Towards quality gates in continuous delivery and deployment

Schermann, Gerald ; Cito, Jürgen ; Leitner, Philipp ; Gall, Harald C

Abstract: Quality gates, steps required to ensure the reliability of code changes, are supposed to increase the confidence stakeholders have in a release. In today's fast paced environments, we have less time to perform the necessary precautions to minimize the risk of a faulty release. This leads to an inherent trade-off between risk of lower release quality and time to market. We provide a model for this trade-off of release confidence and velocity that led to the formulation of 4 categories (cautious, balanced, problematic, madness), in which companies can be classified in. We showcase real examples of these categories as case studies based on previous empirical studies. We close by presenting possible transitions between categories that guide future research.

DOI: https://doi.org/10.1109/ICPC.2016.7503737

Towards Quality Gates in Continuous Delivery and Deployment

Gerald Schermann, Jürgen Cito, Philipp Leitner, and Harald C. Gall
University of Zurich, Department of Informatics, Switzerland
{schermann, cito, leitner, gall}@ifi.uzh.ch

***Abstract*—Quality gates, steps required to ensure the reliability of code changes, are supposed to increase the confidence stakeholders have in a release. In today's fast paced environments, we have less time to perform the necessary precautions to minimize the risk of a faulty release. This leads to an inherent trade-off between risk of lower release quality and time to market. We provide a model for this trade-off of release "confidence" and "velocity" that led to the formulation of 4 categories (*cautious, balanced, problematic, madness*), in which companies can be classified in. We showcase real examples of these categories as case studies based on previous empirical studies. We close by presenting possible transitions between categories that guide future research.**

I. Introduction

Staying ahead of competition is one of the main principles of how companies operate. For companies that deliver software, being competitive mostly translates into the ability to deliver new functionality quickly and efficiently. This often leads to the mantra of "move fast and break things", which has been made popular by Facebook [1]. This approach can take a toll on the resulting reliability of a new release. Traditionally, quality gates (steps required to ensure the reliability of change) are supposed to increase the confidence stakeholders have in the reliability of a release. However, a "move fast" environment leaves less time to perform the necessary precautions to minimize the risk of a faulty release. This leads to an inherent trade-off between risk of lower release quality and time to market. Companies, tool vendors, and researchers need to understand the problem space of this trade-off to make informed decisions on how to influence their release process within this area. To provide the means for process comprehension in this space, we propose a high-level model over release *confidence* and *velocity*.

The idea for this work evolved from two recently conducted studies – an analysis of the impact of cloud computing on software development processes [2], and a study on how the practices associated with continuous delivery and deployment found their way into the broad software industry [3]. In both studies, we conducted qualitative interviews with 25 and 20 professional software developers or release engineers, respectively, and validated the results by quantitative surveys with more than 450 ($>$290 and $>$180) practitioners in total. The qualitative parts of both studies revealed and indicated a relation between the effort companies put into quality gates throughout the development process (*confidence*) and the pace with which they can release new versions of their applications (*velocity*). In this work, we investigate this relation and derive a model based on release confidence (emerging from the quality gates) and the velocity of releases. The model is structured into 4 high level categories. We discuss their properties and showcase companies falling into these categories. Moreover, we touch on potential future research on how companies can self-assess their release confidence and velocity, and how academic and industrial research can support and guide the transition within those categories.

The primary contributions of this paper can be summarized as follows:

- Based on a definition of release confidence and velocity within the realm of quality gates in software releases, we formulate a model to classify companies in high-level categories of that spectrum. The model consists of four categories (*cautious, balanced, problematic, madness*) showcasing the tradeoffs between confidence and velocity.
- We identify and discuss ways of how companies can transition between those categories as first guidelines.

The rest of the paper is organized as follows. Section II introduces our model of release confidence and velocity. In Section III, the categories resulting from our model are complemented with case studies from industry. Section IV showcases possible transitions between the different categories. Section V contrasts our work within related literature. Section VI concludes our work and gives an outlook for future research.

