Integrating Configuration Management and Process Management Based on Life Cycle Control

Xin Peng, Wenyun Zhao, Chongxiang Zhu
Computer Science and Engineering Department, Fudan University, Shanghai, China
{pengxin, wyzhao, cxzhu}@fudan.edu.cn

Abstract

Currently Configuration Management (CM) tool and Process Management (PM) tool are both largely applied in software organizations. However, neither of them can effectively support the complex evolutions in today's software development. So we must integrate CM and PM together to provide a systematic framework for managing evolutions. This paper introduces a configuration management framework based on life cycle control of the system. In this framework, the life cycle is represented by a series of iterations, which can be started by change requests. Each iteration is based on the process model and CM activities such as check in/check out are performed in it. Strict life cycle control can make evolutions of artifacts more methodical. Furthermore measurements can be effectively performed on the rich information stored in the repository. Process-based reuse is also simplified in this framework

1. Introduction

Configuration Management (CM) [1] and Process Management (PM) are both key disciplines for high-integrity system development. Many software organizations have introduced CM or PM tools, or both in their everyday development activities. However, the managers and developers are still bothered with questions such as "how much of the project has been completed" [2], "what part of the system is the bottleneck, which delays the whole schedule", or "how the change requests have been treated and what have been produced to meet them", etc. Neither CM nor PM tool can help to answer these questions separately.

CM is concerned with controlling and recording the evolution of all software development artifacts [3]. SEI [4] has concluded four SCM model: the check out/check in model, the composition model, the long transaction model and the change set model. Most of them focus on recording and managing evolution path of configuration items beyond their context, such as objectives and process information. The long transaction model combines a series of associated modifications within a transaction, but the concept mainly focuses on local workspace support and concurrency control. The change set model focuses on supporting management of logical changes to system configurations [4], but the change set is a kind of increment and has no process information.

Version evolution is not isolated. It is associated with specific cause and objective, which can bring a series of development activities and impact the evolution of many other artifacts. Software process management can help to provide information about these. A software process consists of the set of activities performed during software development, their scheduling, and the objects manipulated [5]. So it is stated in [6] "A CM tool is needed to control the evolution of a software system, i.e. to keep track of what. We also need a PM tool to model, execute, and record software processes, i.e. the how and why." The configuration management can also provide extra information for process management, such as "what products and versions have been produced in the activity", or "how much effort has been spent on the change request", etc. This can make the process management more perfect.

Some researchers have realized the need of associating CM with PM. However, we believe that they should be integrated more closely. The CM activities (such as check in/check out) should be controlled by PM, and CM data should be utilized to reflect the status and quality of PM. Aiming at this we proposed LCBCM (Life Cycle Based Configuration Management), an integration framework of CM and PM. In this framework, process is represented as iterations of the process model at various hierarchies of the system (structured by subsystems).

The remainder of this paper is organized as follows. Section 2 introduces related works and compares them with LCBCM. Then related concepts and the basic subsystem-based structure of CM are described in Section 3. Detailed description of the framework is given in Section 4. Section 5 introduces the measurements performed on LCBCM. Finally we provide information on implementation and validation of LCBCM and draw our conclusions in Section 6.

2. Related works

Nowadays the problem of providing more integrated framework for managing changes has been realized by researchers. Lindsay [3] proposes a generic framework for subsystem-based configuration management, in which arbitrary subsystems can be put under configuration control and have their change histories recorded to provide hooks for tracking the cause and outcomes of changes. The framework views the system as a hierarchical collection of subsystems. It provides a series of change types for items of subsystems, such as add, delete, modify, derive, replace, etc. So it can deal with a large set of changes on the subsystem. However, artifacts of different process steps are managed together in a subsystem, so phases of process are not distinguished and traceabilities are missing. This weakens the process capability of the framework.

Conradi [6] describes process management in a software engineering environment, which connects CM and PM through a common data model to integrate "passive" software products and “active” tasks. However, the integration is performed on the data hierarchy and lacks a complete process to guide users.

