5 STEPS OF CONFIGURATION MANAGEMENT FUNCTIONALITIES

By Eric Mariacher

1 ABSTRACT

Now more than ever, companies want to deliver faster, better and cheaper products. Last years, Some Industry best practices have started to emerge to help companies to achieve these goals by improving processes in their organization. Among others, CMMI is a recognized industry best practice. An important CMMI process area specific to software development projects is Configuration Management. Configuration Management has 2 main aspects, each having their own dedicated tools:

1. Source files and release configuration management.
2. Change request management affecting file and release management.

This technical paper describes different Configuration Management functionality steps and their ability to address series of Configuration Management related questions. It will be shown that it is not tools functionalities themselves that make the difference between Configuration Management functionality steps but the level of integration between those 2 kinds of tools.

 © 2006 Eric Mariacher. All rights reserved.

2 CONFIGURATION MANAGEMENT OVERVIEW

The purpose of Configuration Management is to establish and maintain the integrity of work products using configuration identification, configuration control, configuration status accounting, and configuration audits. [PA159]

To maintain integrity of work products requires having visibility and traceability inside Configuration Management processes.

Configuration Management for software projects deals with files, releases and change requests. Change requests can be feature requests during the development phase and bugs during the testing phase.

On the following chapters, a software developer configuration management related tasks and obstacles will be explored and commented through the 5 “Configuration Management Functional Steps”.

2.6 Configuration Management questions

The following questions can be regularly asked by software developers and their management, and should be answered, to get visibility inside Configuration Management Processes:

- Basic File Management related questions:
  - Who made the last modification in this file?
  - What change did you make in this file?
  - When did you make this change?
  - Which version of which files are part of this release?

- Basic Change Requests Management related questions:
  - What bug are you talking about?
  - I don’t find the information I need in this bug report?
  - What are the complexity and/or risk associated to correcting this bug?
  - What is the priority of fixing this bug?
  - Have we finished implementing this change?
  - Who is the owner of this bug?
  - When was this bug opened?
  - I commented it, haven’t you read it?

- Not so Basic Configuration Management specific questions:
  - Why was this change made in this file?
  - Why do we release a new software version?
  - Are we sure we did not forget to integrate a file in this release?
  - Are we sure not to wrongly integrate a file that was not intended to be released yet?
  - Can you show me Project/Global Change trends?
  - Can you show me Project/Global Change statuses?

The ability to answer reliabily none, part or all these questions can be measured in 5 Configuration Management functionality steps. Here “answered reliabily” means, answered with help from tools and not just answered by human goodwill and memory.

3 CONFIGURATION MANAGEMENT STEP 0: NO FILE AND CHANGE

 © 2006 Eric Mariacher. All rights reserved.

MANAGEMENT TOOLS

Step 0 is usually where software developers start their career: jumping directly into software coding. After a few days of coding the need to do some "configuration management" arise, and is usually solved the following way:

Source Files back-ups and releases are managed manually.

Change Requests are manually managed. An example for managing change request is shown in the table below:

Change request title	Change request type	Change request description	Priority	Status	Owner
CR1 description	bug	This is a short description	high	open	myself
CR2 description	Feature request	This is another short description	medium	working	Someone else
CR3 description	bug	This is yet another short description	low	closed	

All of the Configuration Management questions, defined in previous chapter, can be answered by the developers but none of them can be reliabily answered (i.e. automatically with no human intervention).

4 CONFIGURATION MANAGEMENT STEP 1: AT LEAST FILE MANAGEMENT TOOL

After having suffered doing their first and hopefully last "CM step 0" software project, the developer usually purchase or uses some dedicated file and release tool, and even perhaps a change request management tool.

File and release management Step 1 tool manages the following tasks:

- Handle file versioning: ability to retrieve older versions.
- Basic release generation: each release is made of a list of file versions.

Step 1 tools usually answers "Basic File Management" related questions as specified in a previous chapter.

Change Request Management Step 1 tool, if used, has functionalities that are usually able to answer "Basic Change Request Management" related questions. It is not integrated in any way to the file management tool.

 © 2006 Eric Mariacher. All rights reserved.

