Experience Report: A Comparison between Commercial and Open Source Reference Implementations for the Rugby Process Model

Sajjad Taheritanjani, Stephan Krusche, Bernd Bruegge

Technische Universität München, Germany
sajjad.taheri@tum.de, krusche@in.tum.de, bruegge@in.tum.de

Abstract: Rugby is a process model for continuous software engineering which allows developers to continuously deliver prototypes and obtain feedback supporting software evolution. There is a reference implementation of Rugby with commercial enterprise tools used in university capstone courses. However, since these tools are expensive, there is a need to study less expensive alternatives which are available on the market to evaluate whether they can be used in Rugby. In this paper, we compare a second reference implementation with the existing one, focusing on the core use cases and non-functional requirements of Rugby.

1 Introduction

Increasingly dynamic environments lead to shorter development cycles [FS14]. Continuous software engineering (CSE) refers to the organizational capability to develop, release, and learn from software in rapid cycles [Bo14] providing the capability for software evolution [HF10]. Rugby is a process model that proposes a development lifecycle for CSE [KAB+14]. It combines elements of Scrum [Sc04] and the Unified Process [JBR+99] with additional workflows to support CSE and software evolution: review management, release management and feedback management [KAB+14]. Developers work in a project-based organization with multiple projects to deliver executable prototypes and to obtain feedback. Bruegge et al. developed a reference implementation of Rugby that is applied in university capstone courses using commercial enterprise tools: JIRA, Bitbucket Server, Bamboo and HockeyApp [BKA15].

However, Rugby is not limited to these commercial tools. Expensive tools cannot easily be used by individuals and organizations, e.g. open source projects or young startups. The existing reference implementation also poses challenges, e.g. in terms of usability, because the tools have a high learning curve. Therefore, we investigate Rugby's use cases for its three additional workflows and one Scrum core workflows: issue management. We conducted a research [TKB15] which shows for each of these workflows, there are many popular options available: Bugzilla, Redmine, Trac, GitHub Issues and Mantis as the Issue Tracker, GitHub, SourceForge, GitLab, Klin, CodePlex and Codeplan as the version control system (VCS), Jenkins, TeamCity, Travis, Hudson, Wercker, Team Foundation Server and GitLab CI as the continuous integration (CI) server, and Appaloosa, Crashlytics, GoCD and HockeyKit as the continuous delivery (CD). Considering the tight collaboration between these categories in Rugby,

we decided to choose the GitHub Issues as the Issue Tracker, GitHub as the VCS and Travis as the CI solution, all from GitHub to ensure their efficient integration. We also chose Jenkins as the CI server, because it is open source and used in many projects. Both, GitHub and Travis are cloud-based platforms, that can be used for free in open source projects. Since there is no open source alternative for CD in mobile platforms, we chose Crashlytics, which made all its services free after its acquisition by Twitter. As knowledge management using a Wiki is not included in Rugby's continuous workflow (see section 2), we decided to exclude it from our comparison list.

In this paper, we created example projects with these tools to be able to compare them in each category [TKB15] with respect to the most important Rugby use cases for developers, managers, users and also with regards to configurability and flexibility. At the end, using GitHub tools, Jenkins and Crashlytics, which all target on the open source projects, we create a second reference implementation and use it in different example projects to compare and evaluate the differences of how the main use cases are implemented. We also give recommendations about which reference implementation can be used.

2 Rugby Process Model

Rugby is a process model that includes workflows for CD. It allows part timers to work in a project-based organization with multiple projects for the rapid delivery of prototypes and products. Using CD improves the development process in two ways: First, Rugby improves the interaction between developers and customers with a continuous feedback mechanism. Second, Rugby improves coordination and communication with stakeholders and across multiple teams in project-based organizations with event based releases [KAB+14]. Figure 1 shows the integrated continuous workflow with abstract tools and transitions.

