Characterizing Rapid Releases in a Large Banking Company: A Case Study

Elvan Kula*, Ayushi Rastogi
Delft University of Technology
Delft, The Netherlands
e.kula@student.tudelft.nl,
a.rastogi@tudelft.nl

Hennie Huijgens
ING
Amsterdam, The Netherlands
hennie.huijgens@ing.com

Arie van Deursen
Delft University of Technology
Delft, The Netherlands
arie.vandeursen

I. INTRODUCTION

The increased competition in today's business environment has pushed a large number of organizations to shift towards a rapid development model, *i.e.* a software development model with a short release cycle of a few days or weeks [1]. As the trend of rapid releases (RRs) is increasingly being adopted in open-source and commercial software [2], it is vital to understand how RRs work and how they affect the quality of the released software. Recent research on RRs addresses the impact of short release cycles on several aspects of software quality. However, a characterization of rapid releases is missing. By exploring the characteristics of RRs in industry, we can obtain valuable insights in what the urgent problems in the field are, and what data and techniques are needed to address them. Such knowledge can provide researchers with promising research directions that, in our opinion, can help industry today.

This paper presents a case study of rapid releases at ING, a large Netherlands-based internationally operating bank that develops software solutions in-house. Since 2011 the bank has gradually shifted to a rapid development model, which resulted in a mixed environment of rapid and traditional teams. We conducted a mixed-methods study, consisting of a survey with 461 engineers, and a statistical analysis of 2 years of code quality data for 611 teams.

II. RELATED WORK

Recent research on RRs addresses the impact of short release cycles on several aspects of software quality. RRs are claimed to offer a reduced time-to-market and faster user feedback; releases become easier to plan because of their smaller scope; end users benefit because they have faster access to functionality improvements and security updates [1][2][3]. Despite these benefits, previous research in the context of open source software (OSS) projects shows that RRs often come at the expense of reduced software quality. The known challenges are related to software reliability [1][4], accumulated technical debt [5] and increased time pressure caused by strict release dates [6].

* Work completed during an internship at ING

Fig. 1. Continuous Delivery Pipeline at ING NL

Existing studies on rapid release cycles are explanatory and a majority of these studies consider RRs in the context of OSS projects. In this paper, we present new knowledge by performing a case study of RRs in a large software-driven organization.

III. CONTEXT

ING is a large multinational financial organization with about 54,000 employees and over 37 million customers in more than 40 countries [7]. In 2011, ING decided to shorten their development cycles when they planned to introduce *My ING*, a personalized mobile application for online banking. Before 2011, teams worked with release cycles of two to three months, but ING wanted to cut the cycles down to less than a month to stay ahead of the competition. To make shorter release cycles practical, the bank introduced the *Continuous Delivery as a Service* project in 2015 to automate the complete software delivery process. Fig 1 depicts the pipeline. Currently, all teams at ING work with a *time-based release strategy*, in which releases are delivered at regular intervals. Teams can deviate from their regular cycle length in case of a release delay.

Defining rapid release cycles. There is no general definition of the length of a rapid release cycle in literature. In ING, we defined the duration of a rapid release cycle based on the distribution of release frequencies shown in Figure 2. The mean of the distribution is 3.56 weeks, meaning that all teams that follow a cycle time shorter than 3.56 weeks release faster than the average team at ING and therefore rapidly. Since teams work with release cycles of full weeks, we consider teams that follow a cycle shorter than 4 weeks to be rapid and traditional otherwise. This way we identified 433 rapid teams (71%) and 178 traditional teams (29%) at ING.

