Chapter 25 Notes

Systems can be thought of as a set of version, each of which has to be maintained and managed

Configuration Management (CM) is concerned with the policies, processes, and tolls for managing changing software systems.

Versions implement proposals for changes, corrections of faults, and adaptations for different hardware and operating systems.

The use of a CM system ensures that teams have access to information about a system that is under development and do not interfere with each other’s work

CM of a system involves 4 activities:

1. Change Management
   1. Keeping track of requests for changes to the SW from customers and developers, working out the costs and impact of making these changes, and deciding if and when the changes should be implemented.
2. Version Management
   1. Keeping track of the multiple versions of system components and ensuring that changes made to components by different developers do no interfere with each other.
3. System Building
   1. The process of assembling program components, data, and libraries, and then compiling and linking these to create an executable system.
4. Release Management
   1. Involves preparing SW for external release and keeping track of the system versions that have been released for customer use.

CM policies and processes define how to record and process proposed system changes, how to decide what system components to changes, how to manage different versions for the system and its components, and how to distribute changes to customers.

* When a new version of the SW has been implemented, it is handed over by the development team to the QA team. The QA team checks that the system quality is acceptable. If so, it becomes a controlled system, which means that all changes to the system have to be agreed on and recorded before they are implemented.

One of the problems of CM is that different companies use different terms to talk about the same topics.

Terms

* Configuration Item or Software Configuration Item (SCI)
  + Anything associated with a SW project that has been placed under configuration control.
* Configuration Control
  + Process of ensuring that version of systems and components are recorded and maintained so that changes are managed and all versions of components are identified and stored for the lifetime of the system
* Version
  + An instance of a configuration item that differs, in some way, from other instances of that item
* Baseline
  + Is a collection of component versions that make up a system. They are controlled, which means that the versions of the components making up the system cannot be changed. This means that it should always be possible to re-create a baseline from it constituent components.
* Codeline
  + Set of versions of a SW component and other configuration items on which that component depends
* Mainline
  + Sequence of baselines representing different versions of a system
* Release
  + A version of a system that has been released to customer (or other users in an organization) for use
* Workspace
  + A private work are where SW can be modified without affecting other developers who may be using or modifying that software
* Branching
  + Create of a new codeline from a version in an existing codeline. The new codeline and the existing codeline may then develop independently
* Merging
  + Creating of a new version of a SW component by merging separate versions in different codelines. These codelines may have been created by a previous branch of one of the codelines involved
* System Building
  + Creation of an executable system version by compiling and linking the appropriate versions of the components and libraries making up the system.

**Change Management (Section 25.1)**

* Change Management is intended to ensure that the evolution of the system is a managed process and that priority is given to the most urgent and cost-effective changes.
* The process is concerned with analyzing the costs and benefits of proposed changes, approving those change that are worthwhile, and tracking which components in the system have been changed.
  + The process is initiated when a “customer” completes and submits a change request describing the change required to the system. This could be a bug report, where the symptoms of the bug are described, or a request for additional functionality to be added to the system. These request may be submitted using a Change Request Form (CRF)
    - CRF
      * CRF is updated to reflect updates in its processing
      * Contains info on estimated costs of the change, the dates when the change was requested, approved, implemented, and validated. The CRF may also include a section where a developer outlines how the change may be implemented.
  + After a CR is submitted, it is check to ensure that it is valid. The checker may from a customer/application support team, or may be a member of the development team (for internal purposes).
    - Checking is necessary because not all changes requests require action. (It may report a bug that may of already been reported). If it is a valid CR, then it is logged as an outstanding request for subsequent analysis.
  + Next stage if Change Assessment and Costing
    - Handled by the development or maintenance team
    - The impact of the change on the rest of the system must be checked.
      * To do this you must identify all of the component affect by the change
      * If making changes mean that further changes elsewhere in the system are needed, this will obviously increase the cost of change implementation
    - Required changes to the system modules are assessed
    - Then, the cost of making the change is estimated, taking into account the cost of changing related components
  + Next stage
    - A separate group (Product Develop or CCB) decide it is cost-effective from a business perspective to make the change to the software. This group should review and approve all CRs, unless the change simply involves correcting minor errors on screen displays, webpages, or documents. These small requests should be passed to the development team without detailed analysis.
    - Then, the group considers the impact of the change from a strategic and organizational POV rather than a technical one. It decides whether the change in question is economically justified and prioritizes accepted changed for implementation. Accepted changes are passed back to development group and rejected CRs are closed. Significant factors that should be taken into account in deciding whether or not a change should be approved:
      * The consequences of not making the change
        + A more serious bug should have higher priority
      * Then benefits of the change
      * The number of users affected by the change
        + Lower # of users = lower priority
      * The costs of making the change
        + If too costly, CR can be rejected
      * The product release cycle
        + If a new version was just released, it makes sense to implement CR in the next version
* For software products, the customer is not directly involved in decisions about system evolution, so the relevance of the change to the customer’s business is not an issue. CRs for these products come from the customer support team, the company marketing team and the developers themselves.
  + Customer support may submit CRs associated with bugs that have been discovered and reported by customers after the SW has been released. The customer may use a webpage or e-mail to report bugs. A bug management team then checks that the bug reports are valid and translated them into formal system CRs.
  + Marketing staff may also generate CRs when they see competitor functionality and so can developers
* During development, when new version of the system are create through frequent system builds, a simpler change management process is normally used.
  + Problems and changes must still be recorded, but changes that only affect individual components and modules need not be independently assessed. They are passed directly to the system developer which accepts them or makes a case why they are not required.
* In Agile methods, customers are much more involved in the change management process such in assessing changes and assigned them priorities.
* As the development team changes SW components, they should maintain a record of changes made to each component in a Derivation History of a Component.
  + It should have info about the CR that triggered the change

