Introducing Continuous Delivery of Mobile Apps in a Corporate Environment: A Case Study

Sebastian Klepper
TU München
Munich, Germany
sebastian.klepper@tum.de

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

Sebastian Peters
Capgemini
Munich, Germany
sebastian.peters@capgemini.com

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

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

***Abstract*—Software development is conducted in increasingly dynamic business environments. Organizations need the capability to develop, release and learn from software in rapid parallel cycles. The abilities to continuously deliver software, to involve users, and to collect and prioritize their feedback are necessary for software evolution. In 2014, we introduced Rugby, an agile process model with workflows for continuous delivery and feedback management, and evaluated it in university projects together with industrial clients.**

Based on Rugby's release management workflow we identified the specific needs for project-based organizations developing mobile applications. Varying characteristics and restrictions in projects teams in corporate environments impact both process and infrastructure. We found that applicability and acceptance of continuous delivery in industry depend on its adaptability. To address issues in industrial projects with respect to delivery process, infrastructure, neglected testing and continuity, we extended Rugby's workflow and made it tailorable.

Eight projects at Capgemini, a global provider of consulting, technology and outsourcing services, applied a tailored version of the workflow. The evaluation of these projects shows anecdotal evidence that the application of the workflow significantly reduces the time required to build and deliver mobile applications in industrial projects, while at the same time increasing the number of builds and internal deliveries for feedback.

***Index Terms*—Release Management, Configuration Management, Continuous Integration, Continuous Delivery, User Feedback, User Involvement, Agile Methods, Software Evolution**

I. Introduction

Software development is conducted in increasingly dynamic business environments. Fast-changing markets, complex and changing customer requirements, pressure of shorter time-to-market, and rapidly advancing information technologies are characteristics found in most mobile software development projects. Fitzgerald et al. found that 78 % of their interviewed industry managers and executives think achieving digital transformation is critical to their organizations, while 63 % said their pace of technology change is too slow. [1]

One digital transformation happens in the sector of mobile applications. Their usage increased by 113 % in 2013 and another 76 % in 2014 and they increasingly involve critical aspects of businesses [2]. Examples like flexible car-sharing services, apartment sublets and others show that entire business models are now based on mobile apps. User feedback is essential in the development of mobile applications because usability and user experience play an important role [3]. Users increasingly review mobile applications stating useful comments, bug reports, their personal experience, and feature requests [4]. User involvement can help developers to understand their needs, when their feedback is systematically obtained and incorporated into the development process [5].

To deal with digital transformations, agile methods like Scrum [6] advocate flexibility, efficiency and speed. Many software companies succeeded in incorporating agile practices into their development workflow [7]. At the same time other functions inside those organizations (e.g. customer relations, product management, and software releases) still follow traditional cycles measured in months or years instead of introducing an agile approach as e.g. described by Pichler [8].

Continuous software engineering refers to the organizational capability to develop, release, and learn from software in rapid parallel cycles [9]. It includes delivering software to users, collecting and prioritizing their feedback, and incorporating it into the next development cycle, leading to software evolution. The capability to perform these activities in short cycles requires the incorporation of a release management workflow that automates the delivery of software changes to users [10].

In a project-based organization, stovepipes hinder an efficient implementation of continuous delivery because the structure of the organization restricts the flow of information, inhibiting or preventing cross-organizational communication [11]. If there is no direct communication between teams, knowledge silos build up and teams end up reinventing the wheel, from activities of the release management workflow to underlying software and even hardware infrastructure [12]. Even if the software organization takes measures to avoid forming silos – e.g. centralizing IT to standardize infrastructure while establishing cross-functional teams to carry knowledge between projects – they can still remain due to constraints of the client organization.

Applying agile methodologies in software development already requires a culture of openness among the project members and a supportive management style. However, if project teams are not able to adapt their release management workflow to their specific needs and constraints, they might not use it at all. This is why we introduce the concept of tailoring [13] to Rugby's release management workflow [14]. Certain activities of the workflow are optional and the project team can freely choose whether or not to employ them. Additionally, each activity can be adapted to the project's needs: how an activity is carried out (e.g. manually or automatically) can vary across projects, but the overall workflow and its benefits remain. [15] Still, these building blocks form a turnkey solution that enables projects to get started quickly. Compared to a manual setup of workflow and infrastructure, time and effort is saved and more projects are enabled.

