Factors Impacting Software Release Engineering: A Longitudinal Study

Noureddine Kerzazi
Dept. Research & Development, Payza.com
Montreal, Canada
noureddine@payza.com

Foutse Khomh
SWAT, École Polytechnique de Montréal
Montreal, Canada
foutse.khomh@polymtl.ca

***Abstract*—Software release teams try to reduce the time needed for the transit of features or bug fixes from the development environment to the production, crossing all the quality gates. However, little is known about the factors that influence the time-to-production and how they might be controlled in order to speed up the release cycles. This paper examines step by step the release process of an industrial software organization aiming to identify factors that have a significant impact on the Lead Time and outcomes of the software releases. Over 14 months of release data have been analyzed (246 releases from the isolated source code branches to the production environment). We discuss three dimensions under which a series of factors could be addressed: Technical, Organizational, and Interactional. We present our finding in terms of implications for release process improvements.**

***Index Terms*— Empirical Software Engineering, Knowledge management, Software Process, Software Quality, Entropy.**

I. Introduction

There is a trend to reduce the release cycle from months to weeks or even days [1]. When the release process is well controlled (i.e., repeatable) and smooth (i.e., automated when possible), organizations can afford short release cycles. The fact is more evident in the context of wed based applications. However, the scope of release team activities is large: activities range from source code merging between branches, crossing all automated tests, building and packaging the final application, coordinating with other individuals (DBA, IT, etc.), and finally pushing the application to the production servers.

We have observed many times, team members bugging the performance, security, or builds at the last minutes of the release sprint. For instance, integration of parallel changes is error prone [2]. Release issues are not only affecting the current release, but also blocking the upcoming releases, which consequently decreases the capability of delivering values to the end users. Organizations have little information available to assess the effectiveness of their release process.

Determining the factors that impede the release process is arguably the most challenging issue faced by the release engineering field today. It can be helpful to understand the practices, tools, and coordination that are needed to improve the delivery process. Hence, we conducted a longitudinal study to examine 246 releases in a large-scale development context. We analyzed data and observed the release team in action to identify the kinds of problems they face and the extent to which their release process can be improved. The main goal of this paper is to empirically examine the key factors impacting the software release process.

There are many factors that are claimed to have a potential effect on the release process [3]. Some of these factors have been validated empirically [1, 4]. In this paper, we explore the potential impact of technical, organizational, and interactional factors on the lead time of the release process. Technical factors include source code merging and integration, automated tests, and packaging of the application. The organizational factors include the functional dependencies, the design of branching structures, the planning of releases, and the management of branches (syncing). Interactional factors concern aspects such as *(1)* the coordination with developers to fix merge issues, the coordination with architects to resolve performance issues, with database administrators to run scripts at each level reached by the code, and also the coordination with the IT department; *(2)* and socio-technical congruence.

The paper is organized as follows: Section 2 presents the context and describes the research method used to collect data. Section 3 summarizes the factors that have been asserted as influencing the software release outcomes in terms of Lead Time and failures. Section 4 presents the results and limitations followed by our conclusion in section 5.

II. Method

A. Context

The study takes place in a large industrial organization dedicated to the development of a complex financial system. Its web-based products are used in 192 countries. We had the opportunity to be on site for an extended period of time (more than 14 months). The system was composed of 1.5 million lines of code organized in 8,524 source code files. The development team is distributed across two different sites located in Canada and India with a centralized release team. We were aiming to identify factors that negatively impact the release process Lead Time. This goal lead to the following research questions:

RQ1: *What are the factors impacting the release engineering process?*

RQ2: *What is the impact of each factor on the Lead Time of releases?*

B. Data Collection

We collected data based on information pertaining to 246 releases recorded in the release calendar of the company. Figure 1 shows the number of releases grouped by month. Data included timestamps, brief description of the content, main list of features, site location, and the revision tag in the Software Configuration Management system (SCM). We have traced back (timestamp) each release from the *PreRelease* branch to the branch where the changes have occurred. Data extraction was automated thanks to the collaborative system in place, namely TFS. After mining data from the SCM, we decided to exclude 41 releases because the source code was modified directly within the *PreRelease* branch and consequently considered as data outliers that might skewed the Lead Time computation.

