Holistic Approach to Software Build and Release Process with Pre-emptive Measures for Sustainable Quality Control

Muhammad Aiman Mazlan, Mani Hagh Sefat, Nor Ezam Selan, Dickson Lukose
MIMOS Berhad
Technology Park Malaysia
Kuala Lumpur, Malaysia
{aiman.mazlan, mani.haghsefat, nor.ezam, dickson.lukose} @mimos.my

***Abstract*— The need to reduce unnecessary defects and frustration resulting from the build and release process has motivated the authors to develop a holistic approach to improve the software build and release process. This also include the control process to ensure it's sustainability. Throughout software development life cycle, changes to the software can occur at any stages as a result of continuous software improvement due to enhancements or bugs fixing. The quality of the software product is not only contributed by a rigorous testing, software development practices and adherence to the established process, but also by the build and release management process, which is often overlooked. This paper discusses the experiences of the authors to minimize introduction of new "bugs" during the build and release process. The practice covers automation of tasks from multiple software programming tools into one single build script, and implementation of preventive and proactive measures to avoid violation of the established procedures and processes. This practice demonstrated reduction of over 90% in defect density.**

Keywords—component; Configuration Management, Software Build and Release Management, Automated Software Build, Software Configuration Management.

I. INTRODUCTION

Software development life cycle consists of the requirement gathering, design, coding and testing phases [1 - 4]. Depending on the capability maturity level of an organization, quality gates are put in place to ensure the released software products are thoroughly verified and validated to minimize the potential defects reaching the end users.

It is important to ensure that every consideration is taken care of to minimize the unnecessary defects and frustration resulting from the build and release process. This is more crucial in software development that involves projects running in parallel with developers from multiple disciplines.

The authors are part of the Research and Development lab that consists of a group of researchers and developers from diverse background and practices in software development. An Applied Research team produces software components while a development team incorporates the components into a platform and develops applications based on this platform.

When the team was initially formed in 2007, there was no established build and release process. Most of activities were done manually. This was also the time when the lab was in the process of improving and optimizing the Capability Maturity Model Integration (CMMI) process areas towards achieving a higher maturity practice [4].

With the average release cycle of 6 months, the challenges for the software developers were to ensure that the products they develop bug free and mistakes are not repeated. When a bug is found during the software product testing, there is a need to distinguish whether it is due to the source code itself or due to the software build and release management process. If there is an established and reliable build and release management process, one probable cause can be eliminated, thus focusing the investigative resource to reviewing the source code.

This paper will look into the authors' experience in implementing the holistic approach to resolve the product defects as the result of building and release process through an automation and control of the software building processes. It also discusses the implementation of pre-emptive process that provides event triggers to detect markers for potentially deviant practitioners to ensure the process improvements are sustained within the range of system control.

This paper is organized per the following: Section II will discuss the factors contributing to defects from software build and release process, Section III will discuss our approach to automate the software build and release process, and finally, Section IV will conclude this paper with a discussion on result obtained and lesson learnt.

II. RELATED WORK

Chan and Hung [8] have reviewed several Software Configuration Management (SCM) tools and identified some of the problem areas associated with the standard SCM activities. For example, if a constituent item of a software object was changed, all affected items should have been changed. Thus, a new bill of material is required to facilitate developers' awareness of the inter-dependencies between the configurable items. Their paper focus on SCM activities identified problem areas within SCM tools and proposed several suggestions to make SCM tools more effective.

NHN Test Automation Framework (NTAF) is another implementation which make used of automated tools such as Ant, Maven, Continuum and CruiseControl in their projects [13]. Their case studies showed that by incorporating several automated build tools, they have managed to automate

repetitive and error-prone processes for testing in a continuous integration environment.

There are several suggestions on Software Release Management best practices as described in the IT Infrastructure Library (ITIL) [15] and CMMI for Development [4]. The suggestions defined a set of good practices for adopting the Software Change and Configuration Management successfully. However, these suggestions serve as a general guideline. Process tailoring is still needed to cater for specific criteria of the software development environment. Therefore, the process and tool need to support each other to have reliable software build and release process.

