Nationaal Lucht- en Ruimtevaartlaboratorium
National Aerospace Laboratory NLR

NLR-TP-2001-351

Delegating configuration management responsibilities within the software development process

M.A. Guravage

Nationaal Lucht- en Ruimtevaartlaboratorium
National Aerospace Laboratory NLR

NLR-TP-2001-351

Delegating configuration management responsibilities within the software development process

M.A. Guravage

This report is based on a presentation held at the DASIA Conference, Nice, France, 28 May – 1 June, 2001.

Division:	Information and Communication Technology
Issued:	November 2001
Classification of title:	Unclassified

Contents

1 Introduction **3**

2 Configuration Management **4**

3 Organization **5**

4 Infrastructure **6**

5 Process **6**

6 Tools **7**

7 Metrics **7**

8 Conclusions **7**

(10 pages in total)

DELEGATING CONFIGURATION MANAGEMENT RESPONSIBILITIES WITHIN THE SOFTWARE DEVELOPMENT PROCESS

Michael A. Guravage

National Aerospace Laboratory (NLR)
guravage@nlr.nl, fax: +31 (0)20 511 32 10

abstract

A software project produces a number of items during its execution, including various documents, programs, data, and test artifacts. Configuration management (CM) is the aspect of project management that is concerned with identifying distinct variants of these items, and systematically controlling how they change. Configuration management is often conceived and conducted as a process distinctly separate from and in service of software development. This separation is reflected in a division of work where software developers build software components while a configuration manager combines the components in various configurations. We relate our experiences within a project named MPTE where CM was coordinated and monitored by a configuration manager, but the responsibility for performing CM activities was shared among the software developers.

Keywords: Software Engineering; Configuration Management; Process Management; Change Traceability

1 Introduction

A software project produces a number of items during its execution, including various documents, programs, and data; all of which can easily be changed. This mutability is a unique feature of software (as compared with products of other engineering disciplines). It provides tremendous power and flexibility, but at the same time adds complexity in project management because anything can change at any time. To avoid losing control over the project in the face of changes, it is essential that the process of change be properly managed. Configuration management is the aspect of project management that focuses exclusively on systematically controlling the changes that occur during a project [1–2]. It consists of a set of activities planned and performed to identify and organize software items, and to control their modification.

In 1999 the Dutch National Aerospace Laboratory NLR was selected to develop the Mission Preparation, Training, and Equipment (MPTE) – a ground based computer system (software and hardware), supporting the preparation, simulation and validation of missions for the European Robot Arm. The robot arm will be located on the Russian part of the International Space Station. Developing MPTE software employed some twenty five software engineers for a period of more than two years.

Early in the planning stage it was decided to organize the project to involve the project's software engineers in routine configuration management activities. The motivation was three fold. First, to share the burden of work and responsibility and so prevent CM from becoming a bottleneck along the project's critical path to completion. Second, to instill a sense of ownership by delegating CM responsibilities to the people who either developed or modified the code. Lastly, we hoped that the competency gained by sharing software development and CM tasks would contribute momentum to the project.

2 Configuration Management

A primary objective of configuration management is to oversee a succession of evolving software configurations until the cycle converges on a final product stable enough to ship [2–3]. In MPTE we identified a minimum set of essential CM functions [4]to affect an orderly transition along a succession of configurations from design to delivery:

- Retrieve the status and change history of any individual file; including its position in the hierarchy of configuration items (CIs). This information is necessary in order to decide when to start testing, produce baselines, or release software.
- Retrieve any variant and revision of any individual file or configuration item. Changes are applied to specific revisions of a particular variant of code and documentation.
- Undo any previously applied change. It may be necessary that a requirement of a future change requires that a previous change be reversed.
- Support simultaneous changes. Concurrent changes are necessary in order to keep step with software development and prevent the configuration management activities becoming centralized and serialized.
- Prevent unauthorized changes. Rules and process models are implemented to allow authorized changes while preventing unauthorized ones.
- Retrieve the status and history of any error-problem-change (EPC) request. A single EPC may affect multiple source files. The ability to link EPC's with specific revision of individual files, and vise versa, is required to identify all associated changes.
- Build multiple MPTE variants for unit, integration, and system testing, as well as baselines and delivery releases.

