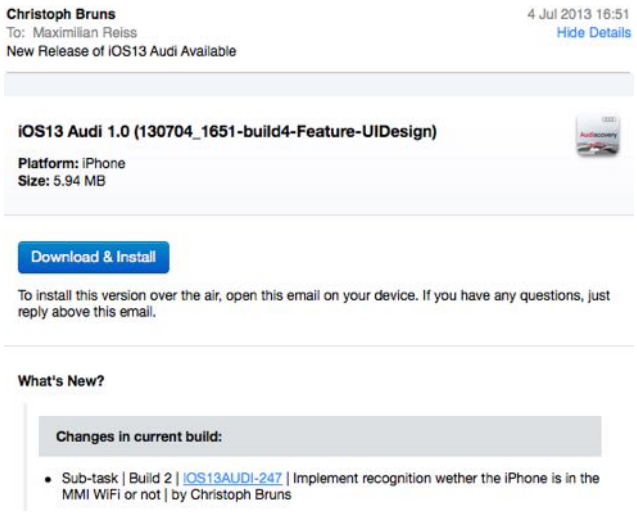

Figure 9: Notification Email for an event-based Release from a Feature Branch

This improves the quality of the communication in the team meeting and shortens the time that is required to explain specific implementation details. Releases from feature branches can also be used to obtain feedback from users to see whether a feature is usable and satisfies all user wishes (2). An example of an email sent to a user for an event-based release of a feature branch is shown in fig. 10. The release notes include one resolved task IOS13AUDI-247 that was implemented in the feature branch UIDesign. It is included as a link directly into the issue tracker, so that the user can see details within one click.

Releases of the development branch can e.g. be used for the status in management meetings (3), i.e. where all team leaders or all project leaders meet each other. This enables a lightweight coordination across teams for the program management because the current implementation status is always visible in form of executable prototypes. Builds from the master branch are used as time-based releases in the way Scrum uses product increments at the end of the Sprint. The team produces such releases for sprint review meetings (4) and does not need to create them manually.

An example of an email sent to a customer for a time-based release of the master branch is shown in fig. 9. The release notes include five resolved tasks, all of them including with a link into the issue tracker. One of these tasks was actually already delivered from a feature branch, but is again included because it was not included in the previous product increment (created from the master branch).

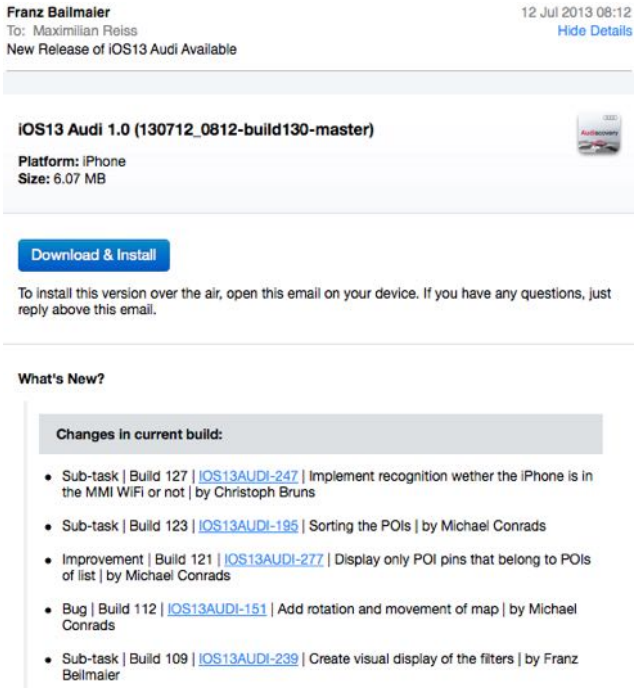

Figure 10: Notification Email for a time-based Release from the Master Branch

For each of the above mentioned events and meetings, a developer or manager can select a successful tested, i.e. releasable, build and deliver it to his own device for demonstration. The use of branches together with an automated deployment process increases the complexity of the interaction with the version control system, because developers now need to consider multiple branches and the according merge procedures and conflicts.

However the ability to use branching increases the flexibility for the developers, because they now have the possibility to create internal releases to test the software on their own devices and external releases just for specific features. Additionally, managers can use a development build to discuss the progress and current issues with other managers and the program management, using the same deployment process and amount of automation.