In our implementation of build and release process, we are not only focusing on creating new process and tool customization but we extend it further to have a new set of control process in place. This is a holistic approach to provide an end-to-end build and release process which is reliable and sustainable with minimum intervention.

III. Factor Contributing to Defects From Build and Release Process

Through several brainstorming sessions with the project team members and usage of six sigma tools such as Project Mapping, Cause and Effect Analysis and 5-Why Root Cause Analysis [5], the major root causes contributing to the software product defects have been identified as listed in TABLE I.

TABLE I. 5-Why Root Cause Analysis

Validated Root Cause	Last Why
Software build is done at different developers' local machines rather than centralized server causing inconsistency and integrity of software executables	There is no automated way for developer to build a package
Too many variations to build a package	There is no automated script to help to build a package
Configuration management process not properly defined	No dedicated person to look into configuration management
All developers have the authority to merge artifacts into mainline	No access control implemented
Developers replace whole directory instead of check in individual elements	Inappropriate use of the configuration management tool

From the root cause analysis, we were able to conclude that when the software development activities are complicated and with many dependencies, automated build tool and mature process of software build and release are necessary. Having a controlled build and release management will definitely help to minimize possible defects contributed from build and release process.

IV. Holistic Approaches for Build and Release

After Software build and release management is a challenging process when it comes to compiling and packaging software package and deliverables. The main idea for release management is to ensure the process is simple and consistent every time we build and release a software product. In line with the organization process mandate, detail implementation and process tailoring are done to suit the lab's demographics and software development criteria. We looked at a broader approach for tackling this problem with reliable and consistent approaches as the following:

1. Appoint a dedicated focal person in charge of the software configuration process. The identified person needs to have the skill set necessary for software quality control and configuration management.
2. Provide training on the software quality process and version control process to the new hires, and refresher trainings for the existing team members. Refresher trainings are required as build process is continuously being improved.
3. Create a list of configuration management guidelines, software build procedures and disseminate the information through trainings and group discussions.
4. Standardize the practice of using the software tools.
5. Automate the repetitive processes.
6. Provide event triggers as a proactive approach to detect marker for potentially deviant practitioners.

The following subsections describe the details of six approaches.

A. Software Configuration Manager

First, the Configuration Manager has the primary responsibility for the configuration management activities and process control for the team. The configuration management activities include configuring and managing configuration management tools, develop and execute configuration management strategies (i.e., version controlled traceability, branching strategy, build and release management). The process control activities include ensuring that the software product artifacts produced by the team adhere to the baselined software development process. The Configuration Manager acts as a bridge between the organization's Software Quality Engineering group and the development team.

B. Training

Every new hire will have to undergo an Induction Training Program to familiarize themselves with the organization process and procedures, as well as on the research and development processes. The training covers the aspect of:

- Software Production Process
- Defect Tracking Process and Tools
- Version Control Process and Tools
- CMMI and Quality Assurance Overview
- Review/Inspection and Metrics

The Software Configuration Management (SCM) training provided in the Induction Training Program focuses on the general practices of the organization. Though, the team would still need to be introduced to a customized practice which is defined in Software Configuration Management Plan (SCMP) of the development team. As such, subsequent trainings which are more focus to the lab practices and guidelines are required. These trainings also serve as refresher trainings for existing team members. The refresher trainings are usually held at the beginning of a new project such that the team members can start with the correct practice from the initial phase of the project.

C. Creation of Process and Guidelines

Configuration Management is a support process defined at Maturity Level 2 in CMMI framework [3, 4, 7]. This support process will establish and maintain the integrity of work products using configuration identification, configuration control, configuration status accounting and configuration audits [3, 4, 7, 8]. Based on CMMI best practices, a SCMP document will be introduced. The primary goal of a SCMP document is to establish and maintain the integrity of the products of the software project throughout the project development life cycle [4, 9, 10]. The configuration management activities include:

- Version Control– branching strategy and label
- Archiving
- Change Management & Problem Tracking
- Configuration Status Accounting
- Release Management

The SCMP document is a live document and is used as a standard operating procedure throughout project development life cycle. The configuration management activities are planned upfront and made known to the project's team members and stakeholders.