Ivica Crnkovic [7] proposes a model for managing requirements using CM functions, in which each requirements specification is structured by hierarchic RS (Requirements Specification) Items. And RS Items, Change Requests and system modules are related to provide a comprehensive process for changes. However, changes do not always start at requirements and these three kinds of objects can not cover the whole development process. In another paper [8] Ivica Crnkovic describes a measurement approach performed on Change Requests, in which Change Requests (CRs) are basic items and seen as logical changes to be made in a software system. Physical changes and their information are related to certain CR. This approach associates evolutions with change management and provides a sound way to measure the process, but a systematic process is missing.

3. CM based on subsystems

In LCBCM, the system is hierarchically structured by subsystems and components, as showed in figure 1. Subsystems are logically coherent collections of software development artifacts, including code, documentation and test sets [3]. A sub-system consists of configuration items, components and other subsystems, these artifacts represent the whole life cycle of the subsystem. Subsystems are also basic units of development arrangement and process management.

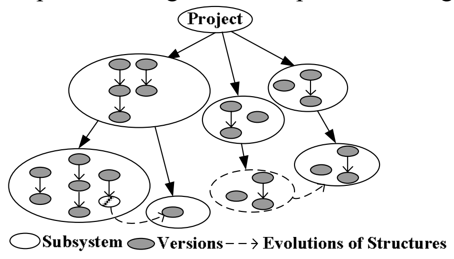


Fig. 1. Evolutions based on subsystems

3.1. Definition

Configuration Items (CI): CIs are atomic version entities and the atomic units of evolution.

Process Property: CIs belong to certain phases of development, such as analysis, implementation, test, according to their characters. Accordingly each activity belongs to certain process phase and performs version evolutions on corresponding CIs.

Evolution: Actions like creation of new CIs or versions, branching, merging, etc..

Active (Inactive) status: The status of a subsystem and CI can be active or inactive. Inactive entities can not evolve any longer. However, they are still a part of the evolution history of corresponding subsystems.

Process Configuration Items (PCI): For a process step of certain subsystem, the milestone configuration of CIs of the same process property is called process baseline of the step. A process baseline corresponds to a special CI, which is called Process Configuration Items. Each PCI version is a milestone reflecting the overall status of artifacts after certain process activities.

3.2. Subsystem-based structure

There are two kinds of relations between CIs. One is vertical evolution relations between various versions of each CI (the version dimension). The other is horizontal evolution relations between CIs of various process steps (the process dimension), for example the relation between a design document and corresponding source files. Effective management of these relations is the key to ensure orderly evolution of the system.

In LCBCM, each subsystem or component is a separate unit of development. The project can be seen as the root subsystem. Subsystems are also basic unit of process management according to a unified model of the project. A subsystem can contain both CIs and children subsystems. The former are direct artifacts and the latter are refined artifacts belonging to certain parts of the subsystem. For example, there are both holistic design and refined designs for each part of a subsystem. Process management can be performed on various levels. This reduces unnecessary associations and can enhance the concurrency between subsystems.

The development mode of components differs much from that of projects. Components can evolve and be reused beyond certain projects. So they can have their own process models. A component will not simply evolve to satisfy the need of a project, since it does not serve for certain project.

3.3. Subsystem-based Evolutions

Generic evolutions include creation of CIs, versions, branching/merging, etc. Subsystem-based evolutions still include creation of subsystems and recomposition of subsystems. Figure 1 presents the evolution types supported in LCBCM, in which dashed arrows represent structural changes of subsystems. CIs and subsystems can be created, deleted or moved from one parent subsystem to another. The deletion here is not true "deletion", but just sets the CI or subsystem to be inactive. A subsystem provides evolution environment for CIs and children subsystems in it.

