4

Configuration Management

LEARNING OBJECTIVES

- To understand the main tasks and responsibilities of software configuration management
- To be aware of the contents of a configuration management plan
- To appreciate the interplay between the role of configuration management in software development and the capabilities of supporting tools

Careful procedures are needed to manage the vast number of elements (source code components, documentation, change requests, etc.) that are created and updated over the lifetime of a large software system. This is especially true in distributed development projects. It is called configuration management.

In the course of a software development project, quite a few documents are produced. These documents are also changed from time to time. Errors have to be corrected, change requests have to be taken care of, etc. Thus, at each point in time during a project, different versions of the same document may exist in parallel.

Often too, a software system itself is not monolithic. Software systems exist in different versions or configurations. Different versions come about when changes are implemented after the system has been delivered to the customer. From time to time, the customer is then confronted with a new release. Different versions of components of a system may also exist during development. For instance, if a change request has been approved, a programmer may be implementing that change by rewriting one or more components. Another programmer, however, may still be using the previous version of those same components.

Different configurations also come about if a set of components may be assembled into a system in more than one way. Take, for example, the system called ACK, the Amsterdam Compiler Kit (Tanenbaum et al., 1983). ACK consists of a set of programs to develop compilers for ALGOL-like languages. Important components of ACK are:

- front ends for languages such as Pascal, C, or Modula-2. A front end for language X will translate programs in that language into the universal intermediate code EM;
- different EM-optimizers;
- back ends, which translate EM-code to assembler-code for a variety of real machines.

A compiler is then obtained by selecting a front end for a specific language, a back end for a specific machine and, optionally, one or more optimizers. Each compiler is a configuration, a certain combination of elements from the ACK system. The ACK system is an example of a product line, from an era before that notion was used.

The key tasks of configuration management are discussed in section 4.1. A Configuration Management Plan lays down the procedures that describe how to approach configuration management. The contents of this document are discussed in section 4.2. Configuration management is often supported by tools. The discussion of those tools is largely postponed until chapter 15.

4.1 Tasks and Responsibilities

Configuration management is concerned with the management of all artifacts produced in the course of a software development project. Though configuration management also plays a role during the operational phase of a system, when different combinations of components can be assembled into one system and new releases of a system are generated, the discussion below centers around the role of configuration management during system development.

We will for the moment assume that, at any point in time, there is one official version of the complete set of documents related to the project. This is called the **baseline**. A baseline is 'a specification or product that has been formally reviewed and agreed upon, that thereafter serves as the basis for further development, and that can be changed only through formal change control procedures' (IEEE610, 1990). Thus, the baseline is the shared project database, containing all approved items. The baseline may or may not be stored in a real database and supported by tools to assist in retrieving and updating its elements. The items contained in the baseline are the **configuration items**. A configuration item is 'an aggregation of hardware, software, or both, that is designated for configuration management and treated as a single entity in the configuration management process' (IEEE610, 1990). Possible configuration items are:

- source code components,
- the requirements specification,
- the design documentation,
- the test plan,
- test cases,
- test results,
- the user manual.

At some point in time, the baseline will contain a requirements specification. As time goes on, elements will be added: design documents, source code components, test reports, etc. A major task of configuration management is to maintain the integrity of this set of artifacts.

This is especially important if changes are to be incorporated. Suppose that, during testing, a major flaw in some component is discovered. We then have to retrace our steps and correct not only that component but also the corresponding design documents, and possibly even the requirements specification. This may affect work being done by other people still using the old version. Worse still, someone else may wish to make changes to the very same component at the same time.

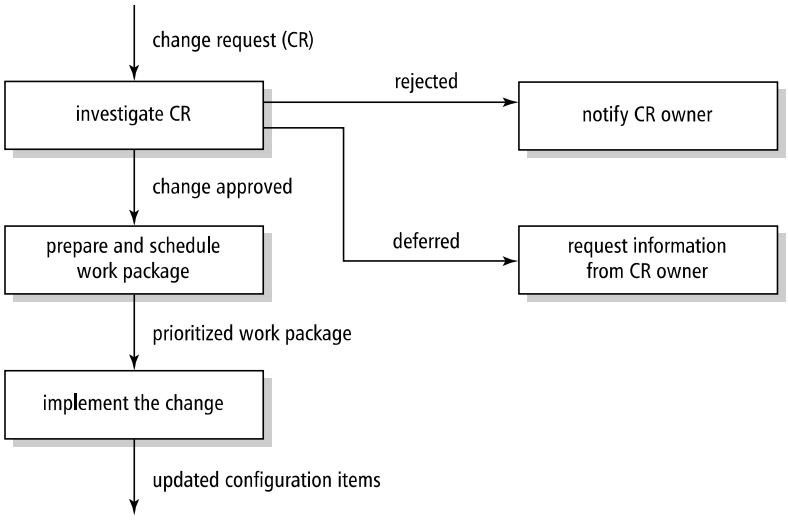

Figure 4.1 Workflow of a change request

Configuration management takes care of controlling the release and change of these items throughout the software life cycle.