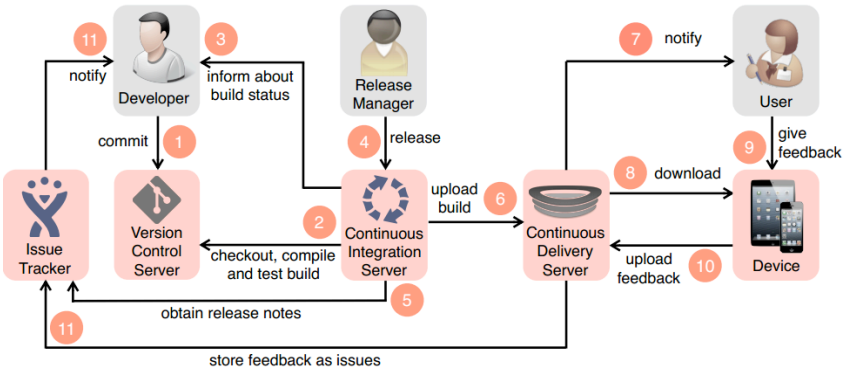

Figure 1: Rugby's continuous workflow with abstract tools and transitions [KAB+14]

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 release. Each release includes re-

lease notes, which are collected automatically by the continuous integration server 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 that. 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 [KA14].
Rugby is a customizable process model that can be adapted and refined to the team's needs [KAB+14]. It supports CSE and software evolution with one Scrum core workflow together with three additional workflows: (1) issue management needs an issue tracker, (2) review management needs a VCS, (3) release management needs a CI and a CD server, (4) feedback management needs a CD server and an issue tracker [KAB+14]. These workflows are also customizable and can be included in other agile processes, in particular Scrum and Kanban [KAB+14].

3 Comparison between Different Tools in each Workflow

For each of these workflows, we select alternatives and compare them with the corresponding commercial tools in the existing reference implementation with respect to the most important core use cases and non-functional requirements in each workflow. At the end of each section, there is a comparison table which shows scores for each feature of the tool (combination). Scores range from 1 (very poor quality or support) to 5 (excellent quality or support). In case a tool does not support a feature, a dash sign is shown (-).

3.1 Issue Management Workflow

Issue tracking is a software tool that enables the developers to record and track the status of all issues associated with each configuration object in the project [Pr05]. In this section, we compare JIRA by Atlassian and GitHub Issues for issue management.

JIRA has a customizable workflow for issues which makes it easy to tailor based on different needs in each project. The issue workflow consists of statuses and transitions that an issue goes through during its lifecycle modeled as a finite state automaton. The workflow can be edited or a new one can be created from scratch. JIRA supports sprint planning and sprint reviewing as defined in Scrum.

On the other hand, GitHub Issues, does not have customizable workflows for issues. However, a list of milestones can be configured with a corresponding due date. A milestone shows the last edit date, percentage of completed tasks, the number of open and closed issues and merge requests, which can give a general overview of the project progress. With some configurations, GitHub Issues can support agile project management. Using milestones and labels, it is possible to simulate sprint backlogs and versions which are not available by default [Ke15].

Therefore, the main advantages of JIRA in comparison with GitHub Issues are its customizable workflows for issues and support for variety of issue types as well as different reports and charts. GitHub Issues main advantage is its usability which is very simple and easy to work in comparison with JIRA.

Relevant features for Rugby	JIRA	GitHub
Customizable issue states and transitions	5	1
Configuration possibilities	5	1
Taskboard support	5	3
Versions support	5	4 (using Labels)
Backlogs support	5	3 (using Milestones)
Sprint planning/review support	5	3
Usability	3	5

Table 1: Issue management workflow comparison [TKB15]

3.2 Review Management Workflow

VCS is a system that keeps track of files and their history and have a model for concurrent access [Ot09]. In this section, we compare Bitbucket Server, formerly known as Stash, by Atlassian and GitHub for review management.