II. Confidence-Velocity Model

In this section, we introduce the model that emerged from our previously conducted studies. As a first step, we define the terms of release confidence and velocity used within this work and then discuss the derived model in more detail.

A. Overview

Companies' quality assurance processes usually comprise testing activities (manual and/or automated) and code reviews. The outcome of those activities have substantial impact on the decision whether an application or change is ready to be released, or additional development effort is required for the purpose of leveraging stability and quality.

Automation, starting from committing changes to version control systems, through quality gates, until a new version is ready to be released is a fundamental element of continuous

delivery and deployment (CD) [4]. CD influences many phases of traditional development processes [5] and is associated with several risk mitigation strategies for keeping the impact of issues low [3].

Confidence. Tests and code reviews are essential elements for companies to ensure that changes, once deployed to production environments, behave as expected. In this work, we define release confidence as the amount of confidence gained on the three quality gates automated testing, manual testing, and code reviews. As shown by Inozemtseva and Holmes [6], single factors such as code coverage are not sufficient to measure the effectiveness of tests. Thus, to quantify release confidence gained by automated and manual testing, and code reviews, multiple factors (i.e., internal and external) need to be taken into account. Besides code coverage, factors such as the number of previously passed tested runs, or the number of automatically detected issues in code sections may be worth considering. Concerning code reviews, not only the quantity of code reviews might play a role (e.g., mandatory for every commit), but also who conducts them (e.g., team members, external experts), their level of expertise, and who makes sure that resulting change requests are implemented correctly.

Velocity. Velocity is the pace with which changes are running through the quality gates, starting with the commit of a change until it reaches the production environment. In our simplified model, we reduce velocity to the time needed to assess each single quality gate and to build and deploy the application. This includes delays due to manual decisions about the release readiness (e.g., approval processes) based on the results of test execution and code reviews. Moreover, we include the effort for developing and maintaining (automated) tests that varies over the course of time. However, we exclude the time invested in setting up and configuring appropriate tooling as those are non-recurring.

Establishing the terms *confidence* and *velocity* allows us to properly formulate the problem space that is inherent when considering these properties within a release process.

B. Confidence-Velocity Categorization Model

Based on confidence and velocity, we derive a simplified model that leads to four categories arranged on a grid from both low to high confidence and velocity. The aim of this model is to illustrate the conflicting nature of the model parameters. The resulting categories depict how this conflict facilitates in software release processes. The model is depicted in Figure 1. In the following we will introduce and discuss the peculiarities of each category.

Cautious. Companies in this category are careful when it comes to releases. They are characterized by a high emphasis on (automated) testing, code reviews for a multitude of commits, and manual approval processes decreasing velocity (e.g., external pressure because of domain requirements or customer expectations). Well-maintained, automated test suites are preferred to manual testing for the purpose of mitigating risks of human-caused errors (e.g., steps not executed correctly).

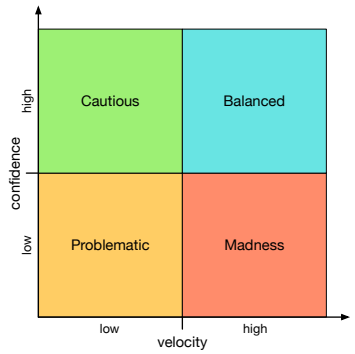

Fig. 1. Confidence-Velocity Categorization based on ratings for confidence and velocity

Manual tests are supplemental and applied to gain additional confidence in areas that are hard to test automatically.

Problematic. Companies in this category lack confidence in their quality gates. Reasons include, amongst others, missing regression testing, the absence of or not enough code reviews, insufficiently maintained test suites (both automated and manual) or their wrong execution, a shortage in quality assurance personnel, or unclear roles and responsibilities regarding quality assurance. Velocity is either low as a direct consequence of this, or because of external pressure (e.g., a company's domain), or internal organizational and technical challenges (e.g., lack of automated tooling, legacy systems).