The way to go about this is to have one shared library or database that contains all approved items, the so-called baseline. Adding an item to this database, or changing an item, is subject to a formal approval scheme. For larger projects, this is the responsibility of a separate body, the Configuration (or Change) Control Board (CCB). The CCB ensures that any change to the baseline is properly authorized and executed. The CCB is staffed with people from the various parties involved in the project, such as development, testing, and quality assurance.

Any proposed change to the baseline is called a change request. A change request may concern an error found in some component, a discrepancy found between a design document and its implementation, an enhancement caused by changed user requirements, etc. A change request is handled as follows (see also figure 4.1):

- The proposed change is submitted to the CCB. To be able to assess the proposed change, the CCB needs information as to how the change affects both the product and the development process. This includes information about the estimated amount of new or changed code, additional test requirements, the relationship to other changes, potential costs, complexity of the change, the severity of the defect (if it concerns one), resources needed, etc. Usually, a special change request form is provided to specify the information needed by the CCB.

- The CCB assesses the change request. The change request may be approved, rejected, or deferred if further information is required. If the request is approved, it eventually results in a work package which has to be scheduled.

- The CCB makes sure that all configuration items affected will eventually be updated accordingly. Configuration management provides a means to establish the status of all items and, thereby, of the whole project.

Thus, configuration management is not only about keeping track of all the different versions of elements of a system; it also encompasses workflow management tasks. The process depicted in the state transition diagram in figure 4.1, for example, describes what goes on in the life cycle of a change request. The process model thus defined exemplifies how the workflow of change requests can be managed.

In a similar vein, the state transition diagram in figure 4.2 shows the workflow of developer tasks during the development of a system component. It shows the possible states of a system component and the transitions in between. For example, after a component has been coded, it is unit tested. If bugs are found during unit testing, further coding is necessary. Otherwise, the component enters the review stage. If the review reveals problems, the coding stage is re-entered. Otherwise, the component is submitted to the CCB for formal approval. Finally, if unit testing does not reveal any errors, the review stage is skipped.

If components are kept under configuration control, configuration management can be used to manage the workflow of development tasks as well. Changes in the status of a component then trigger subsequent activities, as indicated in the development workflow model.

We have to take care that the workflow schemes do not unnecessarily curtail the day-to-day working of the people involved in the project. New items should not be added to the baseline until they have been thoroughly reviewed and tested. Items from the shared database may be used freely by the participants. If an item has to be changed, the person responsible for implementing the change gets a copy of that item and the item is temporarily locked, so that others can not simultaneously update the same item. The person implementing the change is free to tinker with the copy. After the change has been thoroughly tested, it is submitted back to the CCB. Once the CCB has approved it, the revised item is included in the database, the change itself is documented with the item, and the item is unlocked again. A sequence of documented changes thus provides a revision history of that item.

When an item is changed, the old version is kept as well. The old version still has to be used by others until they have adapted to the change. Also, we may wish to go back to the old version if another change is requested. We thus have different versions of one and the same item, and must be able to distinguish them. This can be done through some numbering scheme, where each new version gets identified by the next higher number. We then get, for a component X, versions X.0, X.1, X.2, and so on.

