Release Management for Parallel Development: A Case Study

De Sujoy, Rahut Amit Kumar, and Bhattacharya Uttam

***Abstract*—The objective of this whitepaper is to elaborate a case study where Cognizant consultants helped the IT division of a large private bank of Europe come up with a new Release Management model for migration of their existing legacy mainframe systems to the Java Application Platform (JAP). The existing Release Management model in the bank was analyzed and a new model was adopted which successfully addressed the drawbacks in the pre-existing one. Some basic metrics for gauging the performance of the two models were defined, a prototype spanning one release was carried out in both models, and data for the metrics was captured. The paper also outlines the quantitative analysis of both models on the basis of those metrics, and demonstrates the superiority of the new model over the pre-existing one.**

***Index Terms*—Branching and merging, software configuration management, release management.**

I. Introduction

In today's world, parallel development of multiple releases of a software product is the norm [1]. Software functionality is bundled into a set of planned releases. Each release takes time to be developed, tested and rolled out; thus necessitating development of multiple releases in parallel to make the best use of available resources and time.

Wonderful as parallel development sounds, it comes with its own share of problems. How can one ensure that the bug fixes and performance tweaks made in one release are properly applied in the other simultaneously going on releases [2]? How can such bug fixes and enhancements be safely disseminated to all the different customizations of the same release currently being supported to different clients? Problems like these make Configuration Management (CM) [3] in general and Release handling through Branching Strategies in particular a vital field in current day software development.

Release handling through branches in a CM tool like Subversion is a common software development approach today [4]. However, though all software communities universally agree upon the approach, there is an incessant debate on how to implement a branching and merging strategy. Different schools of thought persist, some in favor of stable trunks, some advising creation of feature branches, and others in favor of release branches [5].

The reason for the raging debate is that most of the branching and merging strategies come with their own share of unpleasant consequences [6]. Depending on factors like the nature of the software product, how frequently releases need to be done, existing knowhow in the project team, experience with different branching strategies and how well the underlying tool supports branching and merging, software developers decide on the strategy, or on a mix of different strategies.

Manuscript received August 14, 2012; revised November 9, 2012.

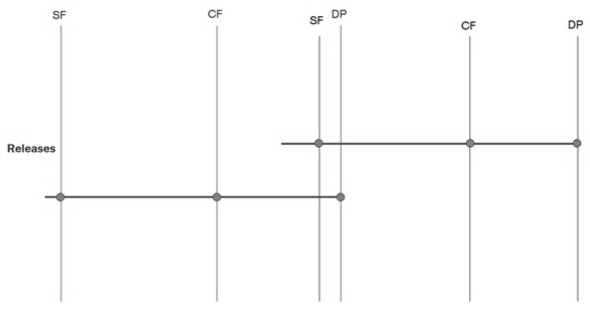
The authors are with the Process & Quality Consulting team with Cognizant Technology Solutions, India. (e-mail: Sujoy.de@ cognizant.com, amit.rahut@cognizant.com, uttam.bhattacharya@cognizant.com).

II. The Project Goal (The Problem)

Developers in the IT division of a large private bank in Europe faced the same problems of parallel development for their software products. A project was formed to migrate a suite of related Mainframe programs to the Java Application Platform (JAP). The project team chose Subversion as the Configuration Management tool for the entire project.

The IT division of the bank had a standard release model consisting of four releases in a calendar year, each spanning just over four months. This meant that consecutive releases overlapped slightly, as depicted in Fig. 1.

The project team decided that they would perform the mainframe to JAP migration in a set of releases which were conformant to the IT division's Standard Release Model (SRM).



Legend	
——	Branch for a specific release
⟶	Branch / Merge / Copy for a bug fix
SF	Scope Freeze (start of development)
CF	Coding Freeze (beginning of system testing)
DP	Deployment into Production environment
●	Milestone at which baseline is taken

Fig. 1. IT division's standard release model with slight overlaps between consecutive releases

A. *Description of the Standard Release Model (SRM)*

Traditionally, the IT division of the bank was responsible for maintaining their legacy mainframe applications. They introduced new functionality in the legacy applications through the four releases per calendar year. Projects carried out the entire software development in the release branches.

When it was time to open a new release, they copied the files over from the old branch directly to the new one, as shown in Fig. 2.

Product documentation (Requirements documents, Architecture models and Design documents) for the new release was supposed to be updated till the Scope Freeze (SF) milestone, then code updates would commence till the Coding Freeze (CF) milestone. After this milestone, a separate testing team would commence with System Testing, and developers would fix bugs till it was finally deployed in the Deployment (DP) milestone. Baselines were to be taken at the SF, CF and DP milestones.