5. EVALUATION

We have used Rugby in two large capstone courses in university in 2012 [3] and 2013 [5]. In this section we describe the evaluation of Rugby in both courses. First we described the setup of the evaluation and discuss interesting observations from both courses. Then we show the results of an online questionnaires, a retrospective and personal interviews about Rugby and the use of release management and feedback with the participants of both courses. Finally we discuss the threats to validity.

5.1 Study Setup

Both capstone courses were multi-project courses with industry partners. In 2012 we had 11 projects and 100 participants, in 2013 we had 10 projects and 100 participants. The setup of these courses requires a high effort for the students as well as for the teaching assistants who prepare and organize them. However this commitment lets students experience real communication in a real project team with up to eight developers, a team leader and a project leader. Project leaders in these course were teaching assistants with more than two years project experience. They are responsible for the project success and the customer communication.

Team leaders were students who took the course in the year before as developer. They are responsible for the organization of team meetings, they control that the mostly inexperienced developers choose the right tasks and review the design as well as the implementation of the developers. A more detailed description about the course and its participants can be found in [4].

In 2012 and 2013, we evaluated Rugby in personal interviews with the release managers after the course. In 2013, we additionally conducted a retrospective with all project leaders and team leaders (see fig. 3) in an online questionnaire. We asked 14 questions about the organization of the course and the management meetings in general, the retrospective was completely anonymous.

In 2013, we furthermore conducted an online questionnaire about Rugby's release management and feedback workflows. We invited 90 participants, all developer and coaches of the course to participate in the online questionnaire. The questionnaire consisted of 16 questions and took about 15 minutes. The participants had four weeks after the final presentations in August 2013 to fill out the survey. We received 41 valid responses out of 90 participants. In 2012 and 2013, we also defined metrics to measure the effect of Rugby's release management and feedback workflows.

5.2 Observations

In 2012 the synchronization of the multiple projects on project management level took too much time because the project manager were not able to report their status precisely in the short amount of time. This was also the feedback we received from some of the project and team leaders in 2012. Often the participants didn't focus on crucial points and discussed issues too long which were not important for all meeting participants. This communication problem was intensified because of the multiplicity of problem statements and the different technical challenges in each of the projects. If a project leader talks about the status of his team without the ability to visualize it, the others can hardly follow and understand it.

In 2013, we introduced executable prototypes produced with Rugby's release management workflow as the central part of two management meetings, one with project leaders and one with the team leaders. The participants of these meetings were able to communicate their status of their team in significant less time. Furthermore the other participants of these two meetings were able to understand the status of the projects much better than in the meetings without executable prototypes.

We also asked the release managers to introduce this technique in the team meetings. From the personal interviews and the retrospective we conducted in 2013, we received a lot of positive feedback about this possibility. If the meeting participates prepare the executable prototype in advance, this technique saves a lot of time and improves the communication.

5.3 Results

In the online questionnaire about release management we evaluated whether the students performed specific tasks in the three different parts of the workflow (version control, continuous integration and continuous delivery) and whether they could obtain feedback with it. We summarize the utilization of these three parts in fig. 11. It shows that in each part more than half of the students performed at least one task and that version control tasks are applied most frequently.

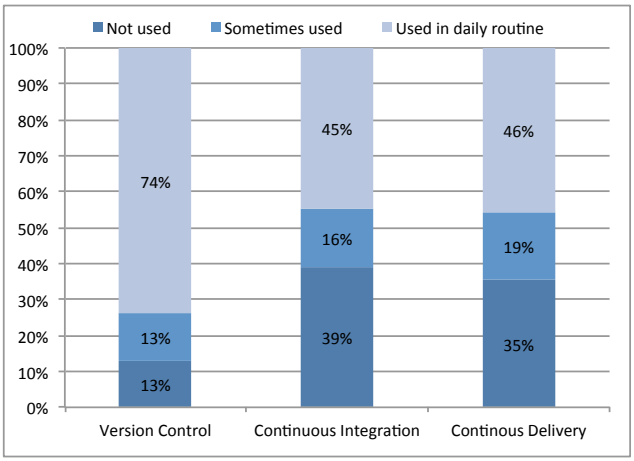

Figure 11: How often did you perform the task?

We also evaluated whether students see benefits in using version control, continuous integration and continuous delivery in Rugby. We wanted to know if e.g. branches help in developing and delivering multiple different prototypes at the same time. We therefore described three typical tasks for each topic and asked the participants if they see benefits using them.

Fig. 12 shows that the participants see the benefits in using a central version control system combined with a simple branching model. Over 80% of the team members think that the branching model helped to work with multiple persons on the same codebase. More than 40% of the developers used features branches to develop multiple prototypes for one functionality.