4. The LCBCM framework

CM manages artifacts produced in the development process, controls the changes to the software and its components, and keeps tracks of evolution [1]. The control should involve not only separate evolutions, but also the complete process of evolutions. LCBCM aims at integrating PM and CM through tracking and recording activities during Software Life Cycle (SLC).

4.1. Overview

The PM in LCBCM is performed on subsystems and components separately. Processes in LCBCM can be considered to be iterative. Each iteration is a complete or partial process execution with particular objective. The LCBCM framework is showed in Figure 2. For a subsystem or component each iteration is fired by an evolution demand. There are planned and unplanned changes during the SLC [9], both of which are sources of iterations. According to the evolution objective, the evolution request can be assigned to one or several subsystems, and the evolution requirements can be specified. Then iterations are started and carried out in involved subsystem. Each iteration consists of several steps. During the execution of each step, a series of version activities can be carried out. In LCBCM, each CI has a process property and can only evolve during the execution of certain steps. All the CM and PM behaviors are recorded in the life cycle database. In this way evolutions of artifacts are attached to iterations. So artifacts can evolve more methodically and evolutions can be managed more effectively.

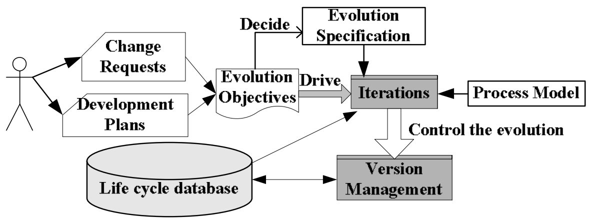


Fig. 2. The LCBCM framework

PM can ensure the system under development to be well supervised and scheduled. Measurements can be performed using data stored in the CM database to present the status of development and help developers to supervise the process. The advantage of this kind of measurements is the accuracy of the data and that it needs no extra efforts [8]. The measurements can provide even more meaningful information, such as efforts, time spent on each development objective or change request. These parameters can be used to supervise and adjust the schedule when necessary.

4.2. Iterations

A process model defines the work flow of a series of development activities, and iteration is an instance of the workflow. Figure 3 presents a process model, which contains four steps. CM is concerned with controlling and recording the evolution of all software development artifacts, not just source code control [3]. So each subsystem maintains the whole life cycle for the corresponding part of the system.

During iteration, when a step completes its activities, a new version of the PCI is created, which represents the updated state of the step. In LCBCM, versions of PCIs are created by process integration. Process integration is the composition of certain versions of related CIs, including CIs of the step and PCIs of the same step on the direct children subsystems (figure 3). For example, the whole design of S includes CIs of the design phase and PCIs of design steps of S1 and S2.

Iteration is an ordinal execution of the steps. The execution can be complete or partial. For example, in the SLC of subsystem S in Figure 3, there exist both a complete iteration A and a partial iteration B. Iteration B may be a performance enhancement process for a change request from the customer, so it begins at the design phase. After the iteration, each involved step produces a new PCI version. These versions form an evolution chain, representing the result of the iteration. Two types of changes are identified during the SLC [9]:

- **Planned changes**. These tend to be changes that add capability.
- **Unplanned**. These tend to be changes that repair defects and unplanned additional requirements.

Iteration of a subsystem can also drive evolutions of other subsystems. So for each iteration, there may be four kinds of evolution sources:

- **External change requests**: They come from change management system and are assigned to certain subsystem or component to implement. So the iteration aims at fulfilling the requests. Several related change requests can be merged to be implemented in iteration. The following iteration and version evolutions can be considered as results of the request.
- **Internal development objectives**: They are scheduled development plans, such as prototype development plan, alpha test version release plan, etc. This is the normal way of system evolution. Generally, the impact of this kind of iterations is much greater and may last longer. The development objective determines the evolution objective of the iteration.
- **Demands of upper subsystems**: Iterations of a subsystem can be decomposed to its children subsystems. After the decomposition, an iteration of the children subsystem is started. The evolution objective is determined by the parent subsystem, which can be specified with the decomposition.
- **Demands from related subsystems**: A subsystem may be related to others. In these cases, evolution of a subsystem may drive other subsystems to evolve and the objective is to maintain the relations. This situation is different from that of the demands of upper subsystems, in which children's iterations are parts of the upper subsystem's.