This simplistic approach worked because introduction of new functionality in each release was a pretty simple affair involving updates to a set of fairly independent code files. Thus, projects could easily contain all development and testing within the four month time frame and very few bugs were reported. This in turn meant very few bug fixes needed to be ported to the new branch when it was time to open one for the next release, thus making it a useful and simple release management strategy for the IT organization. (See Fig. 2.)

B. *Analysis of the Standard Release Model*

Analysis revealed that this standard model was unsuitable for the JAP migration project due to the following drawbacks:

1) The JAP product was being built entirely from scratch, which meant that the team had to create the entire requirements and solution design documentation for it. Since the product documentation for a release had to be finalized by the Scope Freeze milestone, the team had to open the release branches early to allow sufficient time to the Requirements Engineers and Solution Architects to update the documentation.
2) The other disadvantage of the novelty of JAP product was that the team expected significant bugs and change requests in each of the releases. There was also the necessity of opening the releases early. Since opening a new release meant that the entire contents of the earlier branch (including the code) were copied over to a new one, the new releases started off with a very unstable code base.

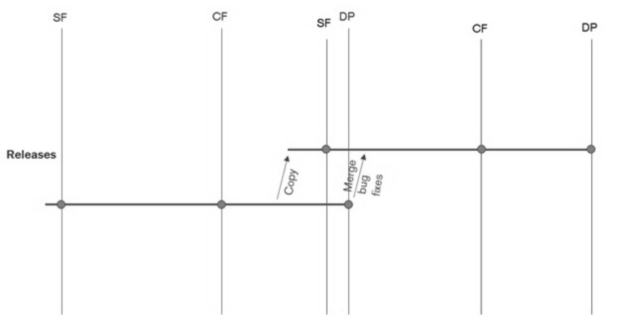


Fig. 2. Release opening strategy in standard release model (SRM)

3) There was huge merging effort from the current release branch to the next one after deployment of the current release (as there were a lot of bug fixes that had to be merged after deployment).

Since there was a huge volume of Requirement Engineering & Solution Engineering deliverables to be created, idle time for these people was completely unacceptable, especially since the documentation was vital to the testers for proper testing of the new system.

When the Scope Freeze milestone had been reached for the current release, the Requirement Engineers and Solution Architects had to wait till the next release was opened before they could start with product documentation for the next release.

From the above analysis, the team decided that significant adaptations needed to be made to the Standard Release Model to accommodate the JAP project.

III. Achieving the Project Goal (the Solution)

The project team held many brainstorming sessions with Cognizant Consultants and other CM experts and various suggestions were proposed. Finally, the team adopted a significantly changed model named “Branch for Testing Model” (BTM).

A. *Description of Branch for Testing Model (BTM)*

The team decided to introduce a mainline branch (Trunk in Subversion) where the development for all the releases would take place. This would enable continuity of work for the Requirement Engineers and Solution Architects as they were no longer forced to wait till the new release branch was opened. Development would proceed on this mainline branch, and at the Coding Freeze milestone it would be branched off to create a release branch. The Scope Freeze and Coding Freeze baselines would be taken on the mainline.

The Testing team would then pick up the code base from this release branch and test it. As bugs are found and fixed, the developers would do frequent merges from the branch to the trunk. This would ensure that the effort required for individual merges was significantly low. The release would finally be deployed from the release branch itself, and the deployment baseline would be taken on the release branch. The Branch for Testing Model is depicted in Fig. 3.

The team migrated the legacy mainframe programs to the JAP platform using this model and the entire functionality was successfully migrated over seven releases. The project went on successfully with the migration while staying well within the limits of planned effort and within acceptable limits of defect density.

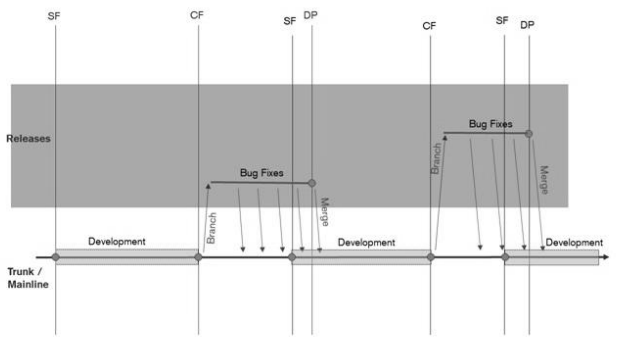


Fig. 3. Branch for Testing Model (BTM)

B. Analysis of the Branch for Testing Model

This model successfully addressed the various drawbacks in the Standard Release Model as follows:

1) As development and product documentation could go on uninterrupted in the mainline (trunk), Requirements Engineers and Solution Architects could move on to the documentation for the next release immediately after freezing them for the current release.
2) The team could open release branches very late into the release. At this time, the code development of the complete functionality of the current release would have been completed and only testing and bug fixes would remain before it was deployed. Thus the code-base in the mainline would be in a very stable state when branched.
3) Since only fixes for the bugs found in System Testing were done in the release branches and no development took place there, the number of files that required merging was much lesser, resulting in much lesser merging effort in total.
4) This was also in line with the golden CM principle of branch late, merge early.

