Factors Impacting Rapid Releases: An Industrial Case Study

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

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

ABSTRACT

Context: 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.
Goal: 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.
Method: Over 14 months of release data have been analyzed (246 releases from the isolated source code branches to the production environment).
Results: We discuss three dimensions under which a series of factors could be addressed: technical, organizational, and interactional. We present our findings in terms of implications for release process improvements.
Conclusions: Our analyzes reveal that testing is the most time consuming activities (86%) along with the need for more congruence among teams, especially in the context of parallel development.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Empirical Software Engineering, Release Engineering, Packaging, and Production.

General Terms

Software Release, Parallel Development, Rapid Release, Lead Time

Keywords

Empirical Software Engineering, Release Management, Software Process, Software Quality, Release Cycles, Lead Time.

1. INTRODUCTION

There is a trend to reduce the release cycle from months to weeks or even days [10]. 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 web 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 (Database Administrators, Testers, 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 a release sprint. For instance, integration of parallel changes is error prone [18]. 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. Therefore, determining the factors that impede the release process is one of the most challenging issue faced by the release engineering field today. Software organizations need to understand the practices, tools, and teams' coordination strategies that are needed to improve the software delivery process.

This paper reports on the results of a longitudinal study that examined 246 releases of a large-scale web-based software system. We analyzed data and observed the release team in action in order to identify the kinds of problems they face and the extent to which their release process can be improved; our main goal was to empirically examine the key factors impacting the software release process.

To this end, we have investigated the impact of technical, organizational, and interactional factors on the lead time of the software 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 rest of the paper is organized as follows: Section 2

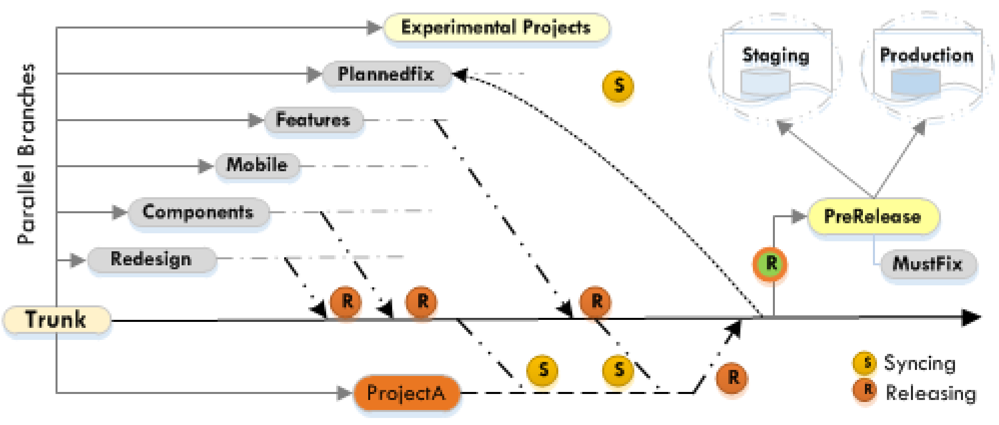

Figure 1: Exemplified Branching Strategy.

presents the context of our study and describes the research method used to collect data. Section 3 presents the results of this study and summarizes the factors that have been found to affect software release outcomes, in terms of lead time and failures. Section 4 discusses the lessons learned and the practical implications of our results. Section 5 outlines the threats to validity of the study. Section 6 discusses related work, and finally, Section 7 concludes the paper and presents an outlook of future work.

2. STUDY DESIGN

This section describes the context, the workflow of activities carried out to push the source code from the development environment to the production, and the data of our study.

2.1 Context

The study takes place in a large industrial organization dedicated to the development of a web based financial system used in 192 countries. We had the opportunity to be on site for an extended period of time (i.e., 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.

