Kanbanize the Release Engineering Process

Noureddine Kerzazi
Department of Research & Development
Payza
Montreal, Canada
Noureddine@Payza.com

Pierre N. Robillard
Department of Computer Engineering
Polytechnique Montréal
Montréal, Canada
pierre.robillard@polymtl.ca

***Abstract*—Release management process must be adapted when IT organizations scale up to avoid discontinuity at the release flow and to preserve the software quality. This paper reports on means to improve the release process in a large-scale project. It discusses the rationale behind adopting Kanban principles for release management, how to implement these principles within a transitional approach, and what are the benefits. The paper discusses the post-transitional product to assess the status of the outcomes.**

***Index Terms*—Release Engineering, Software Process Improvement (SPI), Kanban process model, Action Research.**

I. Introduction

The testing activity of the source code within an isolated branch must be followed by the release of that code to the end user. Prior to the release of a software product, release engineers must perform a list of activities: for examples, planning and coordination, source code merging, integrations, and build construction. Regardless of the model of the software process adopted, there is often variability in release effectiveness.

Releasing source code for one Application/Team is simpler than releasing code from a large and complex organization. Large organizations have dedicated teams to manage the release cycle [1-3]. Regardless of the software process model, the releases of software products must be done one at a time to minimize system failure and customer impact.

Release manager is a central role within the organization. As an "*Air Traffic Control lead*", the challenge is to know where the check-ins are at any given time and where each they will be landing. New versions need to keep coming in an oiled release flow for feeding the value to the customers.

Observing the anatomic constraint of the mainline code stream, pipeline with one release at time, drives the hypothesis of using Kanban ideas [4] to resolve the release problems. Kanban focuses on visualizing the workflow of releases, limiting the work in progress, and controlling the Lead Time.

This paper reports on a transition of release practices from in-house practices to Kanban principles. The transition has been done in a large-scale software organization developing a complex web-based financial system. The transition, done as a part of software process improvement, was mainly a reaction against the discontinuity in release flow. The contributions of this paper are twofold: *(1)* describing the transition of the release engineering from a set of incorporated activities to a structured process according to the Kanban principles, *(2)* assessing the benefits of this transition on the release flow.

The reminder of this paper is as follows: Section 2 presents the context and motivations behind the migration to Kanban principles (pre-transition). Section 3 describes the transition and the implementation phases. Section 4 outlines the post-transition evaluation. Section 5 concludes our work.

II. Context and Motivations

The R&D department of a North American company develops a web-based solution for an online financial system based on .Net Web technologies and Team Foundation Server (TFS) as a collaborative platform. The organization is aware that the quality and reliability of software systems affect heavily and directly the entire business of the company.

As the company continues growing, the *R&D* department has started moving from an old model of release cycle with a prefixed deadline (2 to 3 months) to a new model based on frequent releases (one release or more by day) [1]. The remainder of this section presents the context of the organization, addresses problems and the motivations for the transition to Kanban principles, describes the implicit process of releasing, and presents our research methodology.

A. Context

Software release practices are heavily tied to three elements, as suggested by the literature: the architecture of the system [5], the structure adopted for the source code branching/merging [6], and the organizational aspects [7]. We start describing the context according to these elements.

The organization works on a web solution that is under development since 2004. The architecture is multilayered, with functionalities scattered across different layers and subsystems, starting from the database at the lowest layer, through middleware and web servers on the server side, to client applications using number of APIs at the highest layer. The layered architecture adds complexity to branching and integrating the source code, which are coming from different branches.

The Release Manager deals with 66 branches organized as shown in figure 1. This branching structure is stable and is suitable for the specific needs of different teams working within different process models (*e.g.,* Scrum, XP, FDD, RUP, etc.). The branching structure was created according to specific needs based mainly on code-dependency, team-dependency, and time-delivery-dependency.