Both Bitbucket and GitHub use Git as a distributed VCS to control the source code versions. Features like merge requests (a.k.a. pull request), branch management and merge management are the most important functions [CS14]. Therefore, Bitbucket and GitHub function similarly in their core features. Bitbucket only supports sharing code files as snippets (known as Gists) with additional plugins. Its web interface lacks inline editing unless additional plugins are used.

In contrast to GitHub, Bitbucket's web interface does not show tags. However, this does not affect its functionalities, since it supports all the GitHub's features without needing to use labels. In terms of usability, GitHub is slightly better than Bitbucket. It is simple and everything is easy to find in the menus. One important difference between Bitbucket and GitHub is that they use different mechanism to show compare view (*diff*) [Pe15]. Although Bitbucket's approach is more complex and has an overhead, it provides more accurate and useful merge request diff.

Relevant features for Rugby	Bitbucket Server	GitHub
Merge requests	5	5
Inline code commenting	5	4
Web inline editing	4 (using plugins)	5
Sharing code files and snippets support	4 (using plugins)	5
Commit comments	5	5
Accurate compare views	5	4
Branch/merge management	5	5
Collaborative code review	5	5
Usability	4	5

Table 2: Review management workflow comparison [TKB15]

3.3 Release Management Workflow

CI is a software development practice where developers integrate their work frequently, which leads to multiple integrations per day. Each integration is verified by an automated build, including test, to detect integration errors as quickly as possible [FF06].

In other words, CI is the process of automatically building and running tests whenever a change is committed. On the other hand, CD is a software engineering approach in which teams keep producing software in short cycles and ensure that the software can be reliably released at any time [Ch15]. Rugby uses an enterprise app store as the CD server, a web portal through which end users can access, download, install approved software applications, send feedback about the application and report crashes. In this section, we compare three combinations of CI and CD, Bamboo and HockeyApp, Travis and Crashlytics, and Jenkins and Crashlytics.

Travis is a maintenance free CI server, since it is a hosted service and all the installations and updates are performed server side. Although Jenkins can be a hosted service as well, it is usually use on premise, needing administration effort for installations and updates. Bamboo can be used as a cloud service or as an on premise solution. Jenkins does not use a database for storage which makes it flexible and portable. Only by copying the configuration, Jenkins jobs can be easily migrated across multiple instances which is not possible in Bamboo. Therefore, maintainability in Jenkins is easier.

Additional build agents are needed to scale CI servers horizontally. Bamboo licenses are priced per agents that need to be installed and configured. In Jenkins, unlimited amount of build agents is available and configuration of build agents is very easy. The build agents are configured via Jenkins UI and all of the tools can be installed automatically. Therefore, Jenkins is easier to scale than Bamboo.

A deployment project in Bamboo is a container for holding the software project which is deployed. Besides, plan branches represent a build for a branch in the version control system. When the plan branch build succeeds, it can be automatically or manually merged back into master. Bamboo deployments allow a plan branch to be deployed to a test environment and the feature source code can be tested and evaluated in a real server environment before the code is merged back to master. In Jenkins, there are available plugins to support deployment and branches as well, while in Travis they are supported by Travis' configuration file. Concluding, all tools support deployment and branches. However, in Bamboo, when build jobs call deployments, it is not possible to go back to the build job to perform some post-deployment tasks. In addition, inability to provide input parameters, especially for deployment jobs, is a problem in Bamboo as well.

Each release includes release notes, which describe features and resolved bugs. The release notes are collected by the CI server and can be edited in the manual release step if necessary. Non of the CI solutions can create release notes automatically.

Code testing is the process of automatically building and running tests whenever a change is committed. Code integration is the process of automatically integrating the code after a committed change. All three CI tools support testing and integration.

It is not possible to get crash logs in Travis, without setting up scripts to upload them to a third-party service after completion of a build. In Bamboo and Jenkins, there are a lot of useful information regarding crash logs in test output display. Bamboo has an easy to use UI, Travis' UI is even more simple. However, Jenkins' UI looks out of date. Both HockeyApp and Crashlytics, notify the users about new releases and support feedback notification as well. They enable the CI server to automatically upload a new build and support the download of releases, new versions, push notifications and publication configurations.