The realization of these functions is dependent on mapping them to a collection of configuration management facilities as represented in Figure 1. The top row names actions we wish to perform, while the bottom row names facilities invoked to perform them.

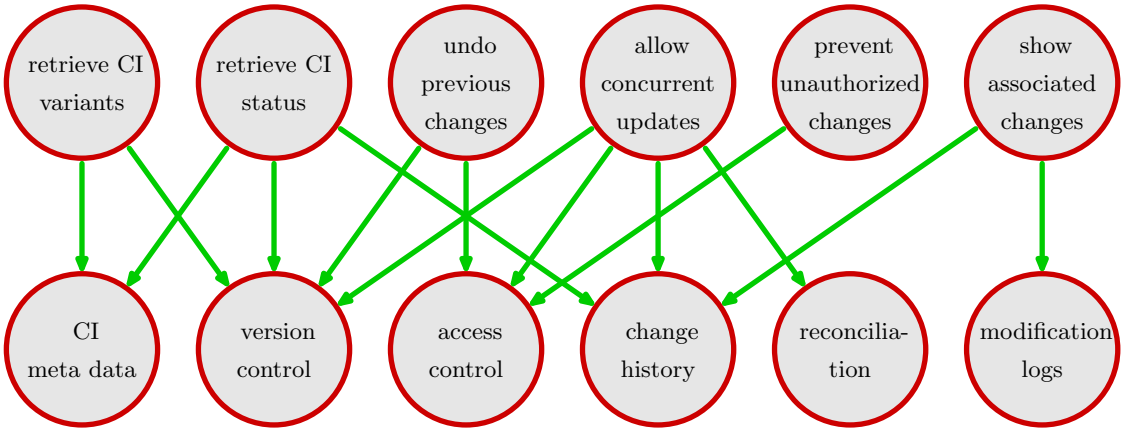


Figure 1 Configuration Management Functions and Facilities

The functions and facilities described above exist and operate in the context of a Software Configuration Management Plan (SCMP). The plan enumerates and describes the CM organization, responsibilities, activities, and procedures specific for MPTE.

In considering a plan, there were several standards from which to choose: one from the IEEE [5], one from NASA [6], and one from the American DoD [7]. The IEEE plan was chosen for its favorable comparison to the other two conducted by Carnegie Mellon University's Software Engineering Institute, and based on the plans' ease of use, completeness, correctness, consistency, and tailorability.

3 Organization

From the outset, we knew that MPTE would be a medium sized project employing dozens of engineers for several years. It was also apparent that configuration management would play a major role in coordinating work performed within the project. Our job was to come up with a project organization and supporting infrastructure that would facilitate software development without compromising the necessary oversight and control. To help us choose between the various ways to organize the CM activities, we asked the following four questions of each alternative:

1. What is the problem you are trying to solve, or conversely, what is the goal you are trying to reach? (Understanding the problem)

2. What has to happen to solve your problem or achieve your goal? (Devising a plan)

3. What rules or process models guide you through your plan? (Carrying out the plan)

4. What have you learned, and can you apply it to improve your process? (Looking back)

From previous experience, one way to organize configuration management is to discriminate between it and software development. The reason for doing so is to maintain oversight and control. The assumption is that, though software development can be shared among many, configuration management is easier to monitor and control when performed by only a few.

The problem for the responsible individual (let's call him the configuration manager) is to personally make sure that all CM tasks are performed correctly. Making CM the responsibility of a single individual requires a very simple plan where the configuration manager can perform every CM task himself. Working under the assumption that he is responsible for performing every CM action, the CM process model becomes centralized; with the result that all CM actions become serialized. Finally, becoming immersed in details leaves precious little time to reflect on how the whole process is working.