5 CONFIGURATION MANAGEMENT STEP 2: LOOSELY LINKED FILE AND CHANGE MANAGEMENT TOOLS

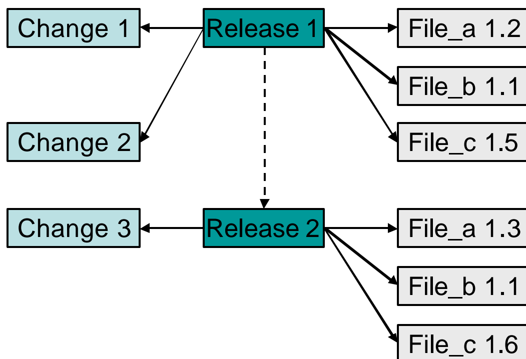
Now our developer regularly working on software projects has the following problem: he is asked to tell a customer (a real one or his boss) what where the differences between release 1.1.4 and release 1.2.3 on a months old previous project. When he looks at the differences between releases 1.1.4 and 1.2.3, if following step 1 methodology, he only has a list of files whose version is different but with no clues about the purposes of differences. To get some of these clues, the developer can adopt a step 2 methodology where file and change requests tools are loosely linked.

Loosely linked, here means that when documenting Releases, change requests completion dates available in the change request tools are taken into account in addition to what is done by step 1 tools. A release note is now documented with:

? A list of file versions as done in step 1.
? A list of integrated change request based on completion date.

Release note documentation is based on the following relations:

? **completion date => integrated change requests**



Step 2 answers “Basic File Management” and “Basic Change Request Management” related questions. Based on dates of change status evolution we can somewhat unreliable answer some of the “Not so Basic Configuration Management specific questions”:

- ~~Why was this change made in this file?~~
- Why do we release a new software version? -> *some change requests have been closed on date xx/xx/xx*
- ~~Are we sure we did not forget to integrate a file in this release?~~
- ~~Are we sure not to wrongly integrate a file that was not intended to be released yet?~~
- Can you show me Project/Global Change trends? -> *some change requests have been closed on date xx/xx/xx*
- Can you show me Project/Global Change statuses? -> *some change requests have been closed on date xx/xx/xx*

 © 2006 Eric Mariacher. All rights reserved.

6 CONFIGURATION MANAGEMENT STEP 3: STRONGLY LINKED FILE AND CHANGE MANAGEMENT TOOLS, FILE ORIENTED RELEASE BUILDING

The developer having been several times by his boss to document differences between releases decides to adopt a step 3 methodology where file and change management tools are strongly linked which means that “every time you check-in a file in the file management tool you are obliged to link this action with a listed change in the change management tool”.

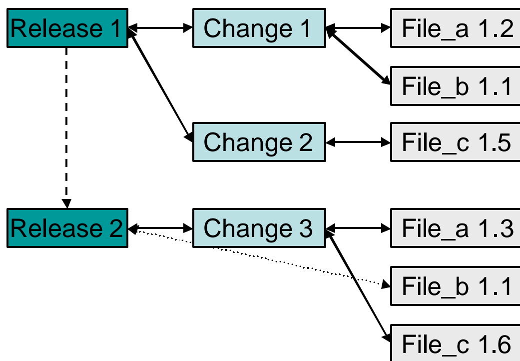
“File oriented release building” means that developers still build a release based on a labeled set of files. As all file check-ins must be linked to change requests, all change requests belonging or affected by this release can also be listed.

A release note is now documented with:

- A list of file versions as done in step 1.
- A list of integrated change request that have been linked to some file check-in.
- A list of integrated change request based on completion date but that have not been linked to some file check-in.

Release note documentation is based on the following relations:

- **file versions => integrated change requests**
- **completion date => integrated change requests**



Step 3 answers “Basic File Management” and “Basic Change Request Management” related questions. For change requests that have file check-ins linked to them Based on dates of change status evolution we can answer “Not so Basic Configuration Management specific questions”.

- Why was this change made in this file? *-> file check-in is linked to this change request*
- Why do we release a new software version?
- ~~Are we sure we did not forget to integrate a file in this release?~~
- ~~Are we sure not to wrongly integrate a file that was not intended to be released yet?~~
- Can you show me Project/Global Change trends?
- Can you show me Project/Global Change statuses?

CM step 3 way of doing things are generally relevant when there are few developers (1 or 2) working on the project at the same time, because informal communication is usually good enough. Developers take the release builder role.