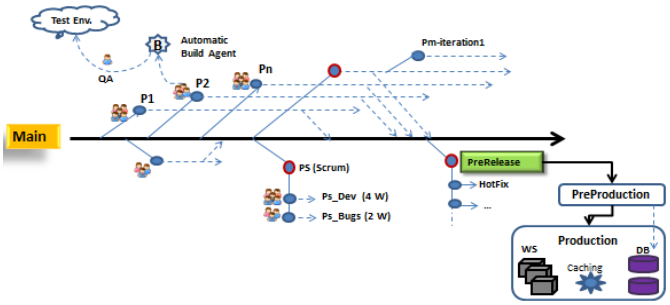

Fig. 1. Structure of Source Code Branching and Path to Production

A development project is branched off into a new branch, where development of new functionalities and system testing are performed. Once the development and test activities are completed[1], the release team merges the code to the main branch in order to integrate the code and to test the integration. Then the code is pushed to the Prerelease branch, where other conflicts might occur when the source codes from different branches are merged. Following that step, quality assurance team (QA) will run regression tests. After building and packaging the source code, coordination with other roles takes place. For instance, we have to coordinate with Database team to run scripts in production environment, with developers to ensure that the configuration parameters will be done in production environment (*e.g.,* frameworks update, specific API, or simply add a key to configuration file). The last step aims at running the software in Preproduction environment which use replicated databases, caching, and other parameters, which are similar to production environment.

B. Motivations

This section lists and describes problems elicited during the pre-transition assessment. It addresses all the problems faced by the organization, although these are not closely related to the release engineering practices in place.

Problem 1: *Lack of visibility on the release progress*. The progress status of a given release is not properly communicated to other stakeholders within or beyond the boundaries of development team (e.g., DB, testers, BA, product owner). For instance, few people know about the state of the current release (e.g., in Prerelease, in Preproduction, etc.).

Problem 2: *Lack of inter-team Coordination*. Coordination problems occur regarding communications between release engineers and other roles. For example, a large number of reported release defects are due to missing database scripts, keys in configuration files, and frameworks or APIs update.

Problem 3: *Poor release planning with low prioritization*. The growing of the organization causes an increased demand for releases (more than one by day). The volatility of release planning becomes problematic. It requires a daily meeting to negotiate and prioritize the release flow.

Problem 4: *Insufficient modularization of the architecture*. Layered and highly coupled architecture increases integration workload. Test efforts become more tedious and time consuming. Since the Integration and Regression tests are included in the release process, the lead time for carrying out releases is often increased.

Problem 5: *Discontinuity in the release flow*. Due to previous problems 1, 2 and 3, release team was often disrupted by resolving integration issues. Consequently, the organization must assume an important and recurrent delay of releases.

Problem 6: *Useless Rework.* As a consequence of the reappearance of already fixed bugs coming from the difficulty of keeping branches in sync and an inappropriate timing for merge back, many useless reworks are required. Due to the problem 3, keeping in sync all branches becomes a nightmare.

Problem 7: *Spaghetti Branching Strategies*. With the absence of conceptual structure of code branching, development teams struggle to find where they have to check-in the source code. Let say a bug has been identified in preproduction environment; it must be clear where this bug should be fixed to avoid blocking the flow of coming releases. Since all edits coming from child branches should transit by the mainline code stream to be released, the main branch becomes a bottleneck.

Problem 8: *A high rate of post-release defects reported by end users*. Although this problem is not always closely related to the process of release, release team has to be aware of the root cause of those defects.

Based on the problems and issues elicited during the pre-transition phase and the benefits of Kanban principles, the organization makes the decision to implement these principles. The next section provides an abstract view of the actual release process. This view is beneficial for the integration of the release practices into a wider development process [5].

C. Modeling the Release Process

In order to share a common understanding about the existing release process, we modeled this process using a process modeling tool [8]. Figure 2 outlines the scope of the process, the steps, and release activities. In the *Branching* phase, the first activity of the release engineering deals with branching strategies. A set of parameters define where and when the source codes have to be branched. For instance, we don't deal with the Hotfixes code in the same way as scrum iteration. The second phase traits the *Pre-Release* activities, which deals with code merging from branches to the mainline code stream, integration, and builds activities. Then the *Release* phase takes place involving running scripts, builds & packaging, and coordination with other roles to push the software to the production environment. Finally, the Post-Release focuses on the product quality and its performance in production environment.