D. Standardization of Process and Tools Usage

In the lab, the team develops software components, incorporates the components into a platform and develops applications based off the platform. Due to this approach to software development, we adopted the following strategies for standardization of process and tools usage.

1) Directory and Source Code Restructuring

Developing components for the software platform that will be used by numerous developers in multiple projects require uniformity in code-structure and package-structure. This is particularly important so that these components can be easily integrated both at the platform level as well as the application level.

As illustrated in Figure 1, Authenticator is one of software component that we develop. We have structured the directory layout in the repository server and as well in the source code. The naming convention starts with the reverse domain name, which is "*my.mimos*", followed by product name, which is "*my.mimos.stp*". Then, the next name would be the functionality area or the component name itself. In Figure 1, it was name as "*my.mimos.stp.auth*". Hence, well-defined directory structure of Java project is important to ensure uniformity and standardization for every Java component project. This will simplify the build automation process at a later stage.

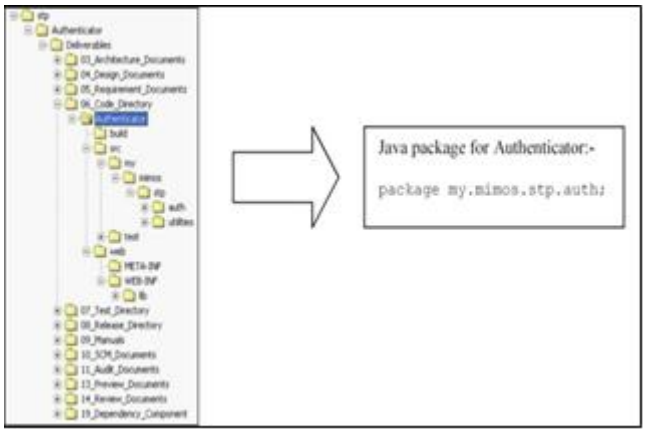

Fig. 1. Code Directory Structure: Authenticator Component.

2) Standardization of the Tools with Unified Solution

In software development life cycle, there are many tools involved during and after development phase. Some of these tools are recommended by team members or enforced at organization level. For example, ProGuard [11, 12] is used to obfuscate the source codes prior to being packaged and JUnit [11] is used during unit testing.

Furthermore, certain standalone Java applications need to be executed via web browser, thus, the usage of Java Web Start (JWS) is needed. JWS allows user to launch Java application from web page, which make use of Java Network Launch Protocol (JNLP) file [11, 12]. For this type of application, the build package needs to be digitally signed by the authorized person. Thus, it requires manual step, where the developer needs to apply his/her digital signature prior to handover the build package to the next step.

Based on the tools or tasks mentioned above, regardless whether they are used in optional or mandatory manner, we need to identify the best way to fit all requirements into one unified solution. In TABLE II. we have made the comparison between Maven [11], Ant [5] and Make [8] build tool based on our requirement. Based on the comparison analysis made, Ant met most of our requirements. In addition, Ant has been widely used in Java projects for quite some time, thus it is mature and stable with a rich library of tasks.

TABLE II. BUILD TOOL ANALYSIS

Our Build's Requirement	Maven	Ant	Make
Support on existing configuration management tool, such as Rational Clear Case and Subversion	Yes	Yes	No
Java code compilation	Yes	Yes	Yes
Javadoc generation	Yes	Yes	No
Support on Junit testing	Yes	Yes	No
Support on code obfuscation	No	Yes	No
Digital signed on build package	Yes	Yes	No
Component version traceability	No	Yes	No
Library dependency	Yes	Yes, with Ivy	No

E. Pre-emptive Control

We have adopted pre-emptive control mechanism in order to sustain the gains from the implemented approach. A trigger is a user-defined procedure that is automatically executed in response to certain events. Most of the configuration management tool has this feature, which need to be configured accordingly. There are two types of triggers: pre-operational trigger; and post-operational trigger. Pre-operational trigger is typically used to enforce process by disallowing the operation from occurring if certain conditions are not met. For example, pre-operational trigger can be used to prevent user from deleting versioned files. Whereas post-operational trigger is executed after a specific operation is completed. Usually, post-operational trigger can be used for clean up session, notification to user or for continuous integration. On top of that, we also need to create user scripts to complement configuration management tool.