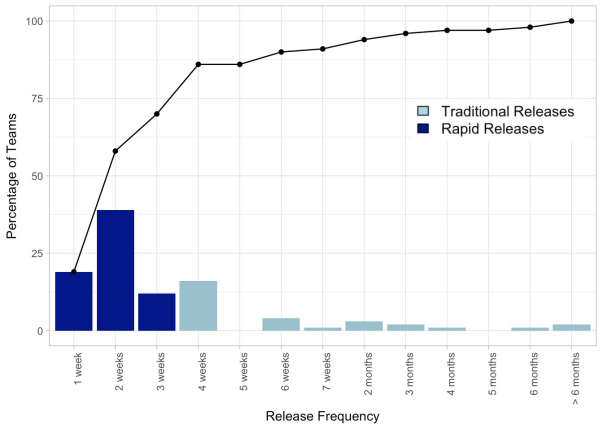

Fig. 2. Distribution of release frequencies at ING: the black line represents the cumulative percentage of the teams.

IV. RESEARCH METHOD

As RRs are driven by the idea of reducing release cycle time, timing aspects are intrinsic to rapid release cycles. It is therefore vital to examine the *timing characteristics* of rapid release cycles to understand how RRs work and when they are feasible in organizations. To explore the prevalence of delays in RRs, we define our first two research questions as follows:

> **RQ1:** *How often do rapid and traditional teams release software on time?*
>
> **RQ2:** *What causes delay in rapid release cycles?*

In RRs, developers have less time to manage software quality, which might affect the quality of the released software. By examining the *quality characteristics* of RRs, we may better understand the long-term consequences of short release cycles and inform the design of tools to help developers manage them. This leads to our third research question:

> **RQ3:** How do rapid release cycles affect code quality?

A. Study Design

We conducted a mixed-methods study to address the research questions [8]. We collected qualitative data using an online survey in two phases [9]. In the first phase, we ran a pilot study with the members of four teams at ING. This allowed us to refine the survey questions. In the second phase, we sent the final survey to the members of all teams at ING Netherlands (ING NL). In total, we contacted 1803 participants belonging to more than 350 teams, each working on their own application that is internal or external to ING. In addition, we analyzed quantitative data stored in *SonarQube*[1] to examine the quality characteristics of rapid releases.

Survey Design. The pilot and final survey were organized into five sections, plus a preliminary section aimed at gathering demographic information of the respondents (*i.e.*, role within the team, total years of work experience and total years in ING NL). The five sections were composed of open-ended questions mixed with multiple choice or Likert scale questions. To address RQ1 we asked respondents to fill in a compulsory multiple choice question on how often their team releases on time. For RQ2 and RQ3 we provided respondents with two compulsory open-ended questions. In addition, we provided the respondents with a mandatory multiple choice question about their team's release frequency and a few optional questions about the way they monitor code quality.

Respondents. For the final survey we obtained 461 responses (26% response rate). 296 out of 461 respondents (64%) work in rapid teams, and 165 respondents (36%) work in traditional teams. Teams are similar in terms of size (95% CI: 5 to 9 members), experience (95%: 10 to 20 years) and number of respondents (95%: 1 to 2 respondents).

B. Survey Analysis

We performed manual coding [10] to summarize the results of the open-ended questions during two integration rounds. In the first round, the first and last author coded a different 10% sample of 40 responses for each question. Next, they met in person to integrate the obtained codes. Then the first author applied the integrated codes to 90% of the answers and the second author did this for the remaining 10% of the responses. In the second round, the first two authors had another integration meeting which resulted into the final set of codes.

C. Collecting Software Metrics

To analyze the quality of software written by traditional teams in comparison with rapid teams, we extracted the SonarQube measurements of releases that were shipped by the 611 DevOps teams in the period from July 01, 2016 to July 01, 2018. In total, we studied the major releases of 3048 software projects. We classified the releases as traditional or rapid based on the time interval between their release date and start date of the development phase (using 4 weeks as threshold). 67% of these releases were developed following a rapid release cycle ($<$ 4 weeks) and the remaining 33% of the releases were developed following a traditional release cycle ($\geq$ 4 weeks). For each release, we extracted the metrics that teams at ING measure to assess their coding performance.

V. RESULTS

This section presents results on timing and quality characteristics of rapid releases. The codes resulting from our manual coding process are underlined.