Relevant features for Rugby	Bamboo + HockeyApp	Travis + Crashlytics	Jenkins + Crashlytics
Build plan configuration	5	4	4
New commit/branch detection	5	4	4
Release notes creation	-	-	-
Testing	5	5	5
Integration	5	5	5
Build status notification	5	2	5
Deploy build to CD server	4	5	5
Feedback/new release notification	5	5	5
App version automatic upload	5	5	5
Release download	5	4	4
CI scalability	3	-	5
CI maintainability	3	5	4
CI usability	4	5	2
App version download usability for the user	5	4	4
Support of multiple mobile platforms	5	4 (iOS, Android)	4 (iOS, Android)

Table 3: Release management workflow comparison [TKB15]

3.4 Feedback Management Workflow

In Section 3.3, we mentioned that Rugby uses an enterprise app store as the CD server. In this section we compare the combinations of HockeyApp and JIRA, and Crashlytics and GitHub Issues. Both Crashlytics and HockeyApp store crash reports and errors as issues. However, since GitHub Issues does not support multiple issue types, it is not possible to convert issues to appropriate work items like bugs or improvements.

In Crashlytics, different user groups can be defined and different users can be added to them. HockeyApp supports five user roles: owner, manager, developer, member and tester. HockeyApp supports more feedback management features in comparison with Crashlytics; while the user can reply to the developer's questions and his feedback records usage context, developer can pull them and reply to them.

Therefore, the main differences between HockeyApp and Crashlytics are the multiple mobile platforms support, user device registration, developer device management and voting mechanism to prioritize feedback requests. While HockeyApp supports different platforms, Crashlytics is only limited to iOS and Android. Although both of these tools have a good usability, HockeyApp is more simple and easier to use. While users can register their iOS and Android devices in HockeyApp, they can do so only for the iOS devices in Crashlytics. HockeyApp also supports developer device management which is not supported in Crashlytics.

Crashlytics takes into account that often a crash occurs and assigns it an impact level. It notifies when a specific crash is more critical than another one. As a particular crash is reported more and more, Crashlytics tracks that information and calls out the crashes that should be dealt with next. On the other hand, HockeyApp provides more depth and accuracy of crash logs. However, it does require more setup compared to Crashlytics.

Relevant features for Rugby	**HockeyApp + JIRA**	**Crashlytics + GitHub Issues**
User device registration	4 (iOS and Android)	3 (iOS)
User feedback upload	5	5
User reply to developer question	5	5
User attach media	5	5
Developer device management	5	3
Developer feedback pull	5	5
Voting mechanism	5	-
Usage context record	5	5
Feedbacks and analytics for developer	5	5
Store crash reports and errors as issues	5	5
Issue conversion to appropriate work item (issue)	5	1

Table 4: Feedback Management Workflow Comparison [TKB15]

4 Conclusion

In this paper, we create a second reference implementation with tools that can be used for free in open source projects, using GitHub tools, Jenkins and Crashlytics, and compare it with the commercial reference implementation. The second reference uses GitHub Issues as the Issue Tracking, GitHub as the VCS, and Crashlytics as the CD server. For CI, there are two options to choose: Travis and Jenkins. Both of these CI servers are candidates to become the CI solution. Cloud vs On Premise of the code repository is the most important factor in choosing, followed by project type. If the project is open source, Travis would be the better option, because there is no setup time and fee. If it is a company project with privacy concerns, Jenkins is a better option, because the setup time is minimal and maintaining the server is rather simple.

Since JIRA, Bitbucket and Bamboo are all Atlassian tools, they can integrate seamlessly using minimal effort and have a high overall usability as well, while the learning curve might be high for new users. On the other hand, GitHub Issues, GitHub and Travis are from GitHub and they also integrate with each other. GitHub tools are simpler and have fewer features than Atlassian tools, so they are easier for new users and can be used for free in open source projects. In addition, maintenance is not an issue while working with GitHub tools, because they are all hosted in the cloud.