This paper is organized as follows. We describe Rugby's initial release management workflow in Section 2 and analyze its applicability in eight industrial projects at Capgemini[1] in Section 3. Additional requirements found in personal interviews with project managers lead to an extended workflow focused on tailoring that we describe in Section 4. We evaluated this workflow and its impact on industry projects and present our findings in Section 5.

II. Rugby's Release Management Workflow

In 2014, we introduced Rugby [14], a lightweight process model that includes concepts of Scrum [6] and the Unified Process [16]. In Rugby, self-organizing teams develop software in project-based organizations using the concept of sprints following Scrum. In particular, Rugby includes release management and user feedback as additional workflows in the software lifecycle model and allows developers to release software event-based to users when they require feedback. Rugby focuses on innovation projects where problem statements are formulated as visionary scenarios and where requirements and technologies can change during the project [17]. Requirements can be further discussed and negotiated within the sprint. [14]

Fig. 1 shows Rugby's release management workflow. It incorporates version control, continuous integration (CI), continuous delivery (CD), and a feedback mechanism. The workflow starts each time a developer pushes source code to the version control server (step 1). This leads to a new build on the CI server that notifies the developer about the build status, e.g. via email or chat (step 2 and 3). If the build was successful and after it passed all test stages, the release manager can release the build on the CI Server which uploads the build to the CD Server. Users are automatically notified about the availability of the new release so that they can download it onto their device (step 4 - 6). Within the application, they can give feedback which is uploaded to the CD server and forwarded to the issue tracker notifying the developer (step 7 - 11). This workflow allows practices like "release early, release often", well established in open source software development [18], resulting in continual improvement [19], [20].

Rugby focuses on the rapid, event-based delivery of mobile applications and increases the number of releases and feedback reports. It 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 eliciting additional or

[1]Capgemini provides consulting, technology, outsourcing services and local professional services. It is presented in over 40 countries with almost 140,000 employees. http://www.capgemini.com

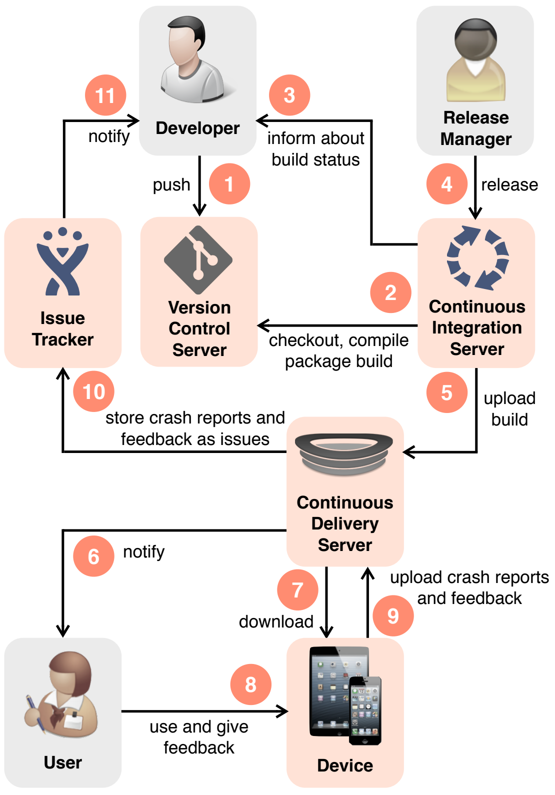

Developer
Release Manager
11
notify
3
inform about build status
push
1
4
release
Issue Tracker
Version Control Server
2
checkout, compile package build
Continuous Integration Server
10
store crash reports and feedback as issues
5
upload build
Continuous Delivery Server
6
notify
upload crash reports and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

Fig. 1. Release management workflow used in Rugby (adapted from [21])

modified requirements during the project. The inclusion of multiple feedback cycles allows developers to respond to user feedback in a structured way with release notes to notify users about changes in the updated release. We successfully introduced and evaluated Rugby in a university setting with industry clients. [14] In the next section we discuss how Rugby can be extended for the use in real corporate environments.

III. Applicability in Industrial Projects

We analyzed the development process for mobile applications in a global company with heterogeneous project environments with respect to team size and project duration. We interviewed eight project managers of Capgemini who develop software solutions for external customers in different industry sectors such as automotive, telecommunication and financial services. The answers of the interviews revealed the following *key issues* in the investigated projects:

There is *no standardized delivery process* for mobile applications. Projects have to rely on knowledge transfer from other projects. Depending on size and complexity, existing approaches might not fit. Therefore the same solutions are reinvented by multiple teams because of the time constraints within mobile projects and limited communication.