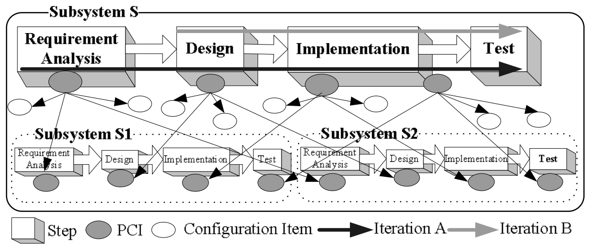


Fig. 3. Iterations and process integration

4.3. Process management in each iteration

In LCBCM, an iteration can start at each step of the model, then the following steps can startup in turn. So each step can have four kinds of states as follows.

Executing: In iteration each step is started by stimulations from the previous step except the starting step. For a step, there may be several stimulations to be treated at one time, and then the developers need to determine the priorities of them.

Suspended: Iteration may be suspended after started, e.g. when an urgent bug-fixing request comes. All the unsubmitted versions in the suspended iteration can not been seen in other iterations. However, they are still valid when the iteration is resumed.

Finish: In iteration each step ends with process integration and a new PCI version. Before the process integration can be done, two preconditions must be met. One is that there are no checked out CIs any more. The other is that if the iteration has been decomposed, corresponding steps of those involved subsystems have completed process integrations first.

Idle: A step is idle if it is not in any iteration. When idle, CIs of the step can only be checked out in read-only mode. However, other steps of the same subsystem and steps of the children subsystems can still be active to evolve.

In LCBCM, when a step ends with process integration it reaches a stable state with some new features. Then the next step can perform development activities according to these new features. The process integration is based on both old and new items and versions. Before the process integration, the step is in an unstable state. In this period, the CIs may evolve greatly, for example several versions of the same item may be produced. However, not all the new items or versions will be included in the following process integration to form a new configuration, because some of them may only be intermediate versions.

When a step starts in an iteration certain version of the PCI should be selected as the common base of the development. So evolution relations between versions of the PCIs can be established. Figure 4 shows the relations, in which each pane denotes a version of the PCI. In the figure horizontal arrows represent traceabilities between steps during each iteration and vertical arrows represent traceabilities between versions of a PCI. Horizontal traceabilities can determine vertical traceabilities. That means the root version of evolution is determined by the previous step in the iteration: the version corresponding to the evolved version in the previous step. For example when the PCI of Requirement Analysis evolves from version 1.0 to 1.1 (Figure 4), the affected version of Design PCI is 1.0. So the evolution in Design phase will start at the newest version of each branch deriving from version 1.0. If there exist multiple branches deriving from the influenced version, things may be a little more complex. In figure 4, there are two branches derived from the version 1.0 of the Design PCI and both of them must evolve to respond the stimulation from Requirement Analysis. So, version 1.1 and 1.0.1.0 of the Design PCI evolve to 1.2 and 1.0.1.1 separately. Accordingly, the following steps of Implementation and Test will evolve.

For a step, all the evolutions in iteration is a transaction, which can be committed with process integration or rolled back when cancelled. Before that, all the uncommitted changes can not be seen in other iterations. This is something like the long transaction model [4]. It should be emphasized that the basic unit of status control is not subsystem but step. That means different steps of a subsystem can be involved in different iterations, so parallelism is promoted greatly.

In some cases there are also needs for an executing step to switch to another iteration. So the iteration is suspended and all uncommitted changes are preserved. Then the new iteration can start based on specific PCI version. If the two iterations occur in the same branch, a new branch will be created for the new iteration. These branches can be merged when necessary.