1) Managing Process Violation

Process Violation Management prevents the user from violating the configuration management process. For example, a user is only allowed to upload files into repository but cannot remove a directory structure or delete versioned control files permanently. There are auto triggers to control these violations from happening and every violation attempt is recorded into log file. Though the system is already configured to fail the attempt but the triggers will give indication of attempt to violate. Every attempt can be categorized as:

- Type 1 - Attempt to destroy/delete a versioned files permanently
- Type 2 - Attempt to modify name of versioned files.

A weekly review on the recorded data will take place and initiate intervention if accumulated violation for a particular user is above the upper control limit (UCL). The process violation trigger ensures that team members are aligned to the process and guideline as defined in SCMP.

In Figure 2, user 1 has reached his UCL, thus discussion with the user on reasons for violation is needed. Potential outcome from the discussion could be to educate, re-train or further update to the process guidelines.

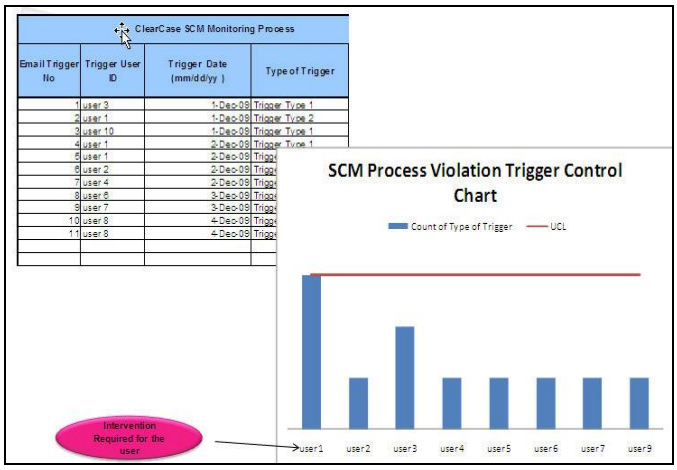

Fig. 2. Process Violation Trigger Control.

2) Implementation of Process Audit

Process Audit trigger is used to notify Configuration Manager if a user fails to perform a certain action after certain duration of time. In order to ensure each team member adheres to the processes and guidelines defined in SCMP, the process control is implemented via automated auditing. First of all, creation of user scripts to run automated auditing for the following type of alert on a daily basis:

- Type 1 - Directory not checked in > 3 days
- Type 2 - File not checked in > 7 days

The weekly review of the alert will determine the status of auditing in configuration management tool. If cumulative alert trigger for particular user is equal or above the UCL, then intervention will be initiated. Interventions means to find out from the user on his/her issues. Action need to be taken once UCL is reached. Otherwise potential build and release related issues will arise. The detective mechanism is important in ensuring that configuration management process is being adhered correctly. In Figure 3, two users have reached UCL, thus intervention is required for them. The potential outcome from the discussion could again be to educate, re-train or further update to the process guidelines.

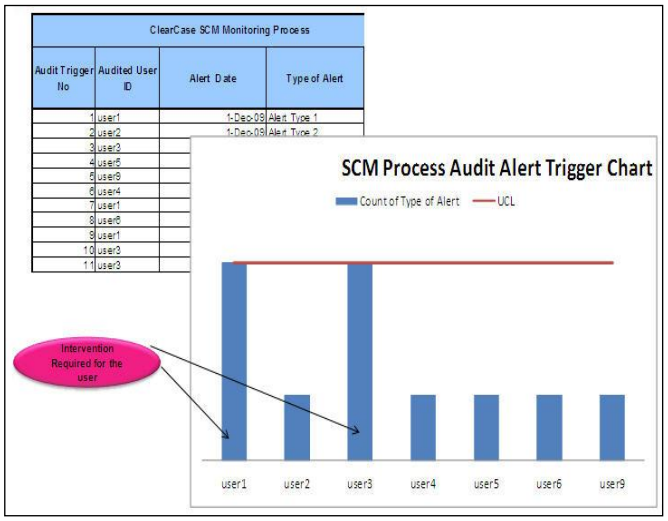

Fig. 3. Process AuditAlert Trigger.