The *infrastructure is not sufficient* for the delivery of mobile applications. It contains the basic development tools such as an issue tracker, a version control repository and an integration server, but these tools are not pre-configured for mobile projects and the development team is responsible for the configuration. A solution for automatic delivery is missing, some projects decide to create their own one.

Automatic testing is neglected in favor of development speed, mainly because the setup effort is too high in mobile projects, which are typically rather short-living. This especially affects unit, system and integration tests. Acceptance testing with the customer is done manually, but not regularly.

Continuity is missing in the development. Even if projects write tests, they rarely automate them. Continuous integration is only taken seriously by very few mobile projects. Even if projects use a build server and a delivery solution, integration and delivery are still done sporadically instead of continuously. How often a version is delivered to the customer is sometimes rather driven by the contract and not by development needs.

We asked the project managers why these issues occurred and why they do not yet apply a standardized workflow for continuous integration and continuous delivery as e.g. defined in Rugby. From their answers, we extracted the following *additional requirements* that need to be addressed by project-based organizations like Capgemini that consider to apply continuous delivery in their mobile projects:

Mobile applications are targeting different platforms, some of them are developed natively, but in different programming languages, others using cross-platform frameworks. The continuous delivery workflow should therefore *support multiple platforms*. Business critical applications require a high amount of security. Customer contracts e.g. do not allow to store applications on cloud-services. *Access control* and *data privacy* are important topics and should be considered. Due to pricing pressure, projects are outsourced to offshore locations. It should be possible to apply activities of the workflow in different countries for *globally distributed teams*.

Applications are developed for multiple markets, thus different *legal aspects* regarding the distribution of applications need to be considered. Different project environments should be supported by *modular standardized components*, that can be easily adapted and maintained. Managers should be able to *collect metrics* about the current state of the project. Larger mobile application projects require *dependency management* for the use of external and internal frameworks. The delivery and feedback system should *distinguish* between automatic and user-generated *feedback*. Furthermore, it should be possible to *deliver* an application manually.

In a company with heterogeneous project environments it is not possible to introduce one single, out-of-the-box workflow for all projects. Instead, project managers must be able to tailor the workflow to their own needs. To address the additional requirements, we extended Rugby's release management workflow. We aim to provide flexible activities as building blocks that can be combined depending on the individual characteristics of a given project.

IV. Extending and Tailoring Rugby's Workflow

Using Rugby's release management workflow as basis, we split it up into the activities configuration management (including version control and dependency management), continuous integration, continuous delivery and feedback, that may be optional or provide variations. We modified and extended each activity to increase both functionality and flexibility. The resulting tailored workflow is presented in Fig. 2. Its major components are the activities *configuration management*, *integration*, *delivery* and *feedback*. Scope and transitions of activities are described in detail to provide projects with multiple variants of how each activity can be carried out (see Fig. 2a).