2.2 Release Process Overview

This section presents an overview of the release process as it is conducted in the company. As shown in Figure 1, the software development 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 within a branch, the release process starts. Typically. the release team carries out a forward integration (*FI*) (see Figure 2) from Trunk to the branch aiming to resolve integration conflicts within the branch instead of Trunk. *FI* ensures stability in the main stream branch. Following that step, the release team runs a collection of integration tests that evaluates the recent integrated features as well as regression tests. It is worth noticing that in the meantime, the trunk is frozen. After the successful completion of these tests (the green light of the QA team is obtained), the release team carry out a backward integration (*BI*), from the branch to the Trunk. The code is stabilized within the Trunk branch and moved

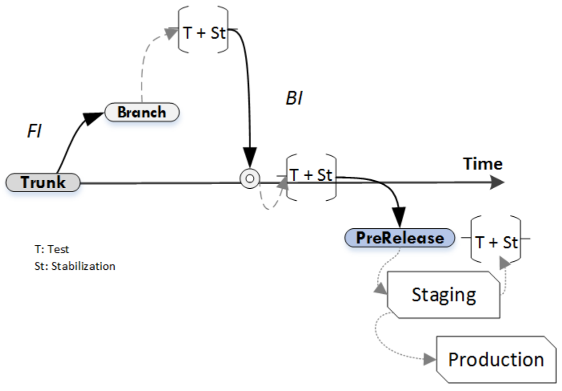

Figure 2: source code Transit from branches to production.

to the *PreRelease* branch. Figure 2 presents an illustration of how the code transits through the trunk during a release. In the *PreRelease* branch, the code is precompiled, packaged and regression tests are triggered. The QA team then carries out smoke tests on the staging environment characterized by a set of configurations close to those used in the production environment along with using real databases. Finally, the packages are pushed to the production environment.

The release team does not release from Trunk because Trunk is always unstable due to frequent integration works, and also because the release team usually has to consolidate multiple packages and code coming from different branches before they can release.

2.3 Data Collection

We collected data based on information pertaining to 246 releases recorded in the release calendar of the company. Figure 3 shows the distribution of the number of releases per month. The collected data included timestamps, brief description of the content of releases, main list of features and bugs within the tracking system, 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 occurred. The data extraction process was automated thanks

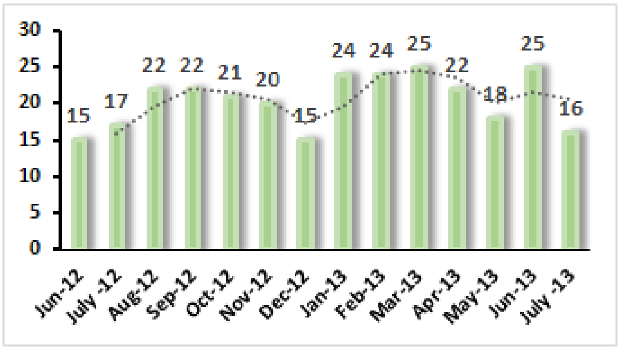

Figure 3: Distribution of Releases by Month.

to the collaborative system in place, namely Microsoft 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.

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 1. We also collected information about the work items that were performed to fix bugs and test the system. Using the collected data, we answer the following research question.

RQ: What are the factors impacting the Lead Time of releases?

3. CASE STUDY RESULTS

In this section, we present and discuss the answers to our research question. First, we present the motivation behind the question, then our analysis approach, and a discussion on our findings.

3.1 RQ: What are the factors impacting the Lead Time of releases?

3.1.1 Motivation

We set out to answer a question we were asked by the software release management of the company: How can we speed up the release process within a parallel development context and how can we make it more predictable? To answer this question we have to identify the factors that impact the Lead Time of edits transits from branches to the production environment, and then we will be able to suggest guidelines for the release process improvement.

3.1.2 Approach

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 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, crossing all the quality gates. 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 have performed a reverse engineering of the releases and extract the following factors based on their impact on the duration of the release process:

A. Technical Factors

We investigate the following technical factors:

- **Merges and Integration:** merges and integration workload depend on the magnitude of the release [8]. 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 [14]), and *(2)* the complexity of the changes (measured with concentration of dependencies [6]). While Churn metrics provide an idea about the size of the release, it is not sufficient for predicting 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 different pieces of source code. We hypothesize that the magnitude of a release influences the Lead Time as well as the product quality.

- **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 though 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 track 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 [10].

- **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

We investigate the following organizational factors:

- **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.

- **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 [2, 14, 20]. 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 organization of parallel development try to align branching structure with architectural components and then organize teams to work in isolated manner on components within dedicated branches [15]. 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.

- **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 [17]. 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

We investigate the following interactional factors:

- **Coordination:** task dependencies drive the need to coordinate work activities [5]. 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 [12]. 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.