 © 2006 Eric Mariacher. All rights reserved.

7 CONFIGURATION MANAGEMENT STEP 4: STRONGLY LINKED FILE AND CHANGE MANAGEMENT TOOLS, CHANGE REQUEST ORIENTED RELEASE BUILDING

Our developer is not alone anymore. He now works in a big project with several other colleagues. But he has been asked to take care of release generation in addition to its regular developer job. Every time a release is to be generated, our developer who is now a release builder (also sometimes called integrator), has to ask all his colleagues what is the exact list of file versions they want to see integrated in the next release, and then he generate a label based on this list of files versions lists. An easier way would be to build a release based on a set of change requests to integrate. That is the step 4 methodology.

Each change request selected automatically draws the files versions needed for its implementation. As all file check-ins must be linked to change requests, all file versions linked can be listed, thus having the list of integrated change requests is enough to correctly and safely document the release note.

A release note is now documented with:

- A list of integrated change request that have been linked or not to some file check-in.
- Optional: a list of file versions as done in step 1.

Release note documentation is based on the following relations:

- **release <=> integrated change requests <=> file versions**

When there is a huge number of files (more then 1000) following CM step 4 methodology simplifies communication as a release can now be identified by a list of change requests and not just some huge list of file versions. There is a bi-directional relation between file versions, change requests and releases.

Step 3 answers “Basic File Management” and “Basic Change Request Management” related questions. For change requests that have file check-ins linked to them Based on dates of change status evolution we can answer “Not so Basic Configuration Management specific questions”.

- Why was this change made in this file?
- Why do we release a new software version?
- Are we sure we did not forget to integrate a file in this release? *-> all files needed to fix a change request is automatically extracted at release building*
- Are we sure not to wrongly integrate a file that was not intended to be released yet? *-> only files needed to fix a change request is automatically extracted at release building*
- Can you show me Project/Global Change trends?
- Can you show me Project/Global Change statuses?

CM step 4 way of doing things are generally mandatory when there are more than 2 developers (1 or 2) working on a project at the same time. It is also more convenient when dealing with a huge number of source files.

8 CONFIGURATION MANAGEMENT CMMI LIMITED APPRAISAL

In the table below a table is comparing ability of the different steps to address some specific CMMI Configuration Management specific practices and some of their sub-practices.

 © 2006 Eric Mariacher. All rights reserved.

5 STEPS OF CONFIGURATION MANAGEMENT FUNCTIONALITIES

	step 0: no file and change management tools	step 1: at least file management tool	step 2: loosely linked file and change management tools	step 3: strongly linked file and change management tools, file oriented release building	step 4: strongly linked file and change management tools, change request oriented release building
level of integration between file and change management tool	**not applicable**	**None or not applicable**	**loose**	**strong**	**strong**
base for building release	**file**	**file**	**file**	**file**	**change request**
CMMI CM Specific Practice and Subpractice					
CM.SG1 Baselines of identified work products are established.					
SP 1.3 Create or release baselines for internal use and for delivery to the customer.					
SubP3 Document the set of configuration items that are contained in a baseline.	Same as SP3.1				
CM.SG2 Track and Control Changes					
SP 2.1 Track change requests for the configuration items.					
SubP2 Analyze the impact of changes and fixes proposed in the change requests.				easier to do as all file check-ins are linked to a change requests	easier to do as all file check-ins are linked to a change requests
SP 2.2 Control changes to the configuration items.					
SubP3 Check in and check out configuration items from the configuration management system for incorporation of changes in a manner that maintains the correctness and integrity of the configuration items.					possibility to check for “cross file/change request" conflicts.
SubP4 Perform reviews to ensure that changes have not caused unintended effects on the baselines.					possibility to check for “cross file/change request" conflicts.
SubP5 Record changes to configuration items and the reasons for the changes as appropriate.			Low tool based reliability	medium tool based reliability	100% tool based reliability
CM.SG3 Integrity of baselines is established and maintained.					
SP 3.1 Establish and maintain records describing configuration items.	1 some comments based on developers good wll	1 list of files version	1 list of files version 2 list of integrated change requests based on date of status change * completion date => integrated change requests	1 list of files version 2 list of integrated change requests based on file versions checked-in 3 list of integrated change requests based on completion date * file versions => integrated change requests * completion date => integrated change requests	1 list of integrated change requests 2 list of file versions based on integrated change requests * release <=> integrated change requests <=> file versions

© 2006 Eric Mariacher. All rights reserved.