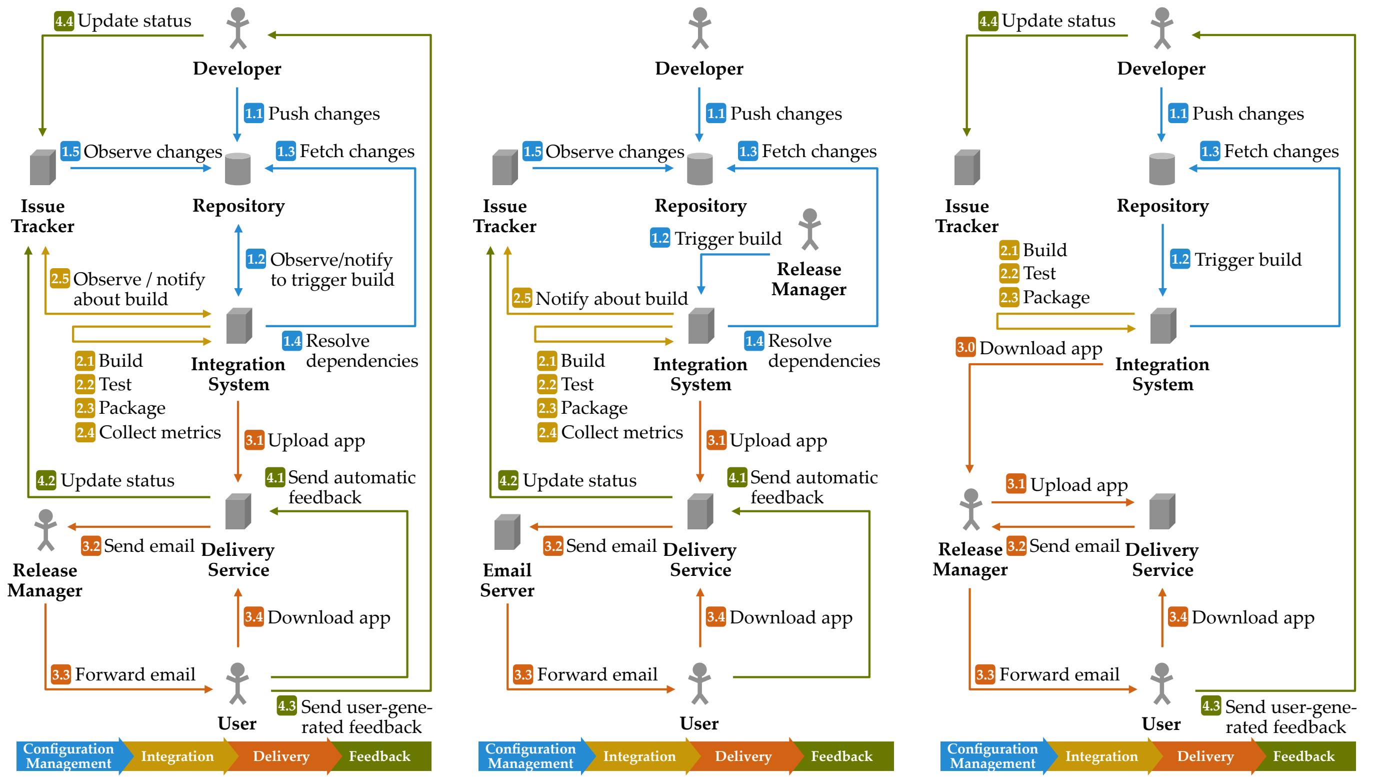
The extended workflow provides tailoring by the possibility to distinguish between mandatory and optional activities. In particular, projects can modify the workflow depending on project size, complexity, staffing, timeline and priorities. For a better understanding of how a tailored version of our workflow could be instantiated, we describe two examples that we have observed at Capgemini:

Large and complex project (see Fig. 2b): Version control uses an extensive branching model in combination with a full-fledged dependency management system (DMS). Builds are triggered manually since the integration system has to resolve dependencies, build a hybrid core app first and then include native wrappers for several platforms. It runs unit and system tests, and also integration tests with a backend API as well as automated acceptance tests for the user interface. Upload to the delivery service occurs automatically and emails are auto-forwarded using lists and filters managed on an email server. Feedback is solely collected automatically from within the app usage and relayed to the issue tracker, which also keeps track of changes and builds.

Small and simple project (see Fig. 2c): Version control uses a compact branching model, but dependencies are kept in the repository instead of a full-fledged DMS. The project has to build a single native app and uses an integration system that automatically fetches changes and runs a few unit and system tests. Apps are then uploaded manually to the delivery service and delivery emails are forwarded manually to users. Feedback is collected from users via phone, email and personal meetings.

Additional requirements and constraints mentioned before have been considered in our implementation: Our release mechanism is based on email and can be configured using mailing lists, removing the need for team members to learn new tools. We developed a custom-built, centralized delivery service developed in Java using modern web frameworks. This avoids uploading client data to cloud services and enables both manual and automated delivery – constraints that prevent the use of available services as e.g. HockeyApp[2] or TestFlight[3]. The integration system has been rebuilt from scratch, using available infrastructure and open-source components such as Jenkins[4] and suitable plugins. It can be ordered from central IT as a turn-key solution running in an isolated virtual machine. While it comes with a basic configuration, projects can adapt the entire tool set to their specific needs.

[2]http://www.hockeyapp.net
[3]http://www.testflightapp.com
[4]http://jenkins-ci.org



(a) Standard workflow, basis for tailoring: automatic build, automatic upload, manual distribution, both automatic and user-generated feedback

(b) Example of tailored workflow for large project: manual build, automatic upload and distribution, only automatic feedback

(c) Example of tailored workflow for small project: no dependency resolution, no metrics, manual upload, only user-generated feedback

Fig. 2. Extended and tailorable release management workflow with color-coding to indicate activities

It is important to note that the implementation of this workflow requires collaboration between several individuals as well as departments. The organizational transformation necessary to introduce and subsequently fine-tune this process involves even more detailed descriptions of aspects such as responsibilities, approval process, project-specific resources, reporting structures etc. However, this is not part of our discussion and covered in depth by others like Humble [10] and Poppendiek [22].