Additionally we asked if they think that continuous integration (CI) leads to faster error detection and correction

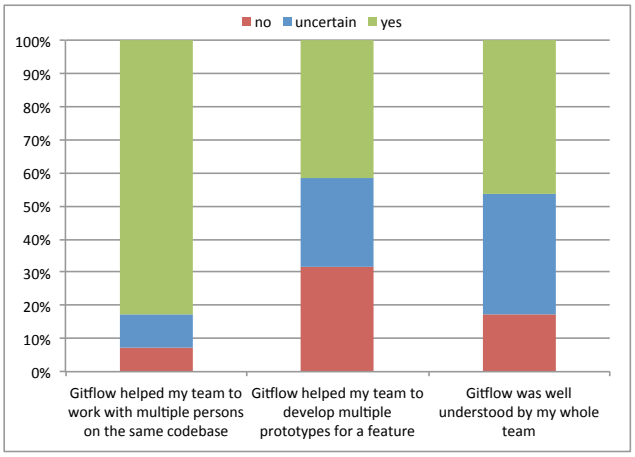

Figure 12: Do you agree to the following statements about Rugby's branching model?

and asked if continuous integration helped them to improved their development workflow. Fig. 13 shows that more than half of the team members see benefits when applying continuous integration.

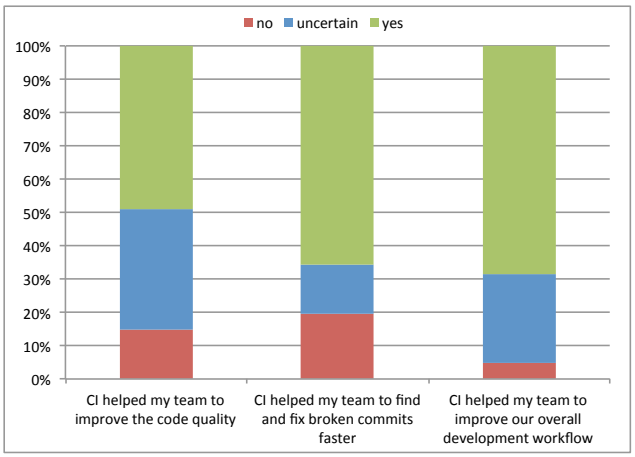

Figure 13: Do you agree to the following statements about Rugby's use of Continuous Integration (CI)?

Finally we asked the participants if they were able to collect more and better feedback from customers when applying continuous delivery (CD). Fig. 14 shows that 60% of the students think that the automated development process helped them to deliver more prototypes compared to a manual delivery process. About 40% think that they could obtain more and better feedback from their customers by applying continuous delivery.

We also used metrics to measure the effect of the release management and feedback workflows in Rugby. As shown in figure 15 and 16 the number of branches increased in 2013 when we first introduced the branching model. The number of commits in the version control system is almost the same, but 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.

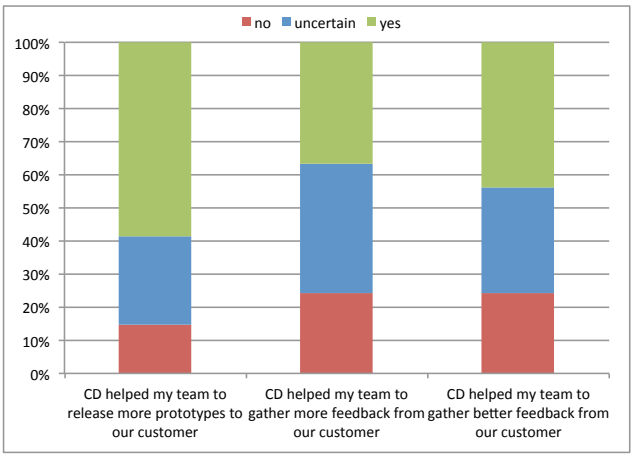

Figure 14: Do you agree to the following statements about about Rugby's use of Continuous Delivery (CD)?

This is caused by the fact that the build server was available from the first day in 2013 whereas in 2012 it was only available after two third of the project course. Most of the builds (in fact more than 94%) were successful in 2013, because the team leaders cared more about always having an executable prototype to present in a meeting. Consequently the absolute number of executable prototypes delivered to the customer is three times higher in 2013 because the teams were able to deliver releases from the first day.