This micromanaged approach may be feasible for small software development projects where the distance between developers and the configuration manager is not too great. However, the cost of providing the control necessary to conduct CM increases exponentially with project size and complexity. In medium and large projects, the workload generated by centralizing CM can create a bottleneck on a project's critical path to completion. In this case, separating software development from configuration management is like the left hand not knowing what the right hand is doing.

An alternative to centralized configuration management, and the one chosen by MPTE, is to delegate CM decisions and execution to the lowest appropriate managerial or technical level in the project's hierarchy of responsibilities. Under the organizational principle of subsidiarity, the configuration manager's problem rises from being personally responsible for the smallest CM detail to seeing that the project's software configuration management plan (SCMP) is implemented. With various CM activities shared among the project team, the configuration manager's responsibility is to monitor CM activities, and intervene when necessary to facilitate compliance with the SCMP. The process model here is distributed. Tasks can be performed in parallel because both the organizational style and the underlying technical infrastructure support it.

A decentralized approach has several advantages over micromanagement. Delegating responsibilities means that most CM activities are performed in parallel without the configuration manager's intervention - team members are not waiting for the configuration managers attention. The parallelism inherent in a decentralized approach can work in projects that are otherwise too big or too complicated for a centralized approach. Lastly, a decentralized model involves managing the project rather than the individuals [8].

4 Infrastructure

One product of the architectural and detail design phases of the project was a hierarchy of configuration items (CIs) representing MPTE's structure and content. This hierarchy reflects the project's software directory structure. Each branch in the directory tree represents a CI in the hierarchy of CIs; culminating at the root of the directory tree with a CI that represents all of MPTE.

Each CI is also a work-package; each assigned to a work-package manager. Each work-package manager is responsible for their CI's internal functionality and external interfaces. There were also several work-packages that had no corresponding CI. These represented management and support tasks, including software configuration management.

To facilitate the delegation of CM responsibilities to engineers working on various CIs, the work-package hierarchy was reflected again in the EPC tracking system. This arrangement allowed work to be assigned to individuals under the auspices of specific work-packages. Decision making responsibilities were assigned to the lowest appropriate work-package level.

5 Process

Version control software manages the hierarchy of software CIs, and EPC tracking software manages the corresponding hierarchy of work-packages. A process connects the two and describes the life cycle of a change request. Each EPC describes a specific change, and follows it from inception, through a set of prescribed states, to resolution. Each step along the way has its own entry and exit criteria; including permission to proceed. Figure 2 shows a set of states modeled on the waterfall process model [9].

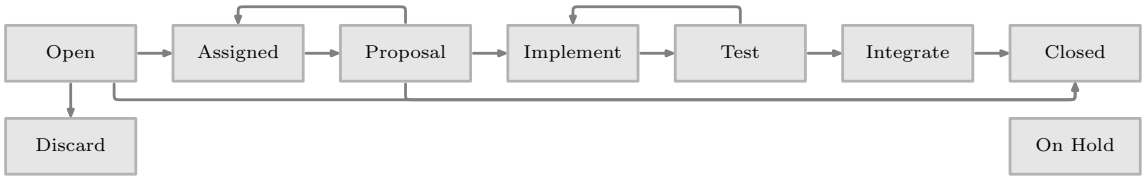


Figure 2 EPC State Transitions

An EPC begins its life assigned to the work-package owned by the change control board (CCB), which evaluates it and assigns it to both an individual and the appropriate CI - work-package pair. The person to whom the EPC is assigned performs an analysis and submits a proposal which, when approved, can be implemented, and so on until the EPC is closed. The point is that there is a well defined process to guide, approve, and monitor the progress of each EPC - without the direct intervention of the configuration manager.

At any one time, any team member can ask the question, "Which EPCs are assigned to me?" or, "How many EPC proposals are awaiting approval?" or, "What are the associated changes to those made in this or that file?" The link between CIs and EPCs allows queries to be answered from information in both the version control and EPC tracking databases.