V. Evaluation

Our goal was to measure the effect of our extended and tailorable workflow on adoption and impact of continuous integration and delivery in mobile projects. We evaluated how the workflow is applied by project teams and how it influences their development process. Regarding *continuous integration*, we were concerned with the degree to which the project has adopted a continuous workflow as well as what and how they test, which metrics they collect and how they use them. Regarding *continuous delivery*, we were interested in the value of using the delivery service in combination with an integration system. We also looked at how this transforms the quantity, quality and nature of feedback the project receives from users. Overall, we addressed the following *research questions*:

- **RQ 1** Integration: How are projects building and testing changes and how often and timely are they doing it?
- **RQ 2** Testing: Which types of tests are projects employing and how are these tests run respectively?
- **RQ 3** Metrics: Which metrics are collected, how are they collected and how is the information used?
- **RQ 4** Delivery: How are projects delivering new builds to users, how much time and effort does it cost and how many users can be reached?
- **RQ 5** Feedback: Which feedback channels are used, how frequently is feedback collected and what is the quality of feedback?

A. *Study Design*

To evaluate our tailored workflow, we introduced it in eight mobile projects of Capgemini with customers from different industrial sectors.[5] We asked project managers to participate in a survey including both qualitative and quantitative questions. Additionally, informal interview questions were included for survey participants to further elaborate on the setup, activities, and situation in their project and to share their opinion about our solution.

[5]Projects used for field-testing the workflow are the ones we surveyed to identify constraints in Section 3.

We chose personal interviews using a survey as evaluation method to find qualitative and quantitative data. We wanted to know how project members use the workflow and how often they use it. We designed our survey to be akin to an expert interview to benefit from the technical and domain expertise of participants. We chose interviewees according to their technical and process knowledge. The format of a survey avoids problematic aspects of an interview such as lack of structure or influence by personality or setting [23].

Survey participants were project managers with multiple years of experience in the mobile domain and insights into how mobile projects work. All surveyed projects were familiar with agile methodologies and the integration system was already known within the company. However, there was little experience with automated delivery in general and no experience with our custom-built delivery service. Measured size and complexity of the surveyed projects revealed that both team and codebase of mobile projects are rather small compared to non-mobile software projects within the same company, which have up to ten times as many members.

B. Findings

We grouped our findings into five categories according to our research questions: integration, testing, metrics, delivery, and feedback. In each category we analyzed the overall impact of our solution as well as project advancements in detail. The majority of projects have adopted the new workflow with 75 % now using both integration system and delivery service for release management, most of them added continuous integration for the first time. The remainder either introduced continuous integration or just used the manual delivery workflow.

a) Integration: While the primary goal of our solution is to move projects to continuous delivery, a prerequisite is to move them to continuous integration [10]. We determined how many projects use a build server, how dependencies are managed and how immediately changes are built and tested. Usage of an integration system increases with a turn-key solution available, although some projects have their build server located at the client due to special requirements.

The high setup effort reportedly was a primary factor preventing projects from practicing continuous integration. Dependencies are now managed in a more structured manner with most projects using both a dependency management system and version control system. The effect of these improvements on the integration process is visible in Fig. 3: Instead of testing changes only sporadically, projects now either integrate changes immediately or at least regularly, e.g. in a nightly or weekly build.

b) Testing: Keeping in mind that the effort of defining and implementing tests remains unchanged, we looked at whether projects test more or differently when the right infrastructure is made available. Fig. 4 compares types of tests employed and the respective method of testing before and after introduction of our integration system. The introduction of a build server mainly transforms the way those tests are run: Unit, integration, and system tests become automated to a

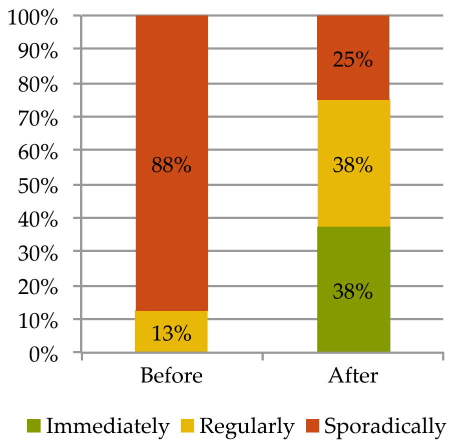


Fig. 3. Percentage of projects that integrate changes with respective frequency/immediacy (before and after workflow introduction)

great extent, reducing the time and effort required for manual testing. Results are immediately visible to all project members, leading to faster detection of bugs and regressions, improving software quality and reducing risk [24].