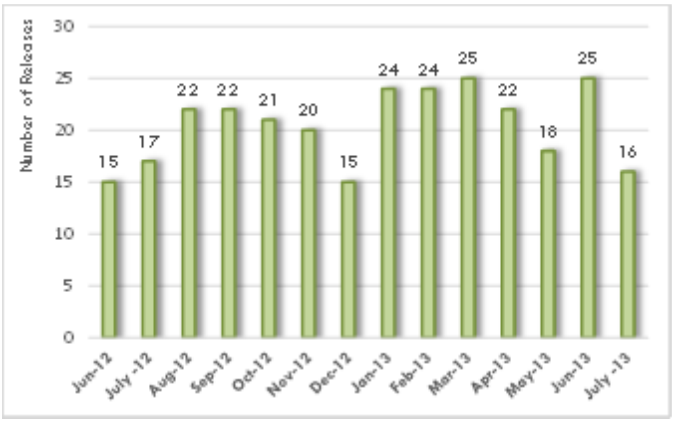

Fig. 1. Distribution of Releases by Month.

For each release, we have traced back the branch from which the code was released and computed the diff between the timestamp when moving the code from that branch towards a releasable branch (*PreRelease)*, crossing the Trunk branch as shown in figure 2. Other relevant information came from work items within the collaborative system such as bugs and reports of test activities.

C. Release Process Overview

This section presents an overview of the release process as it is conducted in the company. As shown in Figure 2, the software development process is parallelized; teams of developers work in parallel on code isolated within separate branches. The branches are recurrently synchronized with the main branch (Trunk). Once the development is completed and tested (manually) within a branch, the release process starts. Release team carry out a forward integration (*FI*) from Trunk to that branch aiming to resolve integration conflicts within the branch instead of Trunk. *FI* ensures stability in the main stream branch. Following that step, Release team run a collection of integration tests that evaluates the recent integrated features as well as regression tests. It's worth noticing that in the meanwhile, Trunk is frozen. When QA team gives the green light, release team carry out a backward integration (*BI*), from the branch to the Trunk. Because the Trunk is always unstable due to the integration work, the release team cannot release from that branch. Another reason relates to the fact that the release team has to consolidate a package, code coming from different branches, before the release. Code is stabilized within the Trunk branch and moved to the prerelease branch. The code is precompiled in the prerelease branch and regression tests are triggered. QA carry out smoke tests on the staging environment with real databases and configurations close to those from the production. Finally the packages are pushed to the production environment.

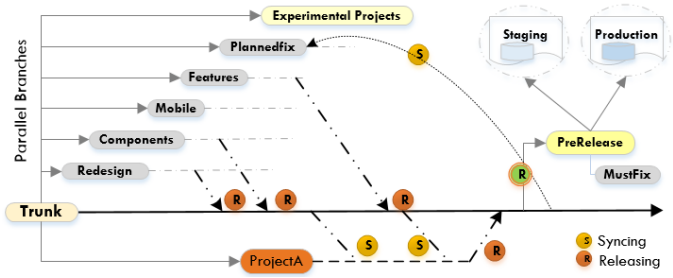

Fig. 2. Exemplified Branching Strategy

III. RQ1: What are the Factors Impacting the Release Engineering Process?

The first step of our investigation towards the identification of factors that might impact the software releases refers to the process point of view. We sought first to identify the breakdown list of release activities, involved roles, and input/output artifacts. These activities range from the integration of code from an isolated branch, the transit of the source code until the production environment. One can observe that a number of responsibilities overlap. For instance, after merging an isolated branch to the mainline, the release team must wait for the results of the integration tests performed by QA team. We extract the following factors based on their impact on the duration of the release process:

A. Technical Factors

a) Merges and Integration

Merges and integration depend on the magnitude of the release [5]. We define the magnitude of a release as the distance, in terms of source code changes, between the trunk and the branch to be released. This distance can be expressed with: *(1)* the size of changes (measured with Churn metrics [6]), and *(2)* the complexity of the changes (measured with concentration of dependencies [7]). While Churn metrics give an idea about the size of the release, it is not sufficient to predict the integration efforts and potential merge issues. For instance, adding a large amount of new code is less risky than changing a method signature. Hence, dependency metrics are required to explore the amount of effort necessary to integrate the code. We hypothesize that the magnitude of a release influences the Lead Time as well as the product quality.

b) Testing

Test activities are time consuming. While unit and regression tests are automated, we still do have manual tests for the newly integrated features. Even most of test activities are carried out by QA team, the release team should wait for the green light

before moving from a branch to another. Manual testing is not supposed to be part of the release process. However, when bugs are found during the release sprint, QA members get involved to fix these bugs (through smoke tests). Furthermore, Technical dependencies makes it difficult to trust partial tests of the system. After each change (even small), the entire system should be re-tested. Consequently, shorter release cycles depend on shorter testing periods [1].

c) Packaging the application

Packaging refers to the pre-compilation, bundling of binary resources, and the preparation of configuration files. In contrast to the normal compilation carried out by developers, the pre-compilation aims to enhance the performance and security of the source code within the production environment. Pre-compilation is more restrictive than a normal build, which might need code adjustment. Bundling binary resources refers to installation of packages and APIs that the code depends on. These packages are generally available in a public binary repository (e.g., *NuGet*, *Artifactory*). Finally, some releases might need a specific setup which is prepared by the release team.