D. Methodology

Action research method has been used to conduct this study. The researchers and organization were engaged in the intended study in action [9] pertaining to software process improvement effort. The researchers participate actively in all the steps of the process transition including diagnosis of the release management practices in place, support for decision making, tooling, suggestion for optimization, and assessment.

[1] Often, there is confusion upon the meaning of "completed". Once the Devs and QAs are done, the functionality has to wait for the release process. The distance between branches and production environment is usually underestimated in terms of time and defect injections.

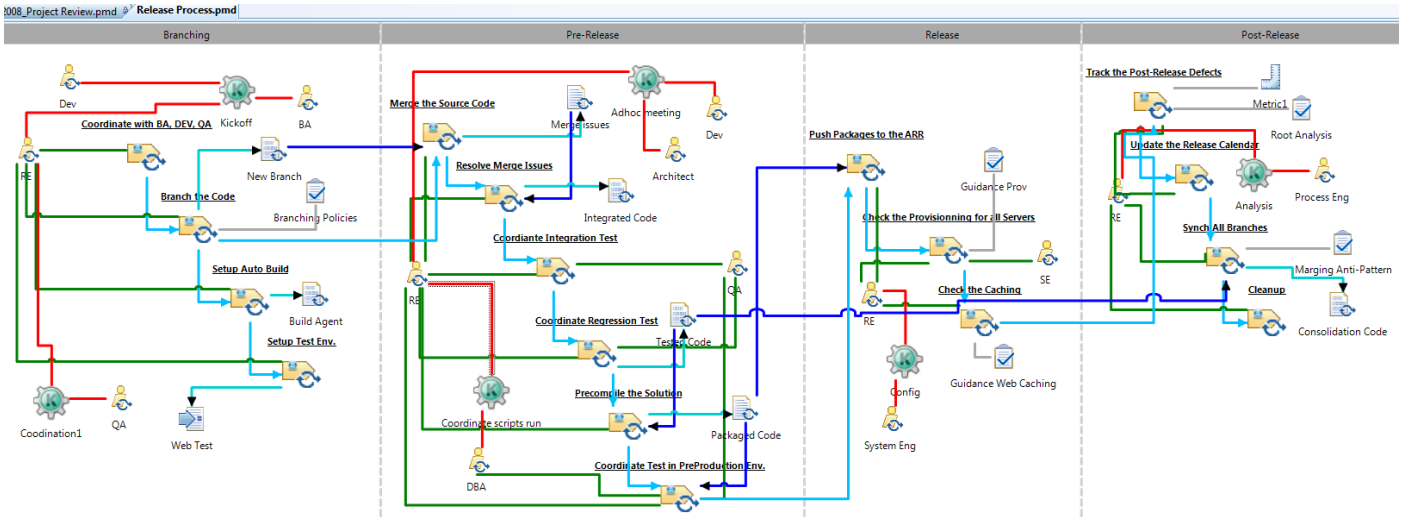

Fig. 2. An Abstract View of the Release Process as Modeled by DSL4SPM tool [8]

III. Transition to Kanban

The hypothesis of this research pertains to the application of lean ideas, encompassed in Kanban process model.

How do Kanban Principles contribute to improve software release process in particular, and the wider development process in general?

A. Kanban Principles

Kanban method has gained momentum recently in the field of software development due to its associations to Lean thinking and its apparent simplicity. Kanban is based on three principles: *(1)* visualization of the workflow, *(2)* limitation of the work in progress WIP, *(3)* measurement and control of the lead time. Kanban principles can be combined with those of Scrum [10].

Visualization of the Workflow- The Kanban board helps to detect potential problems and bottlenecks early [4]. Seeing the whole release process, from branch to production, with a simple Kanban board may be helpful in particular with large and complex system projects. For example, developers can keep track on the transit of their edits.