The overall percentage of tests run in a fully or semi-automated fashion quadruples. This improvement already saves projects a significant amount of time and enables the higher immediacy of integration seen in Fig. 3. Both functional and non-functional acceptance tests as well as additional types like usability of exploratory tests are still conducted manually. This kind of testing does not lend itself well to automation – at least not without significant effort and a highly individual setup [25], [26]. This would defeat the purpose of our approach and is reportedly not deemed worth the effort.

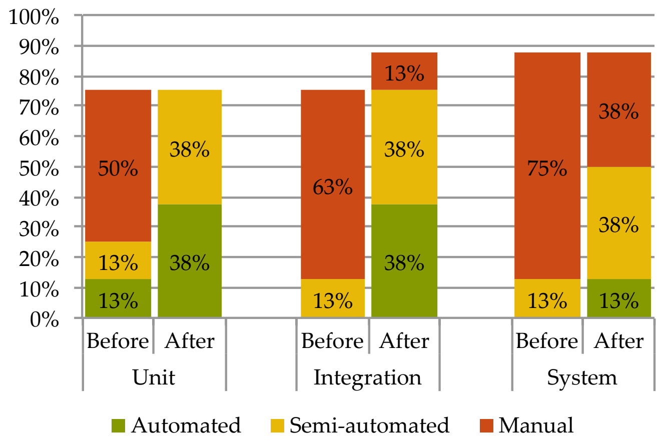


Fig. 4. Percentage of projects that apply certain types of tests with respective degree of automation (before and after workflow introduction)

c) Metrics: Our integration system allows projects to collect various metrics. This allows developers and project managers to gain deeper insights into the codebase, integration process and state of the project [27], [28]. We examined which metrics the projects collect, how they collect them and what the results are used for: Static analysis tools provide *codebase* *metrics* like code quality and test coverage. This information is mainly used for continuous *quality assurance* throughout the project. The integration system reports *integration metrics* like build success rate or duration of build and test steps. Projects use this for *status monitoring* and take action when they encounter failing builds or long-running tests.

Finally, *project metrics* like velocity and cycle time can be obtained from the issue tracker for *decision support*. The issue tracker is ideally connected to both the repository and the integration system and acts as a “single source of truth” regarding state and progress of the project. Fig. 5 shows that a majority of projects makes use of these possibilities while employing automatic metrics collection instead of collecting a few metrics manually or using scripts.

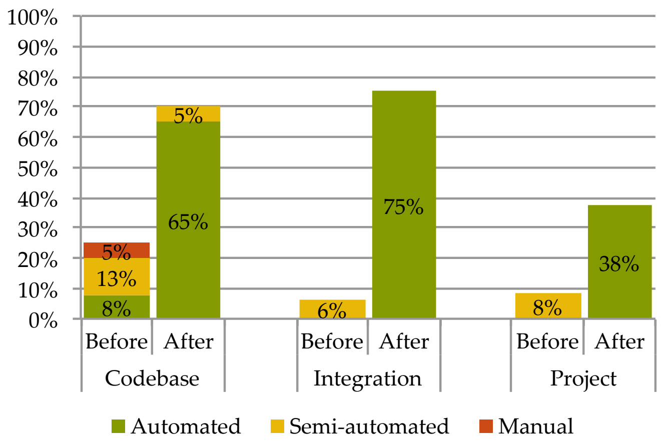


Fig. 5. Percentage of projects that collect certain types of metrics with respective degree of automation (before and after workflow introduction)

d) Delivery: Having implemented a custom delivery service that supports various degrees of automation, we were interested in its impact on complexity, effort, and duration of delivery. About 60 % of projects opted for semi-automatic delivery, i.e. including a manual control step between generation of a releasable build and its delivery to users, while the other 40 % chose to fully automate the process. Automation and simplification yield a reduction in both time and effort required for delivery: Getting a new version in the hands of users now takes fewer steps and requires fewer team members to be involved.

Number of steps required for delivery (e.g. building, signing, packaging, uploading the app and then notifying everyone) reduced from 5 to 1 (median). Projects with a more complicated release process were even able to reduce complexity from 10 to 1 (median). Likewise, involvement of team members in the delivery process was reduced by 25 % (median), in most cases requiring only one release manager. Before the introduction of an automated process, deliveries required one hour up to an entire business day. Projects were able to reduce this duration to 5 minutes (median).