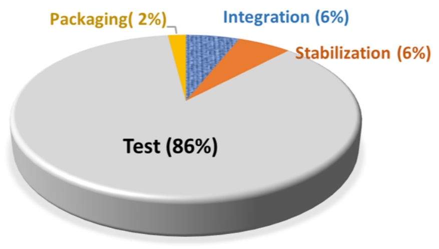

Figure 4: Repartition of the time consuming.

- **Socio-Technical Congruence:** socio-Technical Congruence (STC) refers to the alignment between the technical dimension of work and the social relationship between team members [5]. 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.

3.1.3 Findings

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 4 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.

Moreover, the computation of the time spent on merge activities shows that merges account for only (6%) of the release time, which is far less than the time spent on testing. We also found that developers spend similar amount of time on the stabilization of the code (i.e., 6%). Stabilization refers to the code adjustments after merging the source code between two branches. We observed that the more the merge effort is large, the higher is the stabilization effort. Packaging activities represent 2% of the release time.

A. Impact of Technical Factors

Figure 5 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 (i.e., Trunk).** Further in-

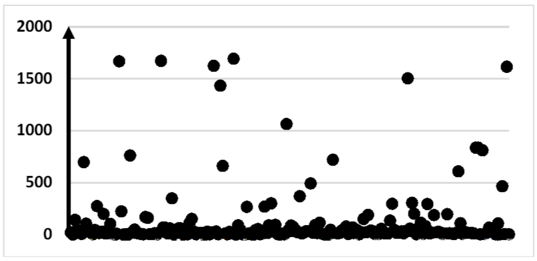

Figure 5: Number of Files Impacted by the Releases.

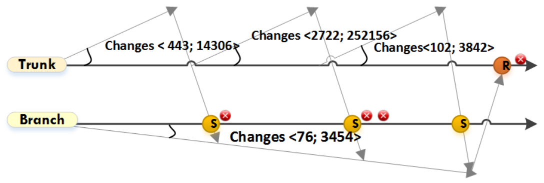

Figure 6: Propagation of Changes between Branches through Time.

vestigations into the concentration of dependencies provides a more accurate estimation of the merge duration. Figure 6 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 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 finish the testing in the branches before moving forward to release.

Release team tried to speed up the process by cutting down the effort of tests. To do so, the 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 went 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 [13]. **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. The release team 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. The release team has to move the code toward the releasable branch taking care of its technical dependencies. The 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 [9]. 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. An analysis of indirect interactions has revealed that the release team often loses control of the process, making it harder to coordinate the back and forth interactions between testers and developers. This finding might explain the high amount of time attributed to testing activities. In future work, we will perform a more detailed analysis of a release team network to measure the effects of emergent interactions on the release team's productivity and product quality.

4. LESSONS LEARNED

In this research work, we set out to identify bottlenecks in software release processes, in order to enable release process improvements. This section describes some of the insights we gained over 14 months of observation and then discusses several implications for software practitioners.

Defense in Depth test- We have observed that functional dependencies due to cross-feature interactions have an impact on the integration failures which in turn affect the endeavor of tests [4]. Bug tracking has to pass through multiple layers of defenses (e.g., Compilers, Static analysis, Code Review, Dynamic Analysis, Unit testing, Integration testing, Regression testing, Dogfooding testing, etc.). For instance, code review practices can prevent logical flaws that might affect the system performance or cause security issues.

Continuous testing- promotes the same ideas of continuous integration practices to test activities. Continuous testing is intended to reduce the time and overhead to keep source code well-tested, especially in the context of parallel development [19]. We believe that continuous testing have the potential to significantly reduce the overhead cause by testing activities (that we reported in Section 3), hence we recommend that, similar to continuous integration, automated regression tests be performed in the background, on developers' workstations after committing changes (i.e., continuous testing).

Automate or drown- When organizations grow and face the scalability barrier, they have to adjust their release process, so it is either Automate or drown. Automating deployment makes the process predictable and lowers the risk involved with each push. Unit and Regression testing must not be only automated as much as possible, but optimized to run in a reasonable time.

Enhance teams' interaction beyond boundaries- We have observed that a higher degree of interaction between releasing, testing, and development teams is required. On multiple occasions, we observed the release team losing control of the release process because of poor communications between QA members and developers; this often resulted in long delays in the release (from hours to days).