**Version Management (Section 25.2.)**

* Version Management (VM) is the process of keeping track of different version of SW component or configuration items and the system in which these components are used. It also involves ensuring that changes made by different developers to these versions do not interfere with each other.
* Codeline = a sequence of version of source code with later versions in the sequence derived from earlier versions
* Baseline = a definition of a specific system. It specifies that component versions that are included in the system plus a specification of the libraries used, configuration files, etc.
  + These are important because you often have to re-create a specific version of a complete system.
* Mainline = a sequence of system version developed from an original baseline
* You need to use Version Control System or Source Code Control Systems.
  + These tools identify, store, and control access to different versions of components.
  + These tools offer the following features:
    - Version and release identification
      * Managed version are assigned identifiers when they are submitted to the system
    - Storage Management
      * To reduce the storage space required by multiple versions of component that differ only slightly, version management systems usually provide storage management facilities. Instead of keeping a complete copy of each version, the system stores a list of differences (deltas) between one version and another. By applying these to a source version, a target version can be re-created.
    - Change history recording
      * All of the changes made to the code of a system or component are recorded and listed. In some systems, these changes may be used to select a particular system version.
    - Independent Development
      * Different developers may be working on the same component at the same time. The version management keeps track of components that have been checked out for editing and ensures that changes made to a component by different developers do not interfere
    - Project Support
      * Support the development of several projects, which share components.
* To save space VM systems only save a list of differences (deltas) between the new version and the older version used to create that new version.
* VM system use the concept of a public repository and a private workspace. When changes are complete, they are checked in the component in the repository. If a component has been checked out, the VM system will normally warn other users wanting to check out that component that it has been checked out by someone else. The system will also ensure that when the modified component are checked in, the different versions are assigned different version identifiers and are separately stored. If conflicts exist between 2 merged versions, the developer has to check for the clashes and modify the changes so that they are compatible.