RQ1: *How often do rapid and traditional teams release software on time?*

A majority of teams, both traditional and rapid, release software timely less often than 50% of the time. Only 8% of the rapid teams and 16% of traditional teams release software 75% to 100% on time. The remaining teams release software less often on time. A distribution of the rapid and traditional teams based on the percentage of times they release software on time is shown in Table I.

[1]https://www.sonarqube.org/

TABLE I
PERCENTAGE DISTRIBUTION OF RAPID AND TRADITIONAL TEAMS BASED ON THE PERCENTAGE OF TIMES THEY RELEASE SOFTWARE ON TIME

% of software releases on-time	RRs	TRs
Almost Always (75 - 100%)	8%	16%
Mostly (50 - 75%)	16%	24%
Rarely (25 - 50 %)	35%	26%
Almost Never (0 - 25 %)	41%	34%

Table I shows that the percentage of teams which release less often increases for traditional and rapid teams. It also shows that rapid teams are always more delayed than their traditional counterparts. Further analysis of the survey responses indicates that a majority of the rapid teams that are on track 75% to 100% of the time develop web applications (54%) and desktop applications (18%). The rapid teams that are less than 25% of the time on track develop mobile applications (68%) and APIs (19%).

RQ2: *What causes delay in rapid release cycles?*

Respondents mentioned several factors that they think introduce delays in releases. A list of these factors arranged in the decreasing order of their relevance in rapid teams is shown in Figure 3.

Figure 3 shows that dependencies and infrastructure (which can be considered a specific type of dependency) are the most prominent factors that are perceived to cause delay in rapid teams. Developers attributed technical cross-project dependencies and organizational task dependencies to cause delay in their releases. Many respondents indicate to struggle with estimating the impact of both types of dependencies in the release planning. Regarding infrastructure, respondents mentioned that issues are related to the failure of automation tools and sluggishness in the pipeline.

Other factors which were listed in at least 10% of responses are testing (in general and for security), following mandatory procedures (such as quality assurance and security testing) prior to every release, fixing bugs, and scheduling the release, including planning effort and resources.

Traditional teams experience similar issues. Similar to rapid teams, traditional teams report to be largely influenced by dependencies. The other factors which were considered important by at least 10% of the respondents are scheduling, procedure, and security testing.

RQ3: *How do rapid release cycles affect code quality?*

Here we examine the quality characteristics of rapid releases and compare the perceptions of developers with code quality data for rapid and traditional teams.

A. Developers' Perceptions

Developers had mixed opinions on how rapid release cycles affect code quality. A distribution of the effect of factors (improve, degrade, no effect) related to RRs as perceived by developers is shown in Figure 4. It shows responses suggesting improvements in quality in green, degradation in quality in red, and no effect in grey.

A majority of developers perceive that the small changes in code and rapid feedback improve software quality. The small changes make the code easier to review, positively impacting the refactoring effort. Many developers also mention the benefits of the rapid feedback in RRs. Feedback from issue trackers and the end user allows teams to continuously refactor and improve their code quality based on unforeseen errors and incidents in production. Rapid user feedback is reported to lead to a greater focus of developers on customer value and software reliability.

Most developers report to experience an increased deadline pressure in RRs, which negatively affects the code quality. They believe that this leads to an increase in workarounds and poor implementation choices. Many developers also report that given the shorter timeframe of RRs they often cut the overall time spent on refactoring and regression testing. A majority of the developers acknowledge that system design can suffer from the short-term focus in RRs. As one of the respondent puts it "*smaller decisions are made and it gets easier to lose sight of the big picture*".

A minority of the developers mention not to experience any decrease in quality due to certain coding standards. Their releases have to pass a threshold regarding code quality and security, which is enforced through code reviews and enabled through sufficient time for code refactoring (on average 25% of the time per sprint).

B. Software Quality Measurements

The results of our software quality analysis are summarized in Table II. To account for differences in project size, we normalized all metrics by dividing them by SLOC.