These time savings implicitly improve cycle time and free up project members to address other tasks [10], [29]. As a side effect, projects reportedly plan to involve more external users in the future, now that complexity of delivery no longer increases with number of users.

e) Feedback: Finally, we were interested in whether applying continuous delivery not only saves time and effort but also increases the quantity and/or quality of feedback that can be collected from users. Because actual quantity of feedback is hard to measure and compare, frequency of collection was used as a proxy. Our results show which channels are utilized for feedback collection: For weekly feedback collection, 90 % of projects use email and phone, followed by meetings (50 %) and software tools (25 %). 40 % of projects supplement this with additional meetings, virtual meetings or custom tools on a less frequent basis.

Projects prefer channels that yield unstructured feedback and state ease of use as the primary reason for this. Reportedly, channels that facilitate personal communication also yield the best feedback quality. More frequent delivery as well as better integration between tools also yield more defined feedback. These findings correspond to the ones that we reported in [21].

C. Threats to Validity

We designed the survey to gain insight into the development process of mobile projects and to measure the impact of our release management workflow. We thoughtfully worded each question to avoid ambiguity of leading questions and carefully selected both test projects and survey participants. Despite our best efforts, our results may be subject to the following limitations:

- *Reliability:* Factors such as duration of the evaluation period, number of metrics, or level of detail may have influence on the reliability of our results. In any case, we can consider our findings anecdotal evidence for the impact of continuous delivery on mobile projects in a corporate environment [23].
- *Generalizability:* Number of projects and variation of project characteristics may be too low in order to achieve generalizeable results [30]. For example, we did not find a correlation between project size or complexity and any of the observed effects since the data is not sufficient to yield significant results. However, consistent results across all surveyed projects are an indication that our results apply to other projects as well.
- *Selection bias:* Projects participating in our survey may have already worked in an agile fashion and had a special interest in automated integration and delivery [30]. Although this might skew the impact of our solution to the positive, cultural acceptance generally is a prerequisite for successful agile projects [10], [31].
- *Researcher bias:* Bias caused by an appreciation for agile in general or our solution in particular as well as the positive results of our previous study [14] may have influenced the wording of our questions. To alleviate this threat, we chose survey participants with a level of expertise that allows them to correct for ambiguity and added open-ended questions to encourage full, meaningful answers [23].

VI. Conclusion

In this paper we showed how to introduce continuous delivery in a corporate environment where no standardized workflow for mobile projects existed before. Projects had to reinvent the wheel and created isolated solutions for the integration and delivery of apps when we investigated the current state at Capgemini, a global provider of consulting, technology and outsourcing services. Based on Rugby's release management workflow we analyzed the specific needs for continuous delivery in a heterogeneous project-based organization.

When comparing project-specific setups in the company with centralized services in university, we found that the decentralized nature of projects and the varying project characteristics impact the development process and infrastructure. We aimed to answer how Rugby's workflow needs to be extended in order to fit these requirements and restrictions and how much value continuous delivery provides to mobile projects under these conditions. The heterogeneity of the projects made it necessary to introduce tailoring in the workflow.

To address key issues in industrial mobile projects with respect to development process, infrastructures, neglected testing and continuity, we extended Rugby's release management workflow and made it tailorable, by splitting it into four activities – configuration management, integration, delivery and feedback – with optional and configurable steps. The tailorable workflow allows project managers to choose a solution that fits to their project environment and if necessary to incrementally introduce the components step by step.

We applied and evaluated our tailored workflow in eight mobile projects with diverse project environments and customers. We addressed research questions with respect to integration, testing, metrics, delivery and feedback. We found that duration of integration and delivery significantly decreased while their frequency increased. The investigated projects value the possibilities of automated testing and metrics collection. Due to the close evaluation after the introduction of the workflows we could not find significant changes in the feedback collection. From our evaluation, we have first anecdotal evidence that the tailored workflow provides a good compromise between technical promises and actual business needs.

The value of the introduced workflow is appreciated, especially the low time and effort required for setup that allows projects to get started quickly. Still, there is potential for improvement and expansion. The existing components already have the potential for highly complex and individual setups, e.g. finer control over the build process with build matrices and branch filters, automated user interface tests, or integration tests with a backend built in parallel. The next step is to apply a longer observation period to improve the incorporation of user needs and evaluate whether development behavior (e.g. number and size of commits) as well as feedback (e.g. quality and quantity) will change over time. Overall, we expect our work to be a strong basis for the agile development of mobile applications using continuous delivery in a standardized but customizable fashion in corporate environments.