6 Tools

The configuration management functions and facilities from Figure 2 are supported by several software tools. We use the concurrent version control system (CVS) to perform version control and configuration management, and KEYSTONE to perform change request tracking. CVS needs little introduction. It is the well known successor to RCS, and supports branching and reconciliation of conflicts due to concurrent updates. KEYSTONE is a web based change request tracking tool written in HTML and PHP, an HTML-embedded scripting language, which provides access to MPTE's EPC database.

CVS and KEYSTONE were linked via former's several commit triggers. One trigger called a custom MYSQL client program. It used information gleaned from the CVS log message and the change request database to inquire whether the EPC associated with the files being committed possessed the necessary approval.

7 Metrics

As MPTE progressed, and the version control and EPC databases grew, it was possible to extract various metrics from them - see Figures 3 & 4.

From the CVS activity log a distribution of software repository activity was extracted (Figure 3.1), demonstrating that sharing CM activity among the members of the development team actually worked. The flurry of activity in April and June 2000 corresponds with the integration and system testing.

From the CVS repository, lines of code were counted on a monthly basis for the period of a year beginning in March 2000 (Figure 3.2). The graph shows that the number of lines and percentage of comments remained constant; which might indicate that there are still some code efficiencies to be found. Efficiencies near the end of a project are often reflected in a slight drop in the total lines of code.

From the KEYSTONE database, the jump in reported defects in April and June corresponds to integration and system testing respectively (Figure 4.1). And while the number of defects continued to grow through December 2000, the rate of growth leveled off by the end of the period for which data was available.

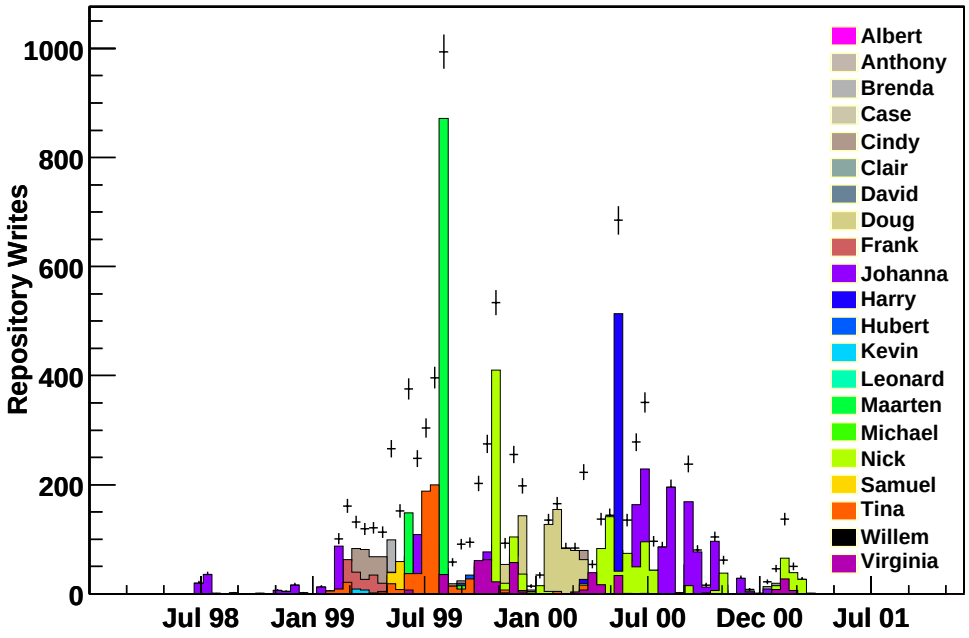
Lastly, from the CVS and KEYSTONE databases, the rate of defects per line of code leveled off at slightly more than one defect per two thousand lines of code (Figure 4.2). Such information helped management to decide when MPTE would be released.