V. Implementation of Automated Software Build and Release

The build and release activities are highly repetitive. We have created an Ant build scripts to automatically compile and generate a release package. This can be achieved when there is a standard Java directory structure in place for each Java component. The Ant script need to be tailored for each component as every component may have different characteristics, depending on the type of software product (i.e., backend components, client components, platform tools or utility tools). We have created a template properties file which is used by Ant build script during execution to make the configuration dynamic and user friendly for developers to customize.

There are two important files in Ant build script. They are *build.xml* and *build.properties* as illustrated in Figure 4. Bear in mind, we are developing multiple Java components with different characteristics. Hence, it is important to ensure the Ant build script is generic and reusable for other projects.

In each project, developers will be tasked to use and customize the Ant build script based on the template given to build their portion of the software components prior to send them for integration. The developers are also responsible to test the customized Ant script at both, his/her local environment and at server environment. On top of that, the developers are required to use the same version of Java Development Kit, Java Integrated Development Environment (IDE) version, Java libraries, databases and utilities tools downloaded from common repository. Only tools and libraries taken from the common repository are allowed to be installed on developers' machine. This is to ensure consistency in terms of version of the tools and libraries used.

Ant build tool is based on Java classes and uses XML for configuration file. Ant provides a large number of predefined tasks that can be used. However, a user can create its own task to suit the requirement. The tasks can be grouped into several targets in the XML configuration file. For example, in Figure 4, Ant build script can handle multiple tasks from any Java programming tools. These tasks can be group into targets to define dependencies. This is a best way to automate certain task which involves repetitive procedures, for example code compilation, code obfuscation and etc.

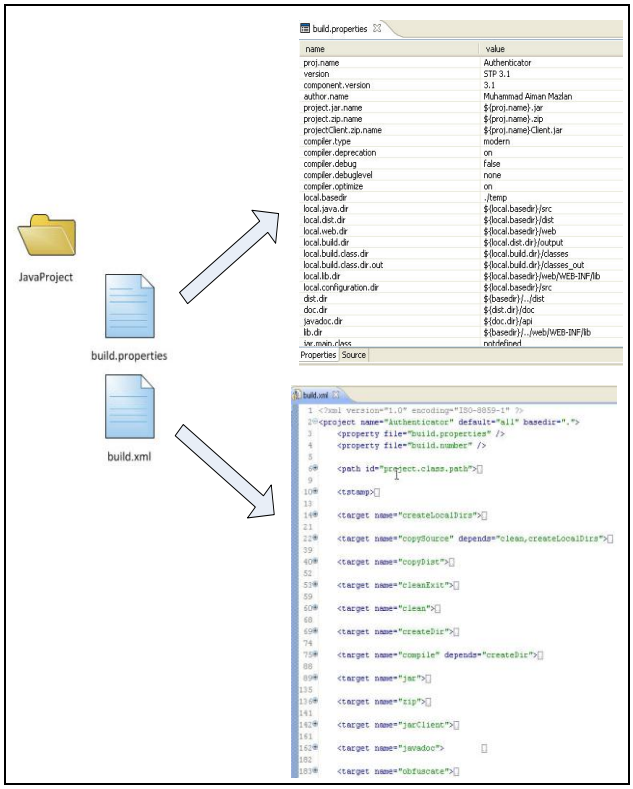

Fig. 4. Excerpt Code of Ant Script and File Structure in Java Project.

A. Improvement Results

We have examined the data captured in five of our software product releases, which are STP 1.1.1, STP 1.2.0, STP 2.0, KRSTE 3.1 and ILMS 1.0. The data that was monitored was the Software Configuration Management (SCM) issue (count of PR) over total number of PR raised, and over the kilo line of code (KLOC) of the release software product.