WIP- In contrast to Scrum which has time-boxed iterations and the work is pushed in to meet the schedule[10], Kanban team limits the WIP to ensure that the members are not overloaded and the work is "pulled" in when the release team have capacity. The team did not sign up for arbitrary deadlines. Hence, arbitrary deadlines are avoided.

Lead Time (LT)- known as the cycle time is an important metric that refers to the total elapsed time between deciding to release a feature and having it in production. This measure tracks how quickly, repeatable and reliable software is delivered to customers [4]. The challenge is to optimize the LT. Having presented Kanban principles, we present in figure 3 the mapping between the problems presented in section 2.B and those principles. The black bold arrows mean a high correlation.

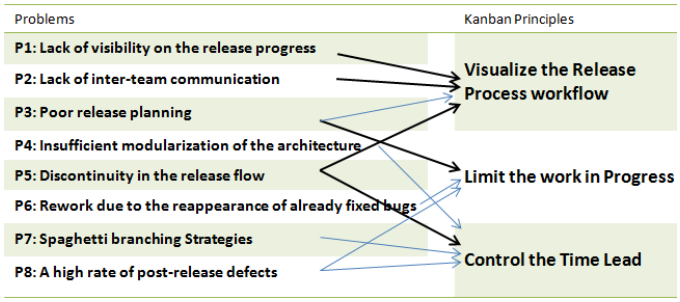

Fig. 3. Mapping Problems to Kanban Principles

B. Visualization of the Release Workflow

The organization uses Team Foundation Server (TFS) as a collaborative platform. TFS is used as a source control repository and system for tagging. Tagging enables to link work items to source code. (*e.g.,* Feature, Bug, etc.). Also, work items in TFS are used heavily for planning purposes.

Figure 4 presents a specific work item, called *Release* (*RWI*), designed for the purpose of planning and to allow the monitoring of each release progress throughout crossing predefined states. When a project is ready to be released, a team member fills the *RWI*. The release team receives a notification and then the process of release starts. One particularity of TFS work items pertains to the ability to configure a set of states that defines the workflow of the item. For our purpose, we add five states for the *RWI*, namely: *BackLog*, *inBranch*, *inPreRelease*, *inPreproduction*, and *inProduction*. These states reflect the release progress status (*i.e.,* Releases flow).

Exploiting the states of *RWI*, we designed a web application that retrieves all active *RWI* from the TFS repository and shows the progress of each release. By doing so, we visualize the releases workflow, their progresses, and the opportunity to measure the Lead Time (*i.e.,* the check-ins transit between a given branch and the Production). Figure 5 shows the Kanban workflow board developed as a Website.

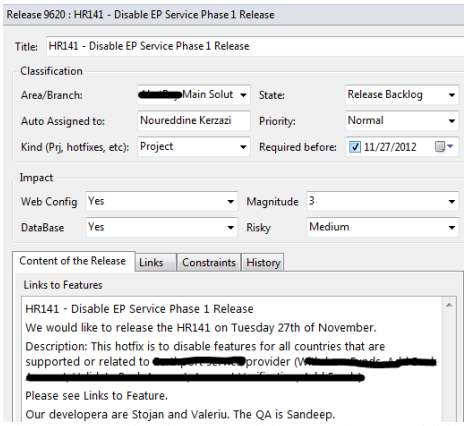

Fig. 4. Example of TFS Work item '*Release*', which followed a set of states from creation to closure. Those transitions represent the release workflow.