Iterations can be decomposed between hierarchies, i.e. the parent subsystem can drive some of its children subsystems to evolve. Decompositions can be recursively performed until the leaf subsystems are involved. After decomposition, each involved child subsystem gets a demand to start a new iteration. After that iterations are performed on these subsystems separately. The pace of the parent subsystem is restricted by the involved children subsystems. Process integration of the parent subsystem can only be done when the same steps of all the involved children in the decomposition have finished. However, evolutions of children systems can exceed the parent. For example: an iteration of subsystem S has been decomposed to children subsystem S1 and S2 and S is waiting for S1 to end the design step; at the same time S2 can have exceeded to the implementation step. Relations such as dependency or consistency can also drive other subsystems into iterations. After that, the two iterations can process separately.

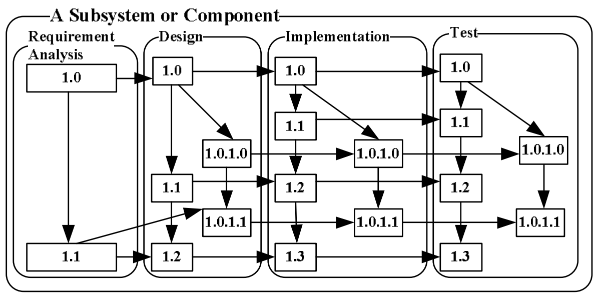


Fig. 4. Evolution relations between PCI versions

5. Process measurement

One of the big challenges of a large software development process is to keep track of the current sate of the project [8]. However, in a project involving more than several developers, it is difficult to grasp the status of the project, such as progress of the project, the bottleneck, efforts of implementing each development objective, etc. CM is located at the center of software development activities. There is a huge amount of information stored in SCM repository [8], which can objectively reflect the status. Furthermore, no extra effort is needed to collect the data. So, CM-based process measurements can help us to understand, control and improve the process. In LCBCM, systematic management of life cycle on subsystems and components provides rich information about efforts, quality, scheduling, etc. So, CM-based process measurement can be performed.

Efforts and progress are relatively easy to measure. In LCBCM, each evolution of CIs is associated to a step of certain iteration, so the measurement can be performed up to the level of process steps.

- **Efforts of each iteration and step**: This can be measured by both time efforts and quantity efforts. The former are the lasting time of each iteration or step. The latter includes the quantity of CIs and versions produced in each step or iteration. These parameters can present the load of a step and the cost for a change request or development objective.
- **Progress measurements**: Progress of the project can be described by records of iterations, such as “how many iterations have been completed”, “what are the iterations being performed”, “which step has the current iteration reached” and “how many CIs and versions have been produced”. Differences of progress between subsystems can also be obtained.
- **Stability**: Stability is an attribute of each step, which can show the changeability of requirements according to various subsystems and steps. For each step, it can be measured by the number of iterations starting at the step and the artifacts modified.
- **Load of each step**: This can be measured by “number of iterations”, “percentage of the active time” and “average waiting time to start an iteration on this step”. If the load distributes unbalanced on subsystems or steps, it may means that the developers are not properly assigned or efficiency of certain part is low.
- **Concurrency**: It can be represented by the degree that multiple iterations can be performed at one time. It can also be measured by the distribution of load indirectly. Generally speaking, it can be promoted by subdivision of subsystems.
- **Baseline analysis**: Each version of a PCI is produced on the basis of specific previous version and implements some new features. So each PCI version can be seen as a accumulation of some baselines. The

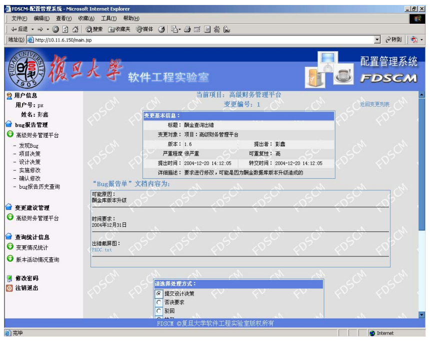
baseline of a version consists of the baseline of the derived version and the change sets performed in the iteration. The baseline analysis can explicitly present the state and logical meanings of each PCI version.