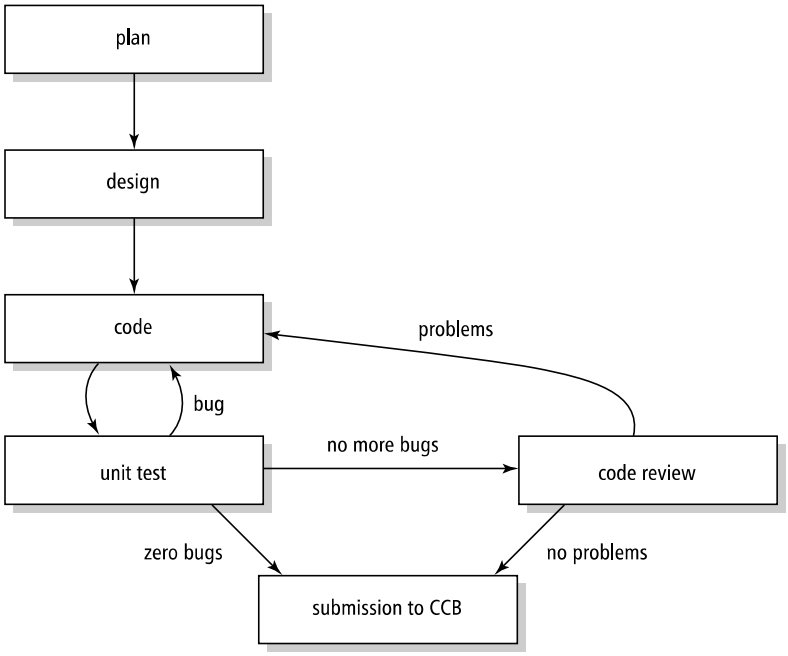

Figure 4.2 State transition diagram of development activities

In a more sophisticated environment, we may even create different branches of revisions. Figure 4.3 gives an example of such a forked development. In the example, component X.2.1. is, say, the result of fixing a bug in component X.2. Component X.3 may concern an enhancement to X.2. It should be noted that merging those parallel development paths again can be difficult. Also, the numbering schemes soon tend to become incomprehensible.

Configuration management is generally supported by powerful tools. Dart (1990) classifies the functionalities of these software configuration management (SCM) tools in eight categories:

- **Components**. SCM tools support storing, retrieving, and accessing components. Several versions of a component may be stored, baselines can be established, and branches of revisions may be created.

- **Structure**. SCM tools support support the representation and use of the structure of a system made up of components and their interfaces. In terms of architectural viewpoints, this is the implementation viewpoint; see section 11.3.

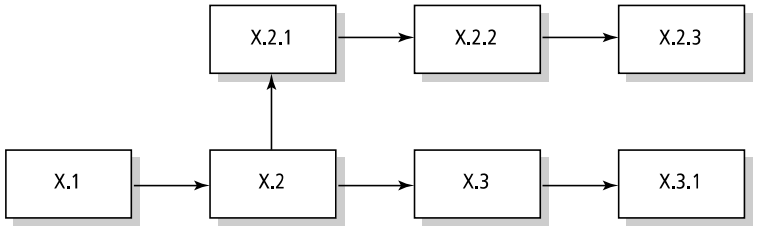

Figure 4.3 Parallel development paths

- **Construction**. SCM tools support the construction of an executable version of the system. By default, the latest version of all configuration elements is used, but it is also possible to regenerate older versions of the system.
- **Auditing**. SCM tools allow one to follow trails: which changes have been made to this component, who did those changes, and why. This way, a searchable archive of the system is maintained.
- **Accounting**. The searchable archive allows one to gather statistics about the system and the development process. We may for instance search for components that are changed very often, and hypothesize about the quality of those components.
- **Controlling**. SCM tools may be used for traceability purposes. If sufficient information is stored, we may trace defects to requirements, analyze the impact of changes, and the like.
- **Process**. SCM tools may support users in selecting tasks and performing those tasks in the appropriate context. For instance, the tool may assist in assigning the handling of a change request to a certain developer, and automatically provide her with a workspace with the components that need to be changed.
- **Team**. SCM tools may support collaboration, for example by generating a workspace for a group of collaborating developers, by noticing conflicts between developers, and the like.

Many SCM tools employ the **version-oriented** model of configurations. A physical change in a component then results in a new version, and different versions are thus characterized by their difference. Some tools use logical changes, rather than physical ones, as a basic unit of work in configuration management. This so-called **change-oriented** model gives a more intuitive way of working and may prevent a lot of user errors when configuring a system. Rather than identifying a configuration by

some arcane sequence of numbers, it is now identified by some baseline plus a set of changes. The set of changes may be empty. We thus specify

baseline X plus 'fix table size problem'

rather than

{X.3.1, Y.2.7, Z.1.4, . . . }.

As noted, SCM tools tools also offer help in constructing an executable version of the system. One writes a 'program' that identifies the various components of the required system and their mutual dependencies. The system in question is then generated by executing this 'program': the components are retrieved automatically from the database containing the source code components, and all components are translated and linked together into an executable system. If the system is smart enough, only those components that have been changed are translated anew.

Early SCM tools emphasized the product-oriented tasks of configuration management. They provide functionality to lock and unlock elements, provide for automatic numbering of revisions, and, by default, provide users with the latest version of an item. If an item is changed, they prompt the user and ask him to document the change. Present-day SCM tools increasingly provide the other functionalities as well, and have become a key ingredient of managing modern, distributed and global, software development. Tools for configuration and version management will be discussed more extensively in chapter 15.

On one hand, configuration management entails procedures on how to handle changes to and versions of documents. On the other hand, it consists of tool support to maintain the version history and ensure an up to date version of the system. In planning-driven development, both aspects are important. In agile projects, emphasis is on the tool support part. Agile projects favor continuous integration, whereby individual work of one developer is rapidly integrated with other parts of the system. And then tests are run to make sure everything still works as expected. Such a development cycle can take a few hours, and at most one day. At the end of the day, one again has a running system that passes all tests. This process is known as the *daily build*.