B. Organizational Factors

a) Functional Dependencies

We have seen many times release team releasing source code without knowing what the code does. The link between technical elements, under the released and functional work items (e.g., Projects, Features, and Bug fixes) should be described in release notes.

b) Design of an Adapted Branching Structure

While developers construct parts of the application, release engineers have to build the pipeline to deliver these parts to the end-users. Thus, having an adequate branching structure is crucial [6, 8, 9]. However, there is no recipe for a good branching structure. We extract a list of principles stated by the release team in order to support the design of an effective branching structure adapted to the context of the organization:

- ***P1***: Have a releasable branch at any time.
- ***P2***: All changes have to go through QA gates.
- ***P3***: Isolate the code not people.
- ***P4***: Source code must transit by merges never by copy/paste.
- ***P5***: Do not freeze the development.

In an ideal situation, a good architecture is to align branching structure with architectural components and then organize teams to work in isolated manner on components within dedicated branches [10]. However, this ideal situation is not possible with layered systems such as web-based systems. The changes, required to develop a new feature, could be scattered in different branches leading to integration failures when it comes to releasing that feature.

c) Release Planning

Releases planning is often underestimated. For instance, a feature can be offered as part of a release only if all its necessary tasks are done before the release date [11]. Hence, the importance of a good release planning. We have observed cases of releases that were blocked because of incomplete interdependent technical elements.

C. Interactional Factors

a) Coordination

Task dependencies drive the need to coordinate work activities [12]. Coordination arises as a response to those questions such as who should do what, when is it required, what approval is requested for which activity, and who should be informed [13]. The effect of coordination goes beyond the boundaries of development teams. Yet, it is often overlooked or neglected when analyzing the release processes. In our context, coordination involves Database administrators (DBA) who are responsible for running scripts in databases related to each stage (e.g., branch, regression, staging, and production), Business analysts (BA) who keep tracking on their ongoing projects, testers (QA) who should be notified when edits have to be tested on some branches, and finally developers who should resolve merge conflicts or help figure out problematic situations in the production environment.

b) Socio-Technical Congruence

Socio-Technical Congruence (*STC*) refers to the alignment between the technical dimension of work and the social relationship between team members [12]. It has been observed that release engineers not only have to coordinate with other teams, but also should exhibit matching skills when interacting with other members. For instance, resolving performance issues happening in production needs *STC* with architects and DBAs. In this paper, we only present an in-depth analysis of technical and organizational factors because of the space limitation.

IV. RQ2: What is The Impact of each factor on the Lead Time of releases?

The Lead Time of the release process is largely impacted by test activities. Although test activities are not supposed to be part of the release process, these activities are included in the process when computing the Lead Time because they are performed after the transition of code from one branch to another (e.g., integration test within the mainline branch).

Figure 3 shows that 86% of the release time is consumed by both manual and automated tests. Testing activities threaten to become the bottleneck of the release process. In fact, because of the often poor description of functional dependencies, release team usually triggers all the regression test cases every time that a change is performed. With a good knowledge of the functional dependencies, the release team will be able to execute only a subset of the test cases, which will considerably reduce the testing time.

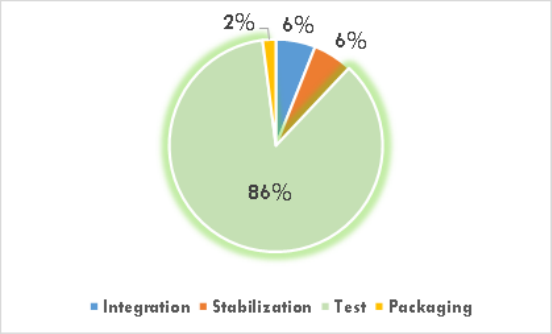

Fig. 3. Repartition of the Effort in the Release Process

Moreover, the computation of the merge effort involves less overhead as compared to tests (6%). We found that the merge effort is correlated with the stabilization effort (6%). Stabilization refers to the code adjustments after merging the source code between two branches. The more the merge effort is large, the higher is the stabilization effort.