6. Evaluation and conclusion

The goal of LCBCM is to manage evolutions in a controlled way. It is achieved by the strict control of iterations and processes. The hierarchical structure makes process management flexible enough to fulfill evolutions of various granularities. CM-based process measurement is also supported. In this framework, process-based reuse is also simplified. Iterations can establish tracking chains between versions of process baselines. So when certain version of a PCI is selected for reuse, versions of the following PCIs in the chain can be identified to reuse together.

Some features of LCBCM have been implemented in FDSCM, which is a SCM tool developed in our current research project, including a configuration tool and a web-based change management system (figure 5). In FDSCM, change requests from change management system can be assigned to specific subsystem or component to drive new iterations. Then we can track the entire history of a change by change documents and the following iteration. The web-based system also publishes CM-related information for managers and developers. The data provided is obtained by measurements, such as efforts, progress, etc. The status of projects can be explicitly gained from these data.

There are still some limitations in this framework. Measurements obtained in iterations can not feed back to affect the evolution of the process. Further investigation in the future will concentrate on these limitations and enhancing the practicability.



用户号：px
姓名：彭鑫

当前项目：高级财务管理平台
变更编号：1

变更基本信息：

标题：	酬金查询出错		
变更对象：	项目：高级财务管理平台		
版本：	1.6	提出者：	彭鑫
严重程度	很严重	可重复性：	高
提出时间：	2004-12-20 14:12:05	转交时间：	2004-12-20 14:12:05
详细描述：	要求进行修改，可能是因为酬金数据库版本升级造成的		

"Bug报告单"文档内容为：

可能原因：
酬金库版本升级

时间要求：
2004年12月31日

出错截屏图：
PROC.txt

请选择处理方式：
[✓ 提交设计决策]
否决要求
驳回

Fig. 5. The web-based change management system

Acknowledgments. This work is supported by the National High Technology Development 863 Program of China under Grant No.2005AA113120, the Science Technology Committee of Shanghai under Grant Nos.04DZ15022, 055115005.

References

[1] Lu Zhang, Hong Mei, Hong Zhu. A configuration management system supporting component-based software development. COMPSAC 2001.
[2] Kasser J.E., Williams V.R., "What Do You Mean, You Can't Tell Me If My Project Is in Trouble?", First European Conference on Software Metrics (FESMA 98), Antwerp, Belgium, 1998.
[3] Lindsay, P., MacDonald, A., Staples, M., Strooper, P., "A Frameworks for Subsystem-based Configuration Management", Proceedings of Software Engineering Conference, Australian, 2001.
[4] Peter H. Feiler, "Configuration Management Models in Commercial Environments", Software Engineering Institute, Carnegie Mellon University, Technical report, CMU/SEI-91-TR-7, March 1991.
[5] Osterweil, L.J., "Software Processes Are Software Too", Proccedings of the 9th International Conference on Software Engineering, pages 2-13, Monterey, California, April 1987.
[6] Conradi, R., Osjord, E., Westby, P.H., Chunnian Liu, "Initial software process management in EPOS", Software Engineering Journal , Volume: 6 , Issue: 5 , Sept. 1991, Pages:275-284.
[7] Ivica Crnkovic, Peter Funk, Magnus Larsson, "Processing requirements by software configuration management", Proceedings of EUROMICRO Conference, 1999.. 25th , Volume: 2 , 8-10 Sept. 1999, Pages:260-265.
[8] Ivica Crnkovic, Magnus Larsson, Frank Lüders, "Software Process Measurements using Software Configuration Management", Proceedings of the 11th European Software Control and Metrics Conference, Munich, Germany. May 2000.
[9] Donaldson S.E., Siegel S.G., "Cultivating successful Software Development", Prentice Hall PRT, 1997.