Figure 5 shows that in STP 1.1.1, 7 out of 15 PRs (50%) were due to the SCM issues. This is almost half from the total number of PRs raised. Based on STP 1.1.1 data, we started to look into the improvement that can be made to reduce this number of defects. Several quick gain approaches were then started prior to the release of STP 1.2.0. The result was an improvement where the number of PRs raised attributed to the SCM issues was 5 out of 20, which is 25%.

We then took the data from the STP 1.1.1 and STP 1.2.0 as our baseline data for further improvements in our STP 2.0 release. In addition, the same improvement was also made into projects KRSTE 3.1 and ILMS 1.0.

Before

Software Product Release	KLOC	Delta	# of PRs	PRs due to SCM Issues	y(SCM Issues)
STP 1.1.1	29.859	3.657	15	7	1.91
STP 1.2.0	36.54	6.681	20	5	0.75
Total		10.338	35	12	2.66
Average		5.169			1.33

After

Software Product Release	KLOC	Delta	# of PRs	PRs due to SCM Issues	y(SCM Issues)
STP 2.0	72.933	36.393	32	2	0.05
KRSTE 3.1	62.778	10.155	42	1	0.10
ILMS 1.0	88.485	25.707	22	2	0.08

Fig. 5. Before and After Improvement.

Further, Figure 6 shows that the PR due to SCM issue/KLOC has significantly reduced to 2 out of 32 defects found. The average defect density for STP 1.1.1 and STP 1.2.0 was 1.33, and the defect density for STP 2.0 was 0.05. The improvement was almost 96% in STP 2.0.0 from baseline data (STP 1.1.1 and STP 1.2.0). Similar trend were observed in KRSTE 3.1 and ILMS 1.0, where improvement factor was 92% and 94%, respectively. We have achieved this result after we implemented automated build and release process.

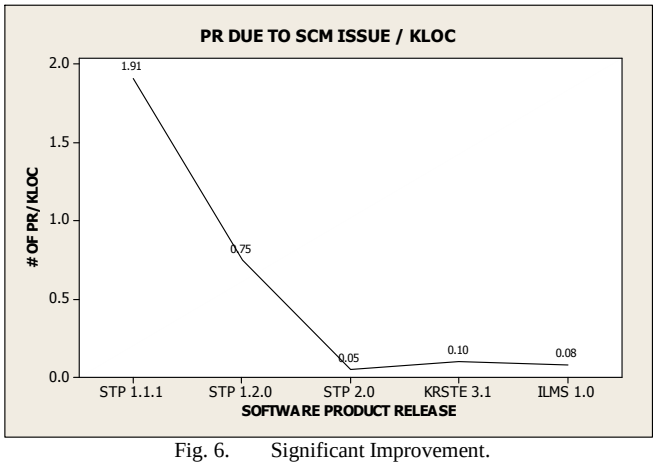

Fig. 6. Significant Improvement.

B. *Problem encontered and lesson learned summary*

1) *Automation*

Several Java application tools can be combined together in Ant build script into several tasks. The tasks later can be arranged to manage the software dependency. The automation helps to reduce any repetitive task, thus team member can concentrate on more important issues.

2) *Source code and build script maintenance*

Similar to source code, Ant build script also requires maintenance to cope with newly implemented features, changes to imported application programming interface (API) library and source code restructuring. Similar process used in the maintenance of the source code need to be applied for the Ant script maintenance.

3) *Process Changes*

Documentation on process and guidelines is important especially in the big organization. In our implementation, we have documented the SCMP to serve as Standard Operating Procedure (SOP) and to be followed by every team member. The SCMP was prepared upfront at the start of the project, and shared with the team members as part of the project kick off activities. This is to ensure that everyone is on the same page and is aware of the guidelines and practices to follow in the software development project.

4) *Dedicated Configuration Manager*

In our implementation, every software product should be released by a Configuration Manager. Prior to releasing software product, the Configuration Manager also need to ensure the software product are correct and tested before it can be released to the independent test team or end users.