4.2 Configuration Management Plan

The procedures for configuration management are laid down in a document, the Configuration Management Plan. For the contents of this plan, we will follow the corresponding IEEE Standard (IEEE828, 1990). This document describes methods to identify configuration items, to control change requests, and to document the implementation of those change requests. A sample table of contents of the Configuration Management Plan is given in figure 4.4. The main constituents of this plan are:

Management This section describes how the project is being organized. Particular attention is paid to responsibilities which directly affect configuration management:

how are change requests being handled, how are development phases closed, how is the status of the system maintained, how are interfaces between components identified? Also, the relationship with other functional organizations, such as software development and quality assurance, is delineated.

Activities This section describes how a configuration will be identified and controlled and how its status will be accounted and reported. A configuration is identified by a baseline: a description of the constituents of that configuration. Such a configuration must be formally approved by the parties involved.

Clear and precise procedures are needed with respect to the processing of change requests if a software development project is to be controlled. A Configuration Control Board (CCB) usually has the responsibility to evaluate and approve or reject proposed changes. The authority, responsibility, and membership of the CCB have to be stated. Since software components are usually incorporated in a library, procedures for controlling this library have to be established as well.

In order to be able to control a software development project, data have to be collected and processed. Information that is normally required includes: the present status of components, versions and change requests, as well as reports of approved changes and their implementation.

Changes to configuration items may affect items outside the scope of the plan, such as hardware items. These external items have to be identified and their interfaces controlled. In a similar vein, interfaces to items developed outside the project have to be identified and controlled.

4.3 Summary

Configuration management is concerned with the management of all artifacts produced in the course of a software development project. It entails the following major activities:

- Configuration items must be identified and defined. A configuration item is a collection of elements that is treated as one unit for the purpose of configuration management. Examples of possible configuration items are the requirements specification, a software component, a test report, and the user documentation.
- The release and change of these items throughout the software life cycle must be controlled. This means that orderly procedures must be established as to whom is authorized to change or release configuration items.
- The status of configuration items and change requests must be recorded and reported. For example, the status of a change request may be: proposed, approved, rejected, or incorporated.

For larger projects, a Configuration Control Board is usually established. The CCB is responsible for evaluating all change requests and maintaining the integrity of

1. *Introduction*
   a. Purpose
   b. Scope
   c. Definitions and acronyms
   d. References
2. *SCM management*
   a. Organization
   b. SCM responsibilities
   c. Applicable policies, directives and procedures
3. *SCM activities*
   a. Configuration identification
   b. Configuration control
   c. Configuration status accounting
   d. Configuration audits and reviews
   e. Interface control
   f. Subcontractor/vendor control
4. *SCM schedules*
5. *SCM resources*
6. *SCM plan maintenance*

Figure 4.4 Sample structure of a software configuration management (SCM) plan (*Source:* IEEE Standard for Software Configuration Management Plans, *IEEE Std 828-1990. Reproduced by permission of IEEE.*)

the complete set of documents that relate to a project. Its tasks and the further procedures for configuration management are laid down in a separate document, the Configuration Management Plan.

The history and development of configuration management is closely tied to the history and development of configuration-management tools. In the early days, these tools emphasized the logging of physical file changes. There was little support for process aspects. Present-day configuration-management systems address process aspects as well (workflow management) and many have adopted a change-oriented next to or instead of a version-oriented view of configurations. More and more, configuration-management tools function as document-management tools in that they support cooperation among a group of people, possibly distributed over multiple sites, working together on a collection of shared objects.

4.4 Further Reading

A readable introduction to the topic of configuration management is given in (Babich, 1986). A more recent source is (Jonassen Hass, 2002). Estublier et al. (2005) gives an excellent overview of the major developments in the field. Weber (1996) and Wiborg-Weber (1997) describe the change-oriented configuration management technology.

Further references on technical aspects of configuration management are given in a later chapter, when tools for configuration and version control are discussed.

Exercises

1. What are the main tasks of configuration management?
2. Describe the role of the Configuration Control Board.
3. What is a configuration item?
4. What is a baseline?
5. Explain the difference between version-oriented and change-oriented configuration management.
6. Discuss the main contents of a configuration management plan.
7. ♡ Discuss differences and similarities between configuration management during development and maintenance.
8. ♡ Discuss possible differences between configuration management in a traditional waterfall development model and the evolutionary development models (see also (Bersoff and Davis, 1991).
9. ♡ Configuration management at the implementation level is often supported by tools. Can you think of ways in which such tools can also support the control of other artifacts (design documents, test reports, etc.)?
10. ♠ Devise a configuration management scheme for a small project (say, less than one person-year) and a large project (say, more than ten person-years). Give a rationale for the possible differences between those schemes.
11. ♠ To what extent could configuration-management tools support the gathering of quantitative project data? To what extent could such tools support project control?