8 Conclusions

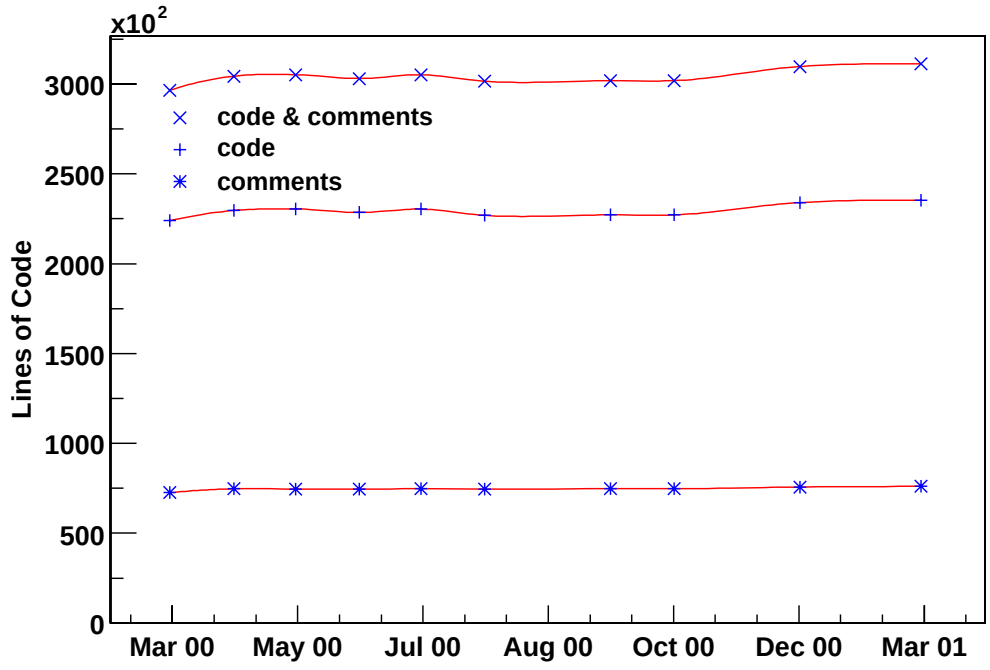
We observed that the implementation of configuration management is often limited by the need to control this key element of any software development project. We found in MPTE that a project organized on the principle of subsidiarity, and supported by a taylored technical infrastructure, allowed CM responsibilities to be shared among the entire software development team without loss of either control or oversight.

Delegating CM resopnsibilities allowed software development and configuration management to preceed simultaneously and in parallel. The job of the configuration manager became managing the implementation of the SCMP rather than overseeing the individual CM activities.

The information collected in the CVS and KEYSTONE databases eventually became a kind of project memory. Not only did the the data yield project metrics, but the information collected for individual EPCs was used to inform subsequent managerial and technical decisions.