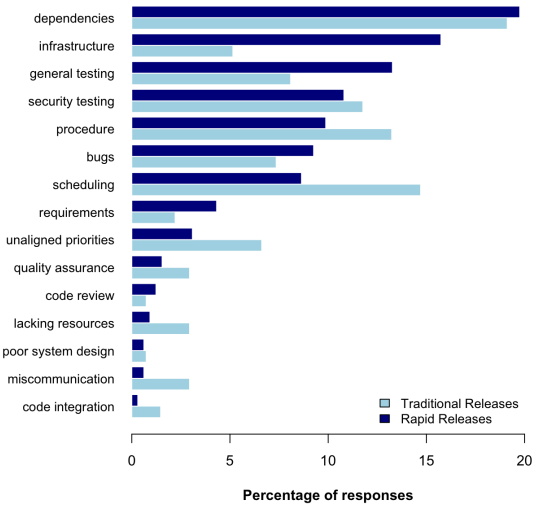

Fig. 3. Factors perceived to cause delays in rapid and traditional teams

TABLE II
EFFECTS OF RRS ON SOFTWARE METRICS

Metric	Mean		Median		Min		Max	
	TR	RR	TR	RR	TR	RR	TR	RR
Coding Standard Violations Density*	0.06	0.03	0.03	0.02	0.00	0.00	0.13	0.09
Cyclomatic Complexity*	0.17	0.14	0.16	0.15	0.09	0.03	0.41	0.22
Branch Coverage*	43.02	57.51	33.90	68.40	0.00	0.00	91.00	97.00
Comment Density	11.07	10.13	8.30	7.20	0.70	0.40	31.50	28.40
Code Churn*	0.03	0.05	0.01	0.03	0.04	0.05	0.15	0.20
SLOC	83983	67066	9286	7447	315	1107	721346	305319

* indicates statistical significance (p-value < 0.05) based on Mann-Whitney U test [11]

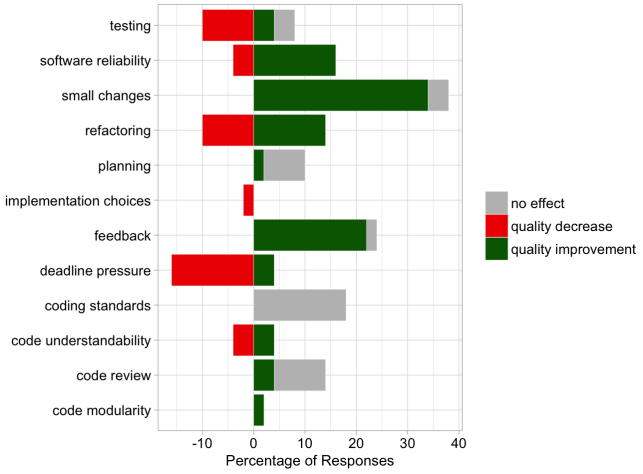

Fig. 4. Developer perception of the impact of rapid releases on code quality

We found that the *cyclomatic complexity* and *coding issues* are significantly lower in RRs. This result corresponds with the perception of developers that it gets easier to review and refactor code in RRs. We also found that the *branch coverage* is significantly higher in RRs. This result corresponds with the perception that the test process in RRs becomes more continuous and allows for more complete testing of new features. The *code churn* is significantly higher for RRs, indicating that there is a higher coding activity in rapid teams than in traditional teams. We did not find a significant difference between the *comment density* and *SLOC* of RRs and TRs.

VI. IMPLICATIONS

Our study indicates that rapid releases can be beneficial in terms of reviewing, testing, and user-perceived quality. Here we identify six core challenges that require further attention, both in practice and in research, in order to move rapid releases forward:

1) **Are rapid releases for everyone?** Rapid releases in API and mobile app development are more often delayed than in other types of projects. This suggests that project type plays a role in the success of RRs. An investigation of factors that affect the release cycle time could help the field to better understand when (and why) RRs are feasible and to adapt their practices.
2) **The balance game: security versus agility.** Security testing can induce delay in rapid teams, suggesting that organizations make a *trade-off* between rapid delivery and security. A financial organization like ING with zero tolerance for systemic failures chooses reduced effectiveness over rapid release. Further research should focus on agile security to work towards the automation of security testing, and the design of security measures that are able to adapt rapidly to changing requirements and a rapid development environment.
3) **Feedback-driven development.** The rapid feedback in RRs improves the focus of developers on the quality of software. Feedback obtained from end users, code reviews and static analysis can be used to (1) guide teams to focus on the most valuable features, and to (2) enable automated techniques to support various development tasks in order to further reduce the cycle length. An exploration of these opportunities would help organizations to improve their software quality.
4) **Release planning.** Regarding delays, our respondents express the need for more insight on improving software effort estimation and streamlining dependencies. Recent approaches such as automated testing, *Infrastructure as Code* and dependency management seem promising for further research on release planning and predictable delivery in the context of RRs.
5) **Dependency management.** We found that the timing of both RRs and TRs is largely influenced by dependencies in the ecosystem of the organization. Our respondents report the difficulty of assessing the impact of dependencies. There is a need for more insight into the characteristics and evolution of these dependencies. This problem calls for further research into streamlining dependency management, such as the work of [12][13][14].
6) **Managing code quality.** In our study we observed that a minority of the teams claim not to experience the negative consequences of RRs on code quality. The self-reported 'good' teams perform regular code reviews and dedicate (on average) 25% of the time per sprint on refactoring. However, this result is context-specific and further research is required to validate best practices for debt management in RRs.

REFERENCES

[1] Khomh, F., Dhaliwal, T., Zou, Y., & Adams, B. (2012, June). Do faster releases improve software quality?: an empirical case study of Mozilla Firefox. In Proceedings of the 9th IEEE Working Conference on Mining Software Repositories (pp. 179-188). IEEE Press.

[2] Mäntylä, M. V., Adams, B., Khomh, F., Engstrm, E., & Petersen, K. (2015). On rapid releases and software testing: a case study and a semi-systematic literature review. Empirical Software Engineering, 20(5), 1384-1425.

[3] Beck, K., & Gamma, E. (2000). Extreme programming explained: embrace change. addison-wesley professional.

[4] da Costa, D. A., McIntosh, S., Kulesza, U., & Hassan, A. E. (2016, May). The Impact of Switching to a Rapid Release Cycle on the Integration Delay of Addressed Issues-An Empirical Study of the Mozilla Firefox Project. In Mining Software Repositories (MSR), 2016 IEEE/ACM 13th Working Conference on (pp. 374-385). IEEE.

[5] Tufano, M., Palomba, F., Bavota, G., Oliveto, R., Di Penta, M., De Lucia, A., & Poshyvanyk, D. (2015, May). When and why your code starts to smell bad. In Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on (Vol. 1, pp. 403-414). IEEE.

[6] Rubin, J., & Rinard, M. (2016, May). The challenges of staying together while moving fast: An exploratory study. In Proceedings of the 38th International Conference on Software Engineering (pp. 982-993). ACM.

[7] ING. (2018, March). 2017 Annual Report ING Group N.V.

[8] Creswell, J. W., & Creswell, J. D. (2017). Research design: Qualitative, quantitative, and mixed methods approaches. Sage publications.

[9] Flick, U. (2014). An introduction to qualitative research. Sage.

[10] Corbin, J. M., & Strauss, A. (1990). Grounded theory research: Procedures, canons, and evaluative criteria. Qualitative sociology, 13(1), 3-21.

[11] Mann, H. B., & Whitney, D. R. (1947). On a test of whether one of two random variables is stochastically larger than the other. The annals of mathematical statistics, 50-60.

[12] Hejderup, J., van Deursen, A., & Gousios, G. (2018, May). Software ecosystem call graph for dependency management. In Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results (pp. 101-104). ACM.

[13] Decan, A., Mens, T., & Claes, M. (2017, February). An empirical comparison of dependency issues in OSS packaging ecosystems. In 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER) (pp. 2-12). IEEE.

[14] Decan, A., Mens, T., & Grosjean, P. (2018). An empirical comparison of dependency network evolution in seven software packaging ecosystems. Empirical Software Engineering, 1-36.