Madness. Madness is the combination of *problematic* with high velocity.Companies in this category benefit from short release cycles, thus early customer feedback and reduced time to market. Issues might be fixed fast, but the lack of proper quality gates make releases risky and stressful. In contrast to *problematic*, quality assurance often plays a minor role by choice. That is, these companies decide on that the (perceived) benefits of quality assurance are not worth the costs.Customer feedback is the main "quality assurance mechanism" – as long as there are no complaints, everything is considered to be fine.

Balanced. This category portrays the vision of continuous delivery [4], [7]. This category is defined by high velocity, thus taking advantage of reduced time to market and early customer feedback, combined with high confidence about the quality of releases. Automation is essential to keep velocity high, with manual testing only playing a minor role. Code reviews are of central importance (e.g., for critical code sections), but are often not overly formal. Moreover, this category is characterized by post-deployment quality assurance mechanisms. This includes runtime monitoring for detecting issues that made it through the quality gates and sophisticated risk mitigation strategies (e.g., CD practices such as canary deployments or dark launches).

III. Case Studies

In this section, we revisit the categories of our model in form of small case studies, not only to shed some light on the decisions as to why companies choose, or fall into, a specific category, but also to demonstrate that these categories make sense, and can be used to study transitions between categories in future research. Table I provides an overview of

TABLE I
OVERVIEW OF COMPANIES PROVIDING CASE STUDIES FOR CATEGORIES

#	Domain	Category	Participant # / Study
C1	E-Government	Cautious	P9 in CD study [3]
C2	Telecommunications	Cautious	P4 in CD study [3]
C3	Document Processing	Problematic	P2 in CD study [3]
C4	E-Commerce	Balanced	P19 in CD study [3]
C5	Project Management	Madness	P11 in cloud study [2]

the companies used for the case studies, as well as the source of these cases.

A. Cautious

One of the major factors forcing companies into this category is external pressure. For example, C1, a company developing applications in the area of e-government, has to keep the release frequency low as their customers simply do not allow more frequent releases. Internally, considering technical requirements, they could release more frequently, but manual decision processes delay the overall velocity forcing them to be *cautious*.

Another example for a company in this category is C2, which is developing web-based applications in the domain of telecommunication systems. Even though their internal pipeline for building, testing, and releasing software would support more frequent releases, they release four times a year. They have strict organizational policies and testing guidelines. Six to eight weeks are explicitly reserved for testing new releases on production-like systems before they are considered "ready" to be released to customers. According to our interview participant, these guidelines (e.g., 100% code coverage for all classes) were even too strict and resulted in chaos when it came to maintaining automated tests. Unit tests were optimized for reaching code coverage, not necessarily to identify bugs. In the end, they put more effort into fixing broken tests than actual feature development. Due to the size of the ever increasing code base, they were not able to fix it and at some day, they had to accept that broken tests just do not pass anymore. This company is an example that being too *cautious* could lead – in certain circumstances – to a decreasing release confidence. Hence, C2 could be also considered for the *problematic* category.

B. Problematic

A company being slow in terms of their release frequency, thus low velocity, coupled with a low release confidence is C3. Velocity is low because of their application type (i.e., on-premises enterprise software), thus there are no incentives in speeding up velocity. However, according to our interview participant, they have severe problems regarding code quality, even though they have a dedicated quality assurance team. QA focuses on automated regression testing, and tests for newly contributed code are not mandatory. Thus, few new code contributions come with associated automated test code. They lack of code reviews, wherein quality issues would be detected and discussed. Hence, developers in C3 also miss out on learning effects. The interview participant mentioned that it would help dramatically if they would introduce code reviews similar to the policy of the Mozilla Development Network[1], especially for newly hired developers. This is again an example for a company targeting *cautious* but failed and thus resides in the *problematic* category.