Design of collaborative tools- One aspect that we found to impact release time is the lack of good overview of the release process. To alleviate this aspect, we recommend that release teams make use of tools that enable the visualization of the release flow and increase the awareness of release team members beyond the traditional boundaries.

5. THREATS TO VALIDITY

The internal validity threats of this study are related to the data extraction process. While we have provided details on the data extraction and filtering used in this study, our results may be affected by issues related to time overlaps, especially when the testers find bugs and assign them back to developers. We validated our findings with the company developers, testers, and release engineers by interviewing individuals from the company and received insights into their perception of the release process [11]. Another limitation of our work lies in the subjectivity inherent to our categorization and classification of the studied factors. Nevertheless, this taxonomy of factors was inspired by our previous analysis of the release process activities [9] and previous works (e.g., [4]) about integration failures.
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 [1] 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.

6. RELATED WORK

Previous research has suggested a number of factors that can influence software release cycle and outcome.

Technical Factors- Brun et al. [3] studied the collaboration conflicts (i.e., integration & merge issues) of nine open source systems and reported that the conflicts are the norm rather than the exception. Authors discovered that 16% of merges required human effort to be resolved, that 33% of merges that were assessed by the version control system as non conflictual in fact contained higher-order conflicts, and that on average conflicts persist for 10 days (median = 1.6 days). Furthermore, Cataldo and Herbsleb [4] examined the impact of technical attributes (e.g., Churn metrics of changes, concentration of changes, number of dependencies) on the integration failures in a large scale global software development project. The authors observed that the number of architectural dependencies among the system components impact the software quality. The higher number of architectural dependencies, the higher the likelihood of integration failures.

Organizational Factors- Existing research has found that organizational structure can influence software quality. For instance, Nagappan et al. [16] claim that organizational complexity influence quality. Authors showed usage of organizational metrics, such as *(i)* organizational distance between developers and *(ii)* number of developers working on a component, among others can be better predictors of defect proneness than traditional metrics such as churn, complexity, and dependencies.

We have explored the organizational dimension and found that we are inline with the importance of organizational dimension. First, as stated by many authors [2, 20], branching structure plays an important role in the development process of large software. While branches allow source code isolation and thus support the modification of different pieces of code in parallel, they add more complexity to the code integration and the transit of edits from a branch to the production environment. Second, a feature can be offered as part of a release only if all its necessary tasks are done before the given release date [17]. This functional dependency affects not only the planning, but also the test effort because the overall system has to be re-tested after each new feature integration. Third, the volatility of release planning is a real problem [9]. Because of the inherent uncertainty, it requires a daily meeting to negotiate and prioritize the release flow. Releases planning and resources allocation are time-consuming activities, which cannot be handled in isolation.

Interactional Factors- In recent years, socio-technical coordination and congruence aspect has attracted the attention of many researchers. In this relatively new field, researchers emphasize the interactions between different roles involved in a software development project. Cataldo et al. [5] reported on their work on calculating the coordination needs within development teams. They build on task dependency and communication to define the coordination needs. Authors also examined the impact of congruence on task performance. In the same way, Gokpinar et al. [7] applied a congruence technique and discovered that a large gap in coordination leads to a larger number of bugs reported, which means a decrease of the software quality. Since the release management is about building a pipeline of software releases, it obviously requires communication and coordination with almost all the roles involved in the software production.

Despite the importance of socio-technical coordination during software release processes (as shown by the results of this study), so far, the academic research community has given little attention to how to improve the specific coordination between release engineers and other actors involved in the development process, and the extent to which we can enhance the socio-technical congruence between them.

7. CONCLUSION AND FUTURE WORK

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 structuring of the factors allows for further analysis. For instance, there is little research related to the collaboration of release teams with other teams. Second, this longitudinal study provides 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 analyzes reveal that testing is 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.

Finally, some challenges, such as test activities in parallel development, need to be addressed and studied more closely. We have received very positive feedback on the results of this case study from the management of the company, which invite us to focus, as further work, on examining modern testing activities, upstream from the development activities such as code review, the institutionalization of nightly regression tests on the developers workstations or on branches and so on. Moreover, as future work we shall replicate the analysis of the release process in other contexts.

8. ACKNOWLEDGMENTS

The first author is working as Release Manager for a North American company, developing a complex web-based solution for an online financial system. Thanks to Sandeep Sandhu QA Team Lead, and Wilton Godinho development Team Lead, for their useful inputs. This work has been partially funded by the Natural Sciences and Engineering Research Council of Canada.