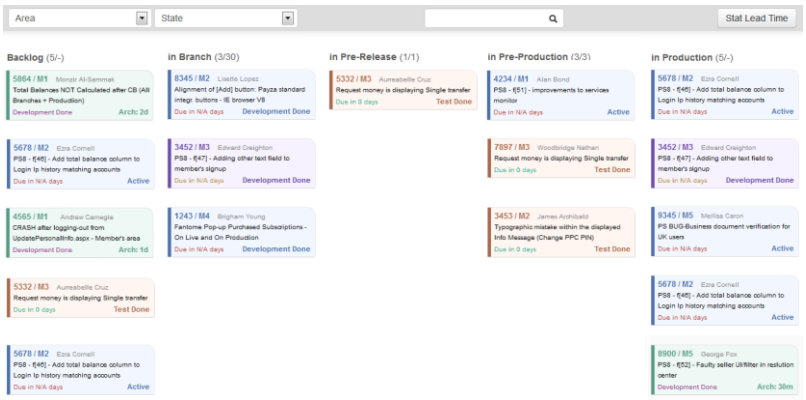

Fig. 5. Kanban Board implemented as a Website to show each release progress. Five Swimlanes representing the states of *Release* work item; Colors indicate the nature of release (Projects, Bugs, Scrum Iteration); M_x refers to the release Magnitude.

IV. Evaluation of Kanban Transition

We performed a post-transition data analysis to assess whether or not intended outcomes are being achieved and to support decisions on maintaining or adapting Kanban for release engineering.

In order to build Kanban board, as shown in figure 5, we developed a web-based .Net Application to query and retrieve the state and data of each *RWI* from the TFS repository. This executive board provides a useful visibility of each release progress and a short-term planning (Problem 1, 2, 3, 5).

Limit of WIP preserves the overheating of release team. Release team is accountable for releasing software, but also for keeping all branches in sync with the main branch (P 3, 6, 7), build scripts, setup of test environments, and so forth. Thus, in order to avoid overloading the release team, we defined the capacity of each release state according to data analysis of 2 months activity (more than 30 significant releases).

Control of Lead Time allows identifying the states that are time-consuming, and therefore, identify avenues for future improvements. For example, we identified that having one branch dedicated to the release constitutes a bottleneck. We are working on a multi-head branch for release (structured sub branches allowing a parallel work on more than one release.

Finally, although the post-release defects are not closely related to the release process itself, we have established a new practice for root cause analysis.

V. Conclusion

This paper reports on a release process improvement effort that has been realized in form of a transition from a set of incorporated activities to a structured process according to the Kanban principles and values. This transition helps to make release flows more visible for all stakeholders, showing releases progress and blockages. Thus, it becomes possible to identify bottlenecks and discontinuity of the release flow. Also, the transition outlined in this paper was worthwhile to the extent that we were able to identify factors that increase the Lead Time. Future work will focus on the release process optimization to make it more fluid. Also, we will establish a root cause analysis of post-release defects aiming to bring more insights on the release engineering practices.

Acknowledgment

Thanks to Payza teams for their graceful collaboration. This work has been partially supported by the NSERC program.

References

[1] F. Khomh,T. Dhaliwal,Y. Zou, *et al.*, Do faster releases improve software quality? An empirical case study of Mozilla Firefox. in *Mining Software Repositories*, 179-188, 2012.

[2] M. Marschall, Transforming a Six Month Release Cycle to Continuous Flow. in *Agile Conference*, 395-400, 2007.

[3] S. Shankland. Google ethos speeds up Chrome release cycle *http://dev.chromium.org/chromium-os*, 2010.

[4] M. Poppendieck and T. Poppendieck *Implementing Lean Software Development: From Concept to Cash*. Addison-Wesley Professional, 2006.

[5] H.K. Wright and E.P. Dewayne. Release Engineering Practices and Pitfalls *ICSE*, Zurich, Switzerland, 1281-1284, 2012.

[6] E. Shihab,C. Bird and T. Zimmermann, The effect of branching strategies on software quality. in *International symposium on Empirical software engineering and measurement*, (Lund, Sweden), 301-310, 2012.

[7] 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.

[8] N. Kerzazi and P.-N. Robillard. Multi-Perspective Software Process Modeling *International Conference on Software Engineering Research, Management and Applications*, Montréal, Canada, 85-92, 2010.

[9] D. Coghlan and T. Brannik *Doing Action Research*. Sage Publication, London, 2005.

[10] H. Kniberg and M. Skarin *Kanban and Scrum: making the most of both*. InfoQ, 2010.