C. Balanced

An example for a company in this category is C4. They take advantage of early customer feedback and reduced time to market. At the same time, they have a good feeling about the quality of upcoming releases, knowing that if things go wrong, they are detected fast and could be reverted or fixed within minutes. However, every deployment comes with a sort of a review, even if it is just by quickly tipping a coworker's shoulder and asking for advice. Such short release cycles with multiple deployments a day need a level of trust and culture. Everyone is aware that it does not matter how large the set of automated tests is, there would always be changes that could have negative impacts and those will not be detected. As our participant mentioned, the degree of testing is proportional to the amount of risk involved. More, but smaller changes and sophisticated runtime monitoring help keeping confidence and velocity in balance.

D. Madness

High velocity paired with low confidence would be the literal mantra of "move fast and break things" [1]. However, this combination is usually countered with the ability to leverage early feedback, safe experimentation, and rollbacks, as described in our previous case study of a *balanced* company. In the case of C5, these measures were not implemented as part of the release process. No quality gates (except the software developer's own tests) were put in place before releasing changes to all users in production. The participant elaborated that the company weighed the costs of having dedicated QA to releasing bugs from time to time. This category is appealing to companies with smaller codebases (i.e., startups), because it provides the ability to push new features very fast, without the cost of QA. However, as the company and its codebase grows, it makes sense to consider transitioning away from the *madness* category.

IV. TRANSITIONS

The derived model allows to discuss the consequences of those categories, identify research gaps on how to better support companies in a certain category as well as on the migration towards other categories. In this section, we will showcase two key transitions: cautious to balanced and madness to balanced.

Cautious to Balanced. Suppose we have a company similar to C2. Even though high automation is reached, there is an extensive set of tests which prevents them from bringing new features or changes in case of issues faster to production. When confronted with runtime issues that are in need of an immediate hotfix, one option would be to execute only a subset of relevant tests ensuring that the hotfix does not break core

[1]https://developer.mozilla.org/en-US/docs/Code_Review_FAQ

functionality. There has been a multitude of research on the field of selective regression testing [8]. However, as we learned throughout the interviews of our two studies [2], [3], none of the companies used any tool resulting from this research. Instead, they either executed all tests or a subset based on intuition. Thus, what is missing here is a transfer of research knowledge back to the software industry, providing companies with tool support which allows them to increase the velocity while maintaining a high level of confidence.

Madness to Balanced. The *madness* category shows the other extreme. There are either no tests at all, or just a small set that leads to little confidence. Thus the task here is to identify those code blocks or functionality which are "hot" in order to have a starting point for testing and reviewing it. Basically, this field is strongly related to bug prediction mechanisms based on static code analysis [9]. One potential extension would be the combination of runtime data (e.g., profiling) and static code analysis. Application performance monitoring tools such as New Relic[2] have gained attention and may provide a valuable entry point for future research.

V. Related Work

To the best of our knowledge, this work is the first attempt categorizing companies based on their effort on quality gates and the time needed for bringing changes to production.

There has been work on release frequencies and their impact. Khomh et al. [10] studied the impact of release frequencies on software quality, one of the observations was that with shorter release cycles users do not experience more post-deployment bugs. Mäntylä et al. [11] analyzed the impact on software testing efforts when reducing the release cycle time in a case study of Mozilla Firefox. Stewart et al. [12] tried to relate code quality to factors such as release frequency, number of releases and the changes in size of the code base across releases in open source software projects, but could not derive any conclusions. McIntosh et al. [13] studied the impact of code review coverage on software quality. Baccelli and Bird [14] investigated the outcomes and challenges of modern tool-based code reviews and observed that, amongst other things, code reviews are not only used for revealing defects, but also serve to promote awareness and knowledge transfer. Slightly related to velocity is the work of Staron et al. [15]. They identified the time-to-release as the main indicator for release readiness based on a number of metrics.

VI. Conclusion

We derived a simple model for release confidence and velocity consisting of four categories. We discussed the peculiarities of each category and presented small case studies. This work is a first step towards a model which allows companies to consider velocity and confidence as a vehicle to aid process comprehension. We propose the following directions for future research:

Quantification of release confidence and velocity. We have provided high-level definitions of *confidence* and *velocity*. To support self-assessment, concrete mechanisms are required allowing a quantification of confidence and velocity. This includes, the identification of factors influencing both dimensions and investigating the importance of those factors (e.g., through weighting) across various company and application types.

Guidelines. We plan to establish guidelines for companies to support them in improving their current situation and migrating between categories.Thus, research has to investigate what the impact of transitions is, identify the problems companies face in the four categories, and how those problems can be tackled to support the migration towards other categories.

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 610802 (CloudWave), and from the Swiss National Science Foundation (SNF) under project "Whiteboard" (SNF Project no. 149450).

[2]http://newrelic.com/

References

[1] D. G. Feitelson, E. Frachtenberg, and K. L. Beck, "Development and Deployment at Facebook," *IEEE Internet Computing*, 2013.

[2] J. Cito, P. Leitner, T. Fritz, and H. C. Gall, "The Making of Cloud Applications: An Empirical Study on Software Development for the Cloud," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*. ACM, 2015, pp. 393–403.

[3] G. Schermann, J. Cito, P. Leitner, U. Zdun, and H. Gall. (2015) An empirical study on principles and practices of continuous delivery and deployment. PeerJ Preprints 4:e1889v1 https://doi.org/10.7287/peerj.preprints.1889v1.

[4] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation*. Addison-Wesley Professional, 2010.

[5] B. Adams and S. McIntosh, "Modern Release Engineering in a Nutshell – Why Researchers should Care," in *Proceedings of the International Conference on Software Analysis, Evolution, and Reengineering (SANER), Future of Software Engineering (FOSE) Track*, 2016.

[6] L. Inozemtseva and R. Holmes, "Coverage is not strongly correlated with test suite effectiveness," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 435–445.

[7] L. Chen, "Continuous Delivery: Huge Benefits, but Challenges Too," *Software, IEEE*, vol. 32, no. 2, pp. 50–54, Mar 2015.

[8] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: A survey," *Softw. Test. Verif. Reliab.*, vol. 22, no. 2, 2012.

[9] N. Ayewah, D. Hovemeyer, J. D. Morgenthaler, J. Penix, and W. Pugh, "Using static analysis to find bugs," *IEEE Software*, vol. 25, Sept 2008.

[10] F. Khomh, T. Dhaliwal, Y. Zou, and B. Adams, "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 (MSR)*. IEEE Press, 2012, pp. 179–188.

[11] M. V. Mäntylä, F. Khomh, B. Adams, E. Engström, and K. Petersen, "On Rapid Releases and Software Testing: A Case Study and a Semi-Systematic Literature Review," *Empirical Software Engineering*, 2014.

[12] K. J. Stewart, D. P. Darcy, and S. L. Daniel, "Observations on patterns of development in open source software projects," *SIGSOFT Softw. Eng. Notes*, vol. 30, no. 4, pp. 1–5, May 2005.

[13] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, "The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects," in *Proc. of the 11th Working Conf. on Mining Software Repositories*. ACM, 2014, pp. 192–201.

[14] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in *Proc. of the 2013 International Conf. on Software Engineering*, ser. ICSE '13. IEEE Press, 2013, pp. 712–721.

[15] M. Staron, W. Meding, and K. Palm, *Agile Processes in Software Engineering and Extreme Programming: 13th International Conference, XP 2012, Malmö, Sweden, May 21-25, 2012. Proceedings*. Springer Berlin Heidelberg, 2012, ch. Release Readiness Indicator for Mature Agile and Lean Software Development Projects, pp. 93–107.