References

[1] M. Fitzgerald, N. Kruschwitz, D. Bonnet, and M. Welch, "Embracing digital technology: A new strategic imperative," MIT Sloan Management Review in collaboration with Capgemini Consulting, 2013.

[2] S. Khalaf, "Flurry analytics insights," 2015, retrieved February 26, 2015 from http://www.flurry.com/blog/flurry-insights.

[3] S. Krusche and B. Bruegge, "User feedback in mobile development," in *Proceedings of the 2nd International Workshop on Mobile Development Lifecycle*. ACM, 2014, pp. 25–26.

[4] D. Pagano and W. Maalej, "User feedback in the appstore: An empirical study," in *Proceedings of the 21st International Requirements Engineering Conference*. IEEE, 2013, pp. 125–134.

[5] D. Pagano and B. Bruegge, "User involvement in software evolution practice: a case study," in *Proceedings of the 2013 international conference on Software engineering*. IEEE, 2013, pp. 953–962.

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

[7] VersionOne, "8th annual state of agile survey," 2014, http://www.versionone.com/pdf/2013-state-of-agile-survey.pdf.

[8] R. Pichler, *Agile product management with scrum: Creating products that customers love*. Addison-Wesley Professional, 2010.

[9] J. Bosch, "Continuous software engineering: An introduction," in *Continuous Software Engineering*. Springer, 2014, pp. 3–13.

[10] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison Wesley, 2010.

[11] C. Handy, *Understanding organizations: managing differentiation and integration*. Oxford University Press, 1993.

[12] W. Brown, H. McCormick, T. Mowbray, and R. Malveau, *AntiPatterns: refactoring software, architectures, and projects in crisis*. Wiley, 1998.

[13] O. Pedreira, M. Piattini, M. Luaces, and N. Brisaboa, "A systematic review of software process tailoring," *SIGSOFT SE Notes*, vol. 32, no. 3, pp. 1–6, 2007.

[14] S. Krusche, L. Alperowitz, B. Bruegge, and M. Wagner, "Rugby: An Agile Process Model Based on Continuous Delivery," in *Proceedings of the 1st International Workshop on RCoSE*. ACM, 2014, pp. 42–50.

[15] G. Kalus and M. Kuhrmann, "Criteria for software process tailoring: A systematic review," in *Proceedings of ICSSP*. ACM, 2013, pp. 171–180.

[16] I. Jacobson, G. Booch, J. Rumbaugh, J. Rumbaugh, and G. Booch, *The unified software development process*. Addison-Wesley, 1999.

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

[18] J. Feller, *Perspectives on free and open source software*. MIT, 2005.

[19] L. Zhao and S. Elbaum, "A survey on quality related activities in open source," *SIGSOFT Software Engineering Notes*, vol. 25, no. 3, pp. 54–57, 2000.

[20] M. Aberdour, "Achieving quality in open-source software," *Software*, vol. 24, no. 1, pp. 58–64, 2007.

[21] S. Krusche and L. Alperowitz, "Introduction of Continuous Delivery in Multi-Customer Project Courses," in *Proceedings of the 36th ICSE*. IEEE, 2014, pp. 335–343.

[22] M. Poppendiek and T. Poppendiek, *Implementing Lean Software Development: From Concept to Cash*. Addison Wesley, 2006.

[23] L. Van Audenhove, "Expert interviews and interview techniques for policy analysis," Vrije Universiteit Brussel, 2013.

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

[25] L. Crispin and J. Gregory, *Agile Testing: A Practical Guide for Testers and Agile Teams*. Addison Wesley, 2009.

[26] B. Marick, "Agile testing directions: tests and examples," 2003, retrieved February 26, 2015 from http://www.exampler.com/old-blog/2003/08/22.

[27] N. Fenton and J. Bieman, *Software metrics: a rigorous and practical approach*, 3rd ed. CRC Press, 2014.

[28] M. Lorenz and J. Kidd, *Object-oriented software metrics: a practical guide*. Prentice-Hall, 1994.

[29] M. Poppendiek and T. Poppendiek, *Lean Software Development: An Agile Toolkit*. Addison Wesley, 2003.

[30] D. Willimack *et al.*, "Evolution and adaption of questionnaire development, evaluation, and testing methods for establishment surveys," *Methods for Testing and Evaluating Questionnaires*, pp. 385–407, 2004.

[31] P. Gorans and P. Kruchten, "A guide to critical success factors in agile delivery," IBM Center for The Business of Government, 2014.