However, this model was also not completely foolproof. Some of its disadvantages were:

1) The approach is not intuitively easy to understand for someone uninitiated to CM principles. Baselines for a release have to be taken partly on the mainline (trunk) and partly on the release branch, which can become confusing.
2) A conflicting requirement over two simultaneous releases is a bit of a challenge and requires careful handling. E.g., functionality in Release 3.0.0 that conflicts with functionality for Release 2.0.0 will need some special handling from the project team, as development for release 3.0.0 being done on the trunk will normally be overridden by the regular merges to it from the Release 2.0.0 branch till the time Release 2.0.0 is deployed into production.

IV. Quantitative Analysis of the Two Models

To get a quantitative idea of how both models fared, the team carried out a small JAP development prototype spanning one release simultaneously in both models and collected data for both. The team comprised of two persons from each of the disciplines of Requirements Engineering, Solution Architecture, Code Development and Testing.

They compared the performance of both models over the following basic metrics: **Branch Open Duration** (how long the release branch was open till deployment); **Total Merging Effort** (the total effort required for all the merges); and **RE, SE Idle Time** (the time Requirements Engineers and Solution Architects were forced to remain idle).

Branch Open Duration: This was the simplest of all the metrics. It was four months in the *Standard Release Model (SRM)* and two months in the *Branch for Testing Model (BTM)*. Going by the golden rule of branch late, merge early, BTM came out on top in this metric.

Total Merging Effort: The total person hours necessary for merges of all the code fixes in both models were recorded. The resulting table is displayed in Table I.

As expected, month four saw a huge spike in SRM as most of the bug fixes were naturally done in month four (last month before deployment). In BTM, the merging effort predictably diminished in the last month as the code base was fairly stable when the branch was created and most of the fixes were already completed by month three. Comparison of total effort demonstrates that BTM is a clear winner compared to SRM.

RE, SE Idle Time: Last but not the least, the idle time in person days for the team was also recorded. The results are tabulated in Table II.

By eliminating the Requirements Engineers' and Solution Engineers' waiting time for a new branch creation, BTM ended up with huge effort (and hence cost) savings over SRM. Here also, BTM came out superior to SRM.

The experiment with the JAP prototype showed a total effort savings of 136 + (22 – 3) / 8 ≈ 138 person days in the BTM model when compared with SRM. Although the data was captured using a prototype, the results prove conclusively that Branch for Testing Model is more efficient than the Standard Release Model.

TABLE I: Total Merging Effort Comparison

	SRM	BTM
Month 1	0	0
Month 2	0	0
Month 3	6	2
Month 4	16	1
Total	**22**	**3**

TABLE II: RE, SE Idle Time Comparison

	SRM	BTM
Month 1	0	0
Month 2	80	0
Month 3	56	0
Month 4	0	0
Total	**136**	**0**

V. Conclusion

Based on the results of the prototype, the IT division of the bank adopted the Branch for Testing Model as the standard release model for all JAP projects.

The brainstorming sessions that helped formulate the Branch for Testing Model had another very positive outcome as well. The central process group of the organization, the quality group, the CM expert team and all members of the project team had very close collaboration all through the time the model was being developed.

We witness only too often that the quality assurance group of an organization and the project teams who actually execute projects operate conflictingly; the quality assurance group tries to align organization processes with software quality standards like CMMI, which project teams find inconvenient to follow. However, in this case, both teams collaborated with each other with the help of Cognizant's consultants to design a process that greatly improved actual execution on the one hand, and was also compliant to the quality standards of the organization on the other.

REFERENCES

[1] M. Sakyo. (August 2004). The challenges posed by parallel development on release management. *CM Crossroads™* [Online]. Available: http://www.cmcrossroads.com/cm-journal-articles/6740-the-challenges-posed-by-parallel-development-on-release-management

[2] B. C. Sussman, B. W. Fitzpatrick, and C. M. Pilato, *Version Control with Subversion*, For Subversion 1.7: (Compiled from r4304), pp. 96.