**System Building (Section 25.3)**

* Is the process of creating a complete, executable system by compiling and linking the system components, external libraries, configuration files, etc.
* The VM tools and System Building (SB) tools must communicate as the build process involves checking out component version from the repository managed by the VM system. The Configuration Description is used to identify a baseline to use.
* Building is complex and is error-prone as 3 systems may be involved:
  + Development System
    - Includes development tools like compilers, source code editors, etc.
    - Developers check out code from the Vm system into their private workspace before making changes. They may wish to build a version of the system for testing before committing changes. This involves local buuld tools that use checked-out versions of the components in the private workspace
  + Build Server
    - Used to build definitive, executable versions of the system.
    - Interacts closely with the VM system.
    - The system build may rely of external libraries that are not included in the VM system
  + Target Environment
    - Is the platform on which the system executes
* Both the Development System and the Build Server may both interact with the VM system
* SB involves assembling a large amount of info about the SW and its operating environment. Therefore, for anything apart from very small systems, it always makes sense to use an automated build tool to create a system build. You don’t need the source code files that are involved in the build. You may have to link these with externally provided libraries, data files (such as a file of error messages), and configuration files that define the target installation. You also have to define the versions of the compiler and other software tools that are to be used in the build.
* A Build System may provide the following features:
  + Build Script Generation
    - The build system should analyze the program that is being built, identify components, and automatically generate a build script.
  + VM System Integration
    - Should check out the required versions of components from the version management system
  + Minimal Recompilation
    - Should work out what source code needs ot be recompiled and set up compilations if required
  + Executable System Creation
    - Should link the compiled object code files with each other and with other required files, such as libraries and configuration files, to create an executable system
  + Test Automation
    - Automatically run automated tests using test automation like JUnit. Thes check that the build has not been broken by changes
  + Reporting
    - Should provide reports about the success or failure of the build and the tests that have been run
  + Documentation Generation
    - Build System may be able to generate release notes about the build and system help pages
* To decide whether a component needs to be compiled again, its signature is used. The signature identifies each source code version and is changed when the source code is edited. By comparing signatures on the source and object files, it is possible to decide if the source code component was used to generate the object code component.
  + 2 Types of Signature that may be used
    - Modification Timestamps
      * Time and date when that file was modified
      * Recompilation is required if the timestamp of the source code is later than the timestamp of the object file
    - Source Code Checksums
      * A checksum function calculates a unique number using the source text as input.
      * If there is no object file with the same signature as the source code file to be included in the system, then recompilation is necessary
      * This approach allows many different versions of the object code of a component to be maintained at the same time
* Agile Methods recommend very frequent builds with automated testing.
  + Steps in Continuous Integration
    - Check out the mainline system from the VM systems into private workspace
    - Build the system and run automated tests. If test do not pass, the build is broken and you should inform whoever check in the last baseline system so that they fix it
    - Make the changes to the system components
    - Build the system in the private workspace and return system tests. IF test fail, continue editing
    - Once the system has passed its tests, check it into the Build System but do not commit it as a new system baseline
    - Build the system on the build server and run the tests. You need to do this in case other have modified components since you checked out the system. If this is the case, check out the components that have failed and edit these so that tests pass on your private workspace.
    - If the system passes its tests on the build system, then commit the changes as a new baseline in the system mainline.
  + Argument for Continuous Integration
    - It allows problems caused by the interactions between different developers to be discovered and repaired ASAP
    - But continuous integration is not always possible to implement because
      * If the system is too large, it may take a long time to build and test
      * If the development platform is different from the target platform, it may not be possible to run system tests in the developer’s private workspace. There may be differences in hardware, OS, or installed SW. Therefore more time is required for testing the system
  + For large system, daily build system may be used
    - Development organization set a delivery time for the system component
    - A new ersion of the system is built from these components by compiling and linking them to form a complete system
    - This system is then delivered to the testing team, which carries out a set of predefine system tests. Meanwhile, developers work on their components, adding to the functionality and repairing faults discovered in previous tests
    - Faults that are discovered during system testing are documented and returned to the system developers
  + Advantages of Frequent Builds
    - Changes of finding problems stemming from component interactions early in the process are increased
    - It encourages thorough unit testing of components
    - It puts pressure on developers to not break the build

**Release management (Section 25.4)**

* A System Release is a version of a SW system that is distributed to customers.
* For customer development, this can be complex, because each customer may have special releases built for them. Also, many releases have to be managed at once.
* When a system release is produced, it must be documented to ensure that it can be re-created exactly in the future.
  + To document a release, you have to record the specific version of the source code components that were used to create the executable code. You must keep copies of the source code files, corresponding executables, and all data and configuration files. Also record the versions of the OS, libraries, compilers, and other tools used to build the software.
* Factors influencing system Release Planning
  + Technical Quality of the System
    - If serious system faults are reported which affect the way in which many customers use the system, it may be necessary to issue a fault repair release. Minor system faults may be repaired by issuing patches that can be applied to the current release of the system
  + Platform Changes
    - You may have to create a new release of an application when a new version of the OS platform is released
  + Lehmand’s Fifth Law
    - Suggests that if you add a lot of new functionality to a system; you will also introduce bugs that will limit the amount of functionality that may be included in the next release. -> A large release will have to be followed by a release that focuses on repairing problems and improving performance
  + Competition
    - New system release may be necessary because a competing product has introduced new features
  + Marketing Requirements
    - Marketing department of an organization may have made a commitment for releases to be available at a particular date
  + Customer Change Proposals
    - Customers may have made and paid for a specific set of system change proposals, and they expect a system release as soon as these have been implemented.
* A release may include
  + Configuration files
    - Define how the release should be configured for particular installations
  + Data Files
    - Such as files of error messages
  + An Installation Program
  + Electronic and Paper Documentation
  + Packaging and Associated Publicity that have been designed for that release
* Release Creation = the process of creating the collection of files and documentation that includes all of the components of the system release. The executable code of the programs and all associated data files must be identified in the VM system and tagged with the release identifier. Configuration descriptions may have to be written for different hardware and OS and instructions prepared for customers who need to configure their own systems. Scripts for the installation program may have to be written. Finally, when all info is available, an executable master image of the SW must be prepared and handed over for distribution to customers of sales outlets.
* You cannot assume that users will move on to the newest release of a system. New releases, cannot rely on the installation of previous releases.
* Release costs are so high that typically new releases or only created for new platforms or for significant added functionality.
* Problems with Downloadable Patches
  + Customers may never find them and may not understand why they should be installed
* Most companies have automatic updating where systems are updated whenever a new minor release becomes available.