Configuration management

- Managing the products of system change

Objectives

- To explain the importance of software configuration management (CM)
- To describe CM planning
- To describe key CM activities namely change management, version management and system building
- To discuss the use of some CM tools

Topics covered

- Configuration management planning
- Change management
- Version and release management
- System building

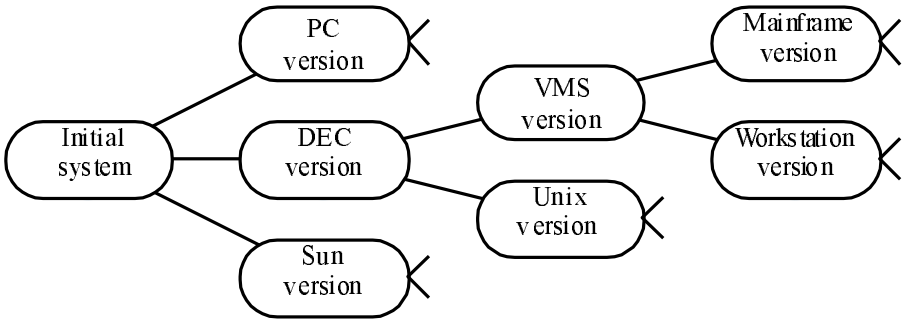
Configuration management

- New versions of software systems are created as they evolve
  - For different machines/OS
  - Offering different functionality
  - Tailored for particular user requirements
- CM is concerned with managing evolving software systems
  - System change is a team activity
  - CM aims to control the costs and effort involved in making changes to a system

Configuration management

- Involves the development and application of procedures and standards to manage an evolving software product
- May be seen as part of a more general quality management process
- When released to CM, software systems are sometimes called *baselines* as they are a starting point for further development

System families



CM standards

- CM should always be based on a set of standards which are applied within an organisation
- Standards should define how items are identified, how changes are controlled and how new versions are managed
- Standards may be based on external CM standards (e.g. IEEE standard for CM)
- Existing standards are based on a waterfall process model - new standards are needed for evolutionary development

Configuration management planning

- All products of the software process may have to be managed
  - Specifications
  - Designs
  - Programs
  - Test data
  - User manuals
- Thousands of separate documents are generated for a large software system

CM planning

- Starts during the early phases of the project
- Must define the documents or document classes which are to be managed (Formal documents)
- Documents which might be required for future system maintenance should be identified and specified as managed documents

The CM plan

- Defines the types of documents to be managed and a document naming scheme
- Defines who takes responsibility for the CM procedures and creation of baselines
- Defines policies for change control and version management
- Defines the CM records which must be maintained

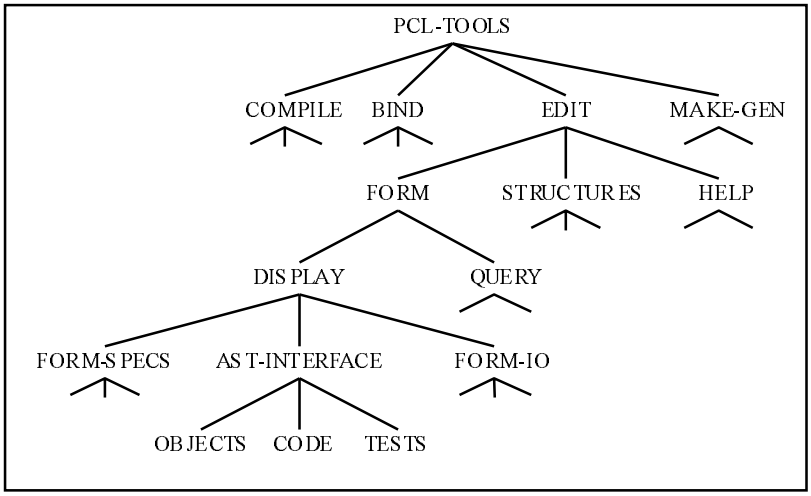
The CM plan

- Describes the tools which should be used to assist the CM process and any limitations on their use
- Defines the process of tool use
- Defines the CM database used to record configuration information
- May include information such as the CM of external software, process auditing, etc.

Configuration item identification

- Large projects typically produce thousands of documents which must be uniquely identified
- Some of these documents must be maintained for the lifetime of the software
- Document naming scheme should be defined so that related documents have related names.
- A hierarchical scheme with multi-level names is probably the most flexible approach

Configuration hierarchy



CM planning database

- All CM information should be maintained in a CM database
- Should allow queries about configurations to be answered
  - Who has a particular system version?
  - What platform is required for a particular version?
  - What versions are affected by a change to component X?
  - How many reported faults in version T?
- CM database should preferably be linked to the software being managed

CM database implementation

- May be part of an integrated environment to support software development. The CM database and the managed documents are all maintained on the same system
- CASE tools may be integrated with this so that there is a close relationship between the CASE tools and the CM tools
- More commonly, the CM database is maintained separately as this is cheaper and more flexible

Change management

- Software systems are subject to continual change requests
  - From users
  - From developers
  - From market forces
- Change management is concerned with keeping managing of these changes and ensuring that they are implemented in the most cost-effective way

The change management process

```
Request change by completing a change request form
Analyze change request
if change is valid then
    Assess how change might be implemented
    Assess change cost
    Submit request to change control board
    if change is accepted then
        repeat
            make changes to software
            submit changed software for quality approval
        until software quality is adequate
    create new system version
else
    reject change request
else
    reject change request
```


Change request form

- Definition of change request form is part of the CM planning process
- Records change required, suggestor of change, reason why change was suggested and urgency of change(from requestor of the change)
- Records change evaluation, impact analysis, change cost and recommendations (System maintenance staff)

Change request form

- Replace with portrait slide

Change tracking tools

- A major problem in change management is tracking change status
- Change tracking tools keep track the status of each change request and automatically ensure that change requests are sent to the right people at the right time.
- Integrated with E-mail systems allowing electronic change request distribution

Change control board

- Changes should be reviewed by an external group who decide whether or not they are cost-effective from a strategic and organizational viewpoint rather than a technical viewpoint
- Should be independent of project responsible for system. The group is sometimes called a change control board
- May include representatives from client and contractor staff

Derivation history

- Record of changes applied to a document or code component
- Should record, in outline, the change made, the rationale for the change, who made the change and when it was implemented
- May be included as a comment in code. If a standard prologue style is used for the derivation history, tools can process this automatically

Example - derivation history

```
// PROTEUS project (ESPRIT 6087)
//
// PCL-TOOLS/EDIT/FORMS/DISPLAY/AST-INTERFACE
//
// Object: PCL-Tool-Desc
// Author: G. Dean
// Creation date: 10th November 1994
//