[3] *CMMI for Development*, Version 1.2, CMMI-DEV, V1.2, Carnegie Mellon, Software Engineering Institute, 2006, pp. 114-130.

[4] L. Wingerd, *Practical Perforce*, 1st ed. O'Reilly Media, 2005, vol. 7, pp. 178-180.

[5] J. D. Meier, J. Taylor, A. Mackman, P. Bansode, and K. Jones, *Team Development with Visual Studio Team Foundation Server patterns & practices*, Microsoft Corporation, vol. 5.

[6] J. Buffenbarger and K. Gruell, "A branching/merging strategy for parallel software development," in *Proc. of 9th International Symposium, SCM-9*, Toulouse, France, September 1999, pp. 1-2.

Sujoy De is a consultant of Cognizant Technology Solutions having 8 years of experience in various fields of Software Quality and Tool Implementation. Mr. De was born in Bankura, India on 28th of July, 1981 and received his engineering degree (Bachelor in Computer Science & Engineering) in the year 2004 from Burdwan University, India, and Diploma in Business Administration in the year 2009 from Pune University, India. He has wide experience in various fields of software quality like Process definition & implementation, process improvement and maintaining the Quality Management System. He has also experience in CMMI Level 3 implementation, ISO 9001 framework and metrics definition. He has worked as a Configuration Manager for the IT division of one of the largest private banks in Europe. He has experience in organization wide implementation of process management applications for application development and maintenance projects and has an in-depth understanding of SDLC concepts, continual improvements and high maturity process areas. In his previous organization, he was instrumental in the organization's achieving the ISO 9001:2000 recertification and its preparation for ISO 140001 certification.

Amit Kumar Rahut is a consultant of Cognizant Technology Solutions having 10 Years of experience in the field of process definition, implementation and process improvement with CMMI, Six Sigma, and ISO 9001 model. Mr. Rahut was born in Kolkata, India on 31st October, 1977 and became an engineering graduate (Bachelor in Technology) in the year 2002 from Calcutta University, India. He has wide experience in the field of consulting with direct interfacing for many clients for process definition, implementation, process improvement and maintaining their Quality Management System. He has implemented CMMI, Six Sigma, ISO 9001 framework, metrics definition for a client organization. He has worked as a Configuration Manager in the IT division of the largest private bank in Europe. He has experience in organization wide implementation of process management applications for application development and maintenance projects and has an in-depth understanding of SDLC concepts, continual improvements and high maturity process areas. He has worked as a Quality Lead for process benchmarking and implementation for a big manufacturing organization and had implemented *Theory of Constraint* project resulting in increased profitability. Mr. Rahut is certified Project Management Professional (PMP®) from PMI, USA and has cleared the ITIL® version 3 Foundation Examination from EXIN/APMG/OGC. He is also a certified Software Quality Analyst from Cognizant Certified Professional examination. Mr. Rahut was also associated with American Quality Society (ASQ) for two years and was a member of Project Management Institute (PMI), USA. He is also an eminent writer in the Cognizant Process Quality Consulting newsletter and is part of the editorial board.

Uttam Bhattacharya is a Senior Consulting Manager of Cognizant Technology Solutions having 19 Years of experience in the field of strategic assessment, process definition, implementation and process improvement in CMMI, Six Sigma, and ISO 9001. Mr. Bhattacharya was born in Kolkata, India on 2nd August, 1970 and obtained his engineering graduation (Bachelor in Technology) in the year 1993 from Calcutta University, India. Mr. Bhattacharya has also completed his MBA (part time) from Calcutta University, India in 2001. He had played the role of Quality manager for Cognizant and was responsible for ensuring quality of deliverables of the projects. He has implemented CMMI, Six Sigma, ISO 9001 framework, metrics definition for various business units in Cognizant. He has also led the CMMI assessment for Cognizant. He has wide experience in the field of consulting with direct interfacing with many clients for Strategic assessment, Process definition, implementation, improvement and maintaining their Quality Management System for the client organizations spread across geographies. He has also led a number of Six Sigma projects. He has wide experience in organization wide implementation of various processes in different types of projects and has an in-depth understanding of SDLC concepts, continual improvements and high maturity process areas. Mr. Bhattacharya is a certified Project Management Professional (PMP®) from PMI, USA and has cleared the ITIL® version 3 Foundation Examination from Quint. He is also a certified Six Sigma Black Belt Certification form BMG, and is a certified internal auditor of ISO 9000. Mr. Bhattacharya is a certified Scrum master from Scrum Alliance and is a member of Project Management Institute (PMI), USA. He is also an eminent writer in the Cognizant Process Quality Consulting newsletter and is part of the editorial board.