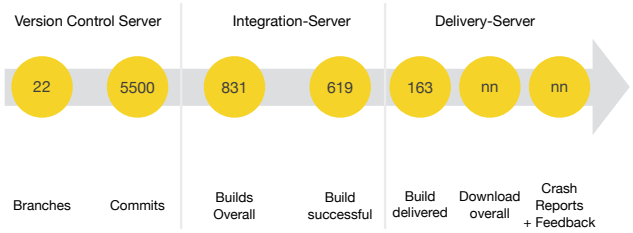

Figure 15: Statistics of the Multi-Project Course in 2012

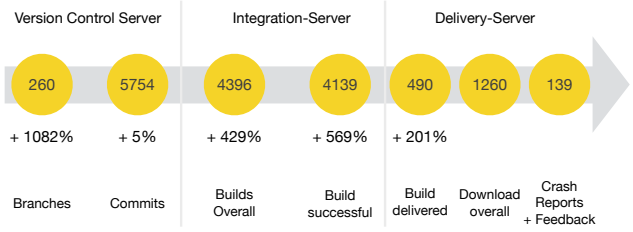

Figure 16: Statistics of the Multi-Project Course in 2013

In 2012 we did not measure the number of downloads, crash reports and feedback reports. However, in 2013 we can see that each delivered build was downloaded 2.5 times on average and that crash reports and the structured built-in feedback were used to create 139 reports.

5.4 Threats to Validity

We see the following threats to the validity in our evaluation. First, our findings apply to a multi-project software engineering course that was setup in university. While we believe that Rugby is applicable in innovation projects, the results might not be applicable in an industrial environment. We know that we cannot generalize our findings because of our setup in the university and the prior knowledge of the students. But we believe that some of the findings apply to software companies with project-based organizations in industry as well.

There might be the problem of selection bias in our online questionnaire, because we sent the invitation to all developers and team leaders of the project course and 41 out of 90 students responded. From the results and our observations we know that some teams used the release management and feedback workflows more than others. This was due to 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 significant 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 assuming that this would give them a good grade. We addressed this threat by making the survey completely anonymous.

6. CONCLUSION

We have established an agile process model Rugby that is based on continuous delivery and event-based releases. Rugby adds two additional workflows to the lifecycle model, release management and feedback. It focuses on rapid delivery and increases the number of releases as we have shown in our evaluation.

Rugby is lightweight and improves the coordination across multiple teams as well as the communication between developers and customers. The use of executable prototypes as communication models reduces the time spent for status reports and discussion and helps in the requirements elicitation. The inclusion of multiple feedback cycles allows developers to respond to user feedback in a structured with release notes to notify users about changes in the updated release.

We like to further investigate Rugby in an industrial project where the circumstances are even more realistic than in our capstone course in university. We plan to accompany a first project using Rubgy in industry in 2014.

7. ACKNOWLEDGMENTS

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

8. REFERENCES

[1] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, et al. Manifesto for agile software development. *The Agile Alliance*, 2001.

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

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

[4] 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.

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

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

[7] M. Fowler. The new methodology. *Wuhan University Journal of Natural Sciences*, 6(1-2):12–24, 2001.

[8] P. Gfader. Use scrum and continuous delivery to build the right thing, 2013. `https://www.scrum.org/Portals/0/Documents/Community%20Work/Scrum.org%20Whitepaper_Continuous%20Delivery.pdf`.

[9] E. Hossain, M. A. Babar, and H.-y. Paik. Using scrum in global software development: A systematic literature review. In *ICGSE'09*, pages 175–184. IEEE, 2009.

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

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

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

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

[14] P. Kruchten. *The rational unified process: an introduction*. Addison-Wesley Professional, 2004.

[15] S. Krusche and L. Alperowitz. Introduction of Continuous Delivery in Multi-Customer Project Courses. In *Proceedings of ICSE'14*. IEEE, 2014.

[16] W. Maalej, H.-J. Happel, and A. Rashid. When users become collaborators: towards continuous and context-aware user input. In *OOPSLA'09*, 2009.

[17] M. Paasivaara, S. Durasiewicz, and C. Lassenius. Using scrum in distributed agile development: A multiple case study. In *ICGSE'09*, pages 195–204. IEEE, 2009.

[18] D. Pagano and B. Bruegge. User involvement in software evolution practice: a case study. In *Proceedings of ICSE'13*, pages 953–962. IEEE, 2013.

[19] R. B. Rowen. Software project management under incomplete and ambiguous specifications. *Engineering Management, IEEE Transactions on*, 37(1):10–21, 1990.

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

[21] H. Takeuchi and I. Nonaka. The new new product development game. *Harvard business review*, 64(1):137–146, 1986.