5) *Integration with external tools*

In our implementation, automation has reduced human intervention in executing task. This has helped to minimized the unnecessary errors contribute by human. At the same time, it has improved productivity. Integration with external tools from other team is another aspect which can be looked further. For example, Sonar application [11, 12] is used to measure the code quality of software product. By having this integration, code quality inspection can be done during the build and release management process. It helps to saves time as issues related to code quality can be captured upfront, prior to releasing the software product.

VI. CONCLUSION

This paper recorded our experiences in implementing a holistic approach to establish the build and release process for Knowledge Technology Research and development team at MIMOS. This has resulted in a significant reduction of software bugs that are attributed to the build and release process. It has tremendously improved the team effectiveness in zooming to parts of the source code whenever bugs are found rather than doubting whether the bugs were merely coming from the build and release process. Although, process transitions from ad hoc build and release management to a controlled build and release management will not happen overnight, there is a need to ensure that the software process and software tools are synergized for the holistic approach to be successful.

ACKNOWLEDGMENT

This work is supported by Six Sigma Project Improvement initiative at MIMOS Berhad.

REFERENCES

[1] Crnkovic, I., "A Change Process Model in an SCM Tool". Proceedings of the 24th Euromicro Conference. volume 2, pp. 794 – 799, Vosteras, Sweden, 1998.

[2] Mazlan, M. A., "Stress Test on J2ME Compatible Mobile Device", Proceedings of the Innovations in Information Technology (IIT 2006), Dubai, UAE, November 2006.

[3] Seawlho, P. and Suwannasart, T., "A SCM Workflow Model for CMM Organizations". Proceedings of the 10th Asia-Pacific Software Engineering Conference, pp. 253 – 260, Thailand, December 2003.

[4] CMMI Product Team, CMMI for Development, Version 1.3 (CMU/SEI-2010-TR-033). Carnegie Mellon University, Software Engineering Institute. November 2010.

[5] Lee, Kevin A., IBM Rational ClearCase, Ant, and CruiseControl: The Java Developer's Guide to Accelerating and Automating the Build Process.IBM Press. 2006.

[6] Jansma, P.A., "When Management Gets Serious About Managing Software". Proceedings of the IEEE Aerospace Conference. pp. 4366 – 4382, USA, Mac 2005.

[7] Kim, S., Whitehead, E.J. and Zhang, Y., "Classifying Software Changes: Clean or Buggy?", IEEE Transactions on Software Engineering, pp. 181 – 196, USA, April 2008.

[8] Chan, A.K.F. and Hung, S., "Software Configuration Management Tools", Proceedings of the Software Technology and Engineering Practice, 8th IEEE International Workshop on Incorporating Computer Aided Software Engineering, pp. 238 – 250, London, July 1997.

[9] Raygan, R.E., "Software Configuration Management Applied to Service Oriented Architecture", Proceedings of the 15th International Conference on Software, Telecommunication and Computer Network (SoftCOM 2007), Croatia, Sept 2007.

[10] Kilpi, T., "Improving Software Product Management Process: Implementation of a Product Support System", Proceedings of the 31st Annual Hawaii International Conference on System Sciences, 1998.

[11] Cay S. Horstmann, Big Java, 3rd Edition. John Wiley & Sons, Inc. 2008.

[12] The Java Tutorials. last updated on 17 March 2011. URL: http://download.oracle.com/javase/tutorial/ (last visited April 2012).

[13] Kim, E. H., Na, J. C. and Ryoo, S. M., "Test Automation Framework for Implementing Continuous Integration". Proceedings of the Sixth International Conference on Information Technology: New Generations (ITNG'09), pp. 784 – 789, Nevada, USA, April 2009.

[14] Louis Glassy, "Using Version Control to Observe Student Software Development Processes". Journal of Computing Sciences in Colleges, Volume 21 Issue 3, pp. 99 – 106, February 2006.

[15] The Buildmeister. Software Release Management Best Practices. Last updated on March 2009. URL: http://buildmeister.com/articles/software_release_management_best_practices (last visited August 2012)