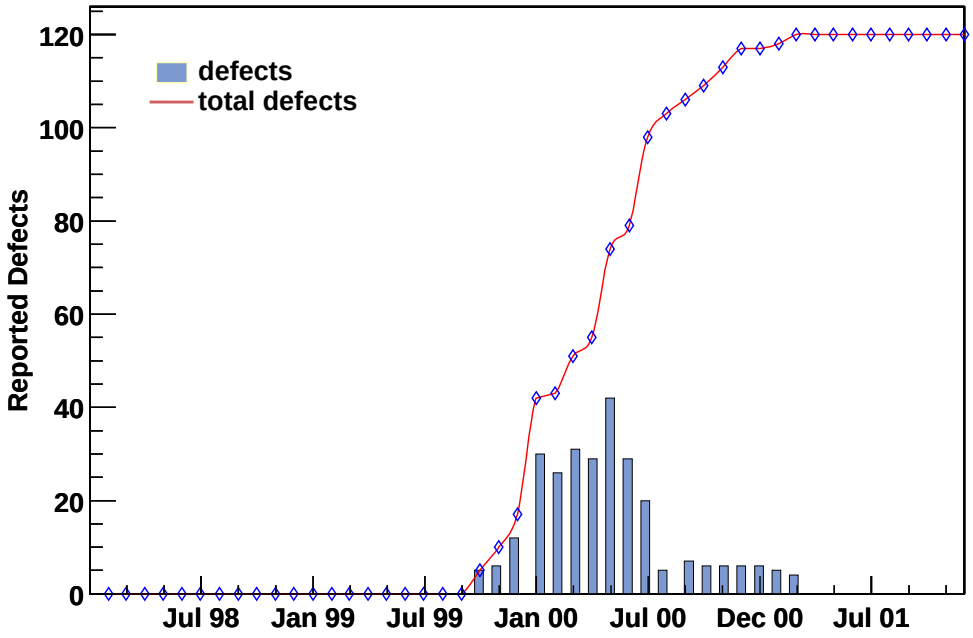


1. software repository activity

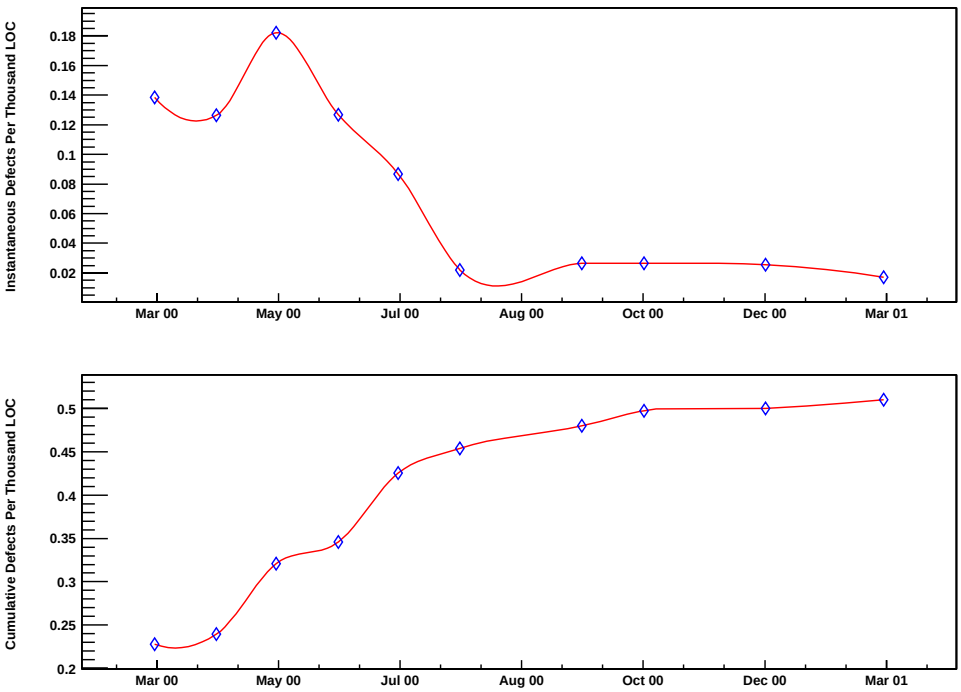


2. lines-of-code

Figure 3 Source Code Metrics



1. defects vs time



2. defects vs loc

Figure 4 Defect Metrics

References

1 Bersoff, E. H. (1979). Software configuration management: A toturial. *IEEE Computer*, pages 6-14.

2 Bersoff, E. H. (1984). Elements of software configuration management. *IEEE Transactions on Software Engineering*, pages 79-87.

3 Humphrey, W. S. (1989). *Managing the Software Process*. Addison-Wesley.

4 Jalote, P. (2000). *CMM in Practice*. Addison-Wesley.

5 IEEE Std 829-1998. (1998). *IEEE Standard for Software Configuration Management Plans*. American National Standards Institute.

6 NASA Sfw DID 04. (1986). *Software Configuration Management Plan Data Item Description*. National Aeronautics and Space Administration.

7 DoD STD 2167A. (1985). *Military Standard for Defense System Software Development*. Department of Defense.

8 Weinberg, G. M. (1991). *Quality Software Management*, volume 1: Systems Thinking. Dorset House.

9 Royce, W. W. (1970). Managing the development of large software systems. *Proceedings of IEEE WESCON*.