9. REFERENCES

[1] V. Basili, F. Shull, and F. Lanubile. Building knowledge through families of experiments. *Software Engineering, IEEE Transactions on*, 25(4):456–473, 1999.

[2] C. Bird and T. Zimmermann. Assessing the value of branches with what-if analysis. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, FSE '12, pages 45:1–45:11, New York, USA, 2012.

[3] Y. Brun, R. Holmes, M. D. Ernst, and D. Notkin. Proactive detection of collaboration conflicts. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ESEC/FSE '11, pages 168–178, New York, NY, USA, 2011. ACM.

[4] M. Cataldo and J. D. Herbsleb. Factors leading to integration failures in global feature-oriented development: An empirical analysis. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE'11, pages 161–170, New York, USA, 2011.

[5] M. Cataldo, P. A. Wagstrom, J. D. Herbsleb, and K. M. Carley. Identification of coordination requirements: Implications for the design of collaboration and awareness tools. In *Proceedings of the 2006 20th Anniversary Conference on Computer Supported Cooperative Work*, CSCW '06, pages 353–362, New York, USA, 2006.

[6] M. Eaddy, T. Zimmermann, K. D. Sherwood, V. Garg, G. C. Murphy, N. Nagappan, and A. V. Aho. Do crosscutting concerns cause defects? *IEEE Trans. Softw. Eng.*, 34(4):497–515, July 2008.

[7] B. Gokpinar, W. J. Hopp, and S. M. R. Iravani. The impact of misalignment of organizational structure and product architecture on quality in complex product development. *Manage. Sci.*, 56(3):468–484, 2010.

[8] A. 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*, pages 189–198, Sept 2006.

[9] N. Kerzazi and P. Robillard. Kanbanize the release engineering process. In *Release Engineering (RELENG), 2013 1st International Workshop on*, pages 9–12, May 2013.

[10] F. Khomh, T. Dhaliwal, Y. Zou, and B. Adams. Do faster releases improve software quality? an empirical case study of mozilla firefox. In *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*, pages 179–188, June 2012.

[11] B. Kitchenham and S. Pfleeger. Personal opinion surveys. In F. Shull, J. Singer, and D. SjÃÿberg, editors, *Guide to Advanced Empirical Software Engineering*, pages 63–92. Springer London, 2008.

[12] R. E. Kraut and L. A. Streeter. Coordination in software development. *Commun. ACM*, 38(3):69–81, Mar. 1995.

[13] A. Mockus, D. M. Weiss, and P. Zhang. Understanding and predicting effort in software projects. In *Proceedings of the 25th International Conference on Software Engineering*, ICSE '03, pages 274–284, Washington, DC, USA, 2003. IEEE Computer Society.

[14] N. Nagappan and T. Ball. Using software dependencies and churn metrics to predict field failures: An empirical case study. In *Proceedings of the First International Symposium on Empirical Software Engineering and Measurement*, ESEM '07, pages 364–373, Washington, USA, 2007.

[15] N. Nagappan, B. Murphy, and V. Basili. The influence of organizational structure on software quality: An empirical case study. In *Proceedings of the 30th International Conference on Software Engineering*, ICSE '08, pages 521–530, New York, USA, 2008.

[16] N. Nagappan, B. Murphy, and V. Basili. The influence of organizational structure on software quality: An empirical case study. In *Proceedings of the 30th International Conference on Software Engineering*, ICSE '08, pages 521–530, Leipzig, Germany, 2008.

[17] A. Ngo-The and G. Ruhe. Optimized resource allocation for software release planning. *IEEE Trans. Softw. Eng.*, 35(1):109–123, 2009.

[18] D. Perry, H. Siy, and L. Votta. Parallel changes in large scale software development: an observational case study. In *Software Engineering, 1998. Proceedings of the 1998 International Conference on*, pages 251–260, Apr 1998.

[19] D. Saff and M. D. Ernst. Reducing wasted development time via continuous testing. In *Proceedings of the 14th International Symposium on Software Reliability Engineering*, ISSRE '03, pages 281–291, Washington, DC, USA, 2003. IEEE Computer Society.

[20] E. Shihab, C. Bird, and T. Zimmermann. The effect of branching strategies on software quality. In *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '12, pages 301–310, New York,USA, 2012.