A. Impact of Technical Factors

Figure 4 shows the amount of files impacted by each release. On average, 142 files (SD = 326.68) are changed for each release. **The duration of merges and integration depends not only on the extent of changes made in the isolated branch, but also on the flow of changes crossing the main branch (Trunk).** Further investigation into the concentration of dependencies provides more accurate estimation of the merge duration. Figure 5 illustrates a real example of the transition of churn metrics between the Trunk and a branch. The example illustrates 3 forward merges; the first one containing 443 files with a code churn of 14,306. After three forward merges that kept the branch in relatively sync with the Trunk, a release happens. 76 files have been merged in the trunk with a code churn equal to 3,454. Resulting in a large effort to keep the branch synchronized. This effort is necessary to avoid teams facing complex and risky big-bang merges afterwards.

Excluding 20 min to run the unit tests plus 54 min to run regression tests, the rest of the time is allocated to manual testing. When tests are not conclusive, developers are involved in a costly sequence of fixing/re-testing. The release team tries to avoid this situation and recommends to always finishing the testing in the branches before moving forward to release.

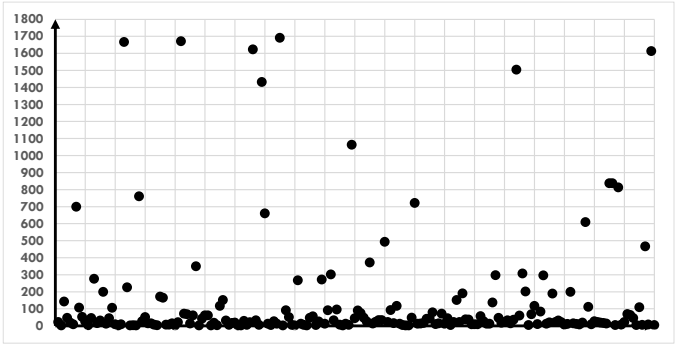

Fig. 4. Number of Files Impacted by the Releases.

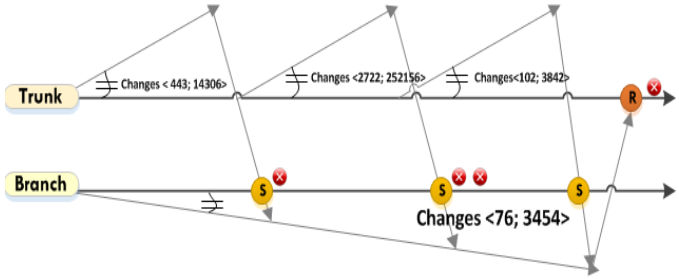

Fig. 5. Propagation of Changes between Branches through Time.

Release team tried to speed up the process by cutting down the effort of tests. To do so, team attempted to consolidate a single package, within the Trunk branch, fed by the code from different branches. The situation was worse than the previous because the integration and code stabilization took more time than expected respectively (15% for integration and 40% for stabilization), the pipeline of release was blocked. The team goes into a vicious circle of bugs' identification, correction, and re-test. In other words, integration tests of changes that come from different branches might be a challenging task. Previous work indicated the importance of the size of the changes on the product quality[14]. **We claim that in the context of parallel development, it's more valuable to release smaller and often. Further analyses are required for more evidence.**

B. Impact of Organizational Factors

We found that over 20% of the release time is allocated to the organizational dimension. First, while release team are dealing with source control *ChangeSets* and versions, BA team deals with features and Bugs. Release team (RT) has to find efficient ways to map the *ChangeSets* to Features and Bugs descriptions. Moreover, there is a need to identify which parts of the system are affected by the release. Second, code can be committed in an isolated deep branch. RT have to move the code toward the releasable branch taking care of its technical dependencies. Branching structure has an impact not only on the transit time of the code, but also on the amount of errors injected while merging. Third, daily strategic planning helps to set priorities and ensure that members are working toward a common goal.

C. Impact of Interactional Factors

Coordination in release activities is a crucial task [15]. From a process point of view, we observed that the release team coordinates with other roles: Developers, Integrators, Testers, Database Administrators, Architects, IT support, and Business Analysts. These coordination activities are embodied in the release process, and consequently, could affect the overall Lead Time of releases. Due to space constraints, we focus only on the interaction with Testers. We consider two levels of interaction: Direct and Indirect. For instance, direct interactions happen between the Release Team and Testers to get the green light to move to the next step of the release process, while the indirect interactions happen between testers and developers for code stabilization. Further analysis of indirect interaction reveals that the release team loses the control of the process making it harder to coordination the back and forth

interactions between testers and developers. This finding might explain the high amount of time attributed to the tests activities. In future work, we will perform a more detailed analysis of a release team's network to measure the effects of emergent interactions on the release team's productivity and product quality.