Considering the price plan of different tools, the proposed reference implementation using open source tools can be used for free, especially for small startups that work on public repositories. However, if private repositories and concurrent jobs in CI are necessary, the price varies and there is not too much difference between the two reference implementations in the terms of price. Using the commercial reference implementation for middle sized companies, with around 100 persons, can be expensive. For such middle sized companies, the open source reference implementation cost is about half of the commercial one.

Therefore, both the existing commercial reference implementation, using JIRA, Bitbucket, Bamboo and HockeyApp, and the proposed open source implementation, using GitHub Issues, GitHub, Travis/Jenkins and Crashlytics, cover most of Rugby's important use cases. The commercial solution is a better choice when security and privacy

are important and repositories should be private. On the other hand, when the project can be public, the open source solution should be preferred. Rugby can be implemented with different tools, either for commercial projects or open source projects. However, there is always a tradeoff between the tools quality and their price. It remains to compare other tools for their use in Rugby projects in the future works.

References

[FS14] B. Fitzgerald, K.J. Stol: "Continuous software engineering and beyond: trends and challenges." Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering. ACM, 2014.

[Bo14] J. Bosch: "Continuous software engineering: An introduction," in Continuous Software Engineering. Springer, 2014; Pages 3–13.

[Sc04] K. Schwaber: Agile project management with Scrum. Microsoft Press, 2004; Chapter 4.

[JBR+99] I. Jacobson, G. Booch, J. Rumbaugh, J. Rumbaugh, G. Booch: The unified software development process. Addison-wesley, 1999; Page 92.

[HF10] J. Humble, D. Farley: Continuous delivery: reliable software releases through build, test, and deployment automation. Pearson Education, 2010; Pages 24-29.

[KAB+14] S. Krusche, L. Alperowitz, B. Bruegge, M. O. Wagner, Rugby: An Agile Process Model Based on Continuous Delivery, Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering. ACM, 2014.

[KKP+15] S. Klepper, S. Krusche, S. Peters, B. Bruegge, L. Alperowitz: Introducing Continuous Delivery of Mobile Apps in a Corporate Environment: A Case Study, 2015.

[BKA15] B. Bruegge, S. Krusche, L. Alperowitz: Software Engineering Project Courses with Industrial Clients, ACM Transactions on Computing Education, 2015.

[Pr05] R.S. Pressman: Software engineering: a practitioner's approach (7th Edition). Palgrave Macmillan, 2010; Page 595.

[KA14] S. Krusche, L. Alperowitz: Introduction of Continuous Delivery in Multi-Customer Project Courses, Proceedings of the 36th International Conference on Software Engineering, 2014; Pages 2-3.

[Ke15] H. Kellaway: Using Github for Lightweight Software Project Management, 2015. Retrieved 06-October-2015 from http://harlankellaway.com/blog/2015/04/02/using-github-issues-for-software-project-management/

[Ot09] S. Otte.: Version Control Systems. Computer Systems and Telematics, 2009 Institute of Computer Science, Freie Universität, Berlin, Germany.

[CS14] S. Chacon, B. Straub: Pro Git (2nd Edition). Apress, 2014; Pages 89-98.

[Pe15] T. Petterson: A better pull request, 2015. Retrieved 13-September-2015 from https://developer.atlassian.com/blog/2015/01/a-better-pull-request/

[FF06] M. Fowler, M. Foemmel: Continuous integration. Thought-Works) http://www.thoughtworks.com/ContinuousIntegration.Pdf, 2006.

[Ch15] L. Chen: Continuous Delivery: Huge Benefits, but Challenges Too. Software, IEEE, 2015.

[TKB15] S. Taheritanjani, S. Krusche, B. Bruegge: A Comparison between Commercial and Open Source Reference Implementations for the Rugby Process Model, A University Research Report, 2016.