9 WHAT IS THE "CROSS FILE/CHANGE REQUEST" CONFLICT?

The "cross file/change request" conflict is linked to the fact that change requests modifications may have incompatible impacts on source files (CM.SP2.2.subp3 & 4 [PA159.IG102.SP102.SubP103] & [PA159.IG102.SP102.SubP104]). It is only relevant when following step 4 methodology, where releases are built based on a change request list rather than a list of files versions. "Cross file/change request" conflict occurs in the following case:

	File_A	File_B
Release 1	File_A version 1	File_B version 10
Change request CR1	File_A version 2	File_B version 11
Change request CR2	File_A version 3	

- Release 1 is made from File_A version 1 and File_B version 10.
- CR2 is implemented after CR1.
- CR1 affects File_A version 2 and File_B version 11.
- File_A version 2 needs File_B version 11 to work properly.
- CR2 affects File_A version 3.
- File_A version 3 is File_A version 2 plus some extra changes not affecting CR1 modifications.

The release builder generates release 2 from release 1 plus correction for CR2 but not including CR1 -> *this will not work because File_A version 3 having version 2 modifications it also needs File_B version 11 to work properly.*

This is the "cross file/change request" conflict.

Release 2 can only be made of:
- release 1 plus correction for CR1.
- Release 1 plus corrections for CR1 and CR2.

When every file check-in is strongly linked to change requests, it is possible to detect "cross file/change request" conflict by using some appropriate tools.

10 CONCLUSIONS

The more file management and change management tools are integrated between themselves, the more these tools can help to answer configuration management related questions and to fulfill with increased confidence some specific CMMI CM subpractices. This is especially the case for all CM.SP3.1**[PA159.IG103.SP101]** and CM.SP1.3.subp3 [PA159.IG101.SP103.SubP103], where an increase in automatisation and confidence can be observed when going from step 0 to step 4.

At step 4, a new abstraction level, "linked change request", has been introduced between files and releases. This abstraction level is reliable provided that:
- It is mandatory for the developer to link file check-in to a change request.
- Releases generation are change request based opposed to file based as in step 3.

This "linked change request" concept is enabling the release builder to become independent from knowing source files at step 4 for the following reason. At steps 0, 1, 2 and 3 the release builder cannot exist by itself. Release building is done by developers because it requires knowledge of which version of which file to integrate in the next release. At step4, though, the release builder can just build releases by specifying which changes to integrate; he does not need to have knowledge of the files below changes. Each change request selected automatically draws the files versions needed for its implementation.

The bigger the projects are in terms of source files and developers the more the tools become necessary to help to deliver "controlled" releases. As previously said, when reaching Configuration Management functionality step 4, the tools enable the developers to concentrate on their primary job which is to code and leave the sometimes painful job of generating and documenting releases to the release builder.

 © 2006 Eric Mariacher. All rights reserved.

11 ABOUT THE AUTHOR

Eric Mariacher has held various positions in embedded software development, coder, architect, project manager at IBM. He is now functional manager for wireless keyboards and receivers at Logitech. He has experience in driving projects of different sizes beginning with 8Kbytes of code running in 8bits microcontrollers coded by one developer to projects requiring tens of developers, several megabytes of code and thousands of files on 32bits microcontrollers and PC computer.

Eric Mariacher main centers of interests are to set-up the right environment for software developers to deliver products, wwhile giving management the best visibility on software development process from requirement phase to field support phase. Setting up the right software development often requires reengineering software development processes. Eric Mariacher enjoys driving these software development processes reengineering measured following CMMI best practices.

Eric Mariacher can be reached at eric.mariacher@gmail.com or eric_mariacher@logitech.com .

12 BIBLIOGRAPHY

[SEI 2002-08a] "CMMI for Software Engineering (CMMI-SW, V1.1): Staged Representation"; CMMI Product Team; Software Engineering Institute; 2002-08; CMU/SEI-2002-TR-029, ESC-TR-2002-029; <http://www.sei.cmu.edu/publications/ documents/ 02.reports/ 02tr029.html>

 © 2006 Eric Mariacher. All rights reserved.