D. Limitations and Threats

Since the results of this study are obtained from a single company, we cannot assume the generalization of our findings. Concretely, the release process activities in the context of this company might be different to other contexts, meaning that there is a possibility that the challenges faced by the studied release team do not occur within other organizations. Nevertheless, we believe that these findings constitute a significant addition to the body of knowledge [16] about factors impacting the software release practices. Data along with observations have been collected throughout a long time interval (over 14 months) in a large industrial company.

Another limitation lies in the categorization and classification of the studied factors. This taxonomy of factors was inspired by our previous analysis of the release process activities [15] and previous works (e.g., [4]) about integration failures.

V. CONCLUSION

This study examines the factors impacting the software release engineering process in terms of Lead Time. The contribution of this paper to the software engineering literature is twofold. First, we set out the factors affecting the release engineering field according to three dimensions: technical, organizational, and interactional. Such structuration of the factors allows further analysis. For instance, there is little research related to the collaboration of release teams with other teams. Second, our findings provide empirical evaluations of eight factors on the release time.

We identified 3 factors pertaining to the technical dimension: Merges & Integration; Tests; and Packaging. Three factors related to the organizational dimension: Functional-dependencies; branching structures; and release planning. Our analyses reveal that tests are the most time consuming activities (86%). A lot of improvement has been done with continuous builds, binary packages bundling, and regression testing. Release engineers need more tools and practices to implement smart automated tests in order to enhance the Lead Time of software releases. This paper also illustrated the need for more congruence among teams, especially in the context of parallel development.

REFERENCES

[1] F. Khomh,T. Dhaliwal,Y. Zou*, et al.*, Do faster releases improve software quality? An empirical case study of Mozilla Firefox. in *MSR'12*, 179-188, 2012.

[2] D.E. Perry,H. Siy and L. Votta Parallel changes in large-scale software development: an observational case study. ACM Trans. Softw. Eng. Methodol., 10 (3): 308-337, 2001.

[3] J. Tsay,H.K. Wright and D.E. Perry. Experiences mining open source release histories Proceedings of the 2011 International Conference on Software and Systems Process, Waikiki, Honolulu, HI, USA, 208-212, 2011.

[4] M. Cataldo and J.D. Herbsleb. Factors leading to integration failures in global feature-oriented development: an empirical analysis. ACM ed. 33rd International Conference on Software Engineering, Waikiki, Honolulu, HI, USA, 161-170, 2011.

[5] A.E. Hassan and K. Zhang, Using Decision Trees to Predict the Certification Result of a Build. in Automated Software Engineering, 2006. ASE '06. 21st IEEE/ACM International Conference on, 189-198, 2006.

[6] N. Nagappan and T. Ball, Using Software Dependencies and Churn Metrics to Predict Field Failures: An Empirical Case Study. in 1' Int Symp on Emp Soft Eng and Measurement, 363-373, 2007.

[7] M. Eaddy,T. Zimmermann,K.D. Sherwood, et al. Do Crosscutting Concerns Cause Defects? IEEE Trans. Soft. Eng., 34 (4): 497-515, 2008.

[8] C. Bird and T. Zimmermann, Assessing the value of branches with what-if analysis. in 20th Int Symp on the Foundations of Soft Eng, (Cary, North Carolina), 2012.

[9] E. Shihab,C. Bird and T. Zimmermann, The effect of branching strategies on software quality. in Int'l Symp on Emp soft. Eng.and Measurement, (Sweden), 301-310, 2012.

[10] N. Nagappan,B. Murphy and V.R. Basili, The influence of organizational structure on software quality: an empirical case study. in ICSE, (Leipzig, Germany), 521-530, 2008.

[11] A. Ngo-the and G. Ruhe Optimized Resource Allocation for Software Release Planning. IEEE Trans on Soft Eng, 35 (1): 109-123, 2009.

[12] M. Cataldo,P.A. Wagstrom,J.D. Herbsleb, et al. Identification of Coordination Requirements Implication for the design of collaboration and Awareness Tools Computer Supported Cooperative, Alberta, Canada, 353-362, 2006.

[13] R.E. Kraut and L.A. Streeter Coordination in Software Development. ACM, 38 (3): 69-81, 1995.

[14] A. Mockus and D.M. Weiss. Understanding and predicting effort in software projects Proc of the 25th International Conference on Software Engineering, Portland, Oregon, 274-284, 2003.

[15] N. Kerzazi and P.N. Robillard. Kanbanize the Release Engineering Process 1st International Workshop on Release Engineering, San Francisco, CA, USA, 9-12, 2013.

[16] V.R. Basili,F. Shull and F. Lanubile Building Knowledge Trough Families of experiments IEEE Trans on Soft. Eng., 25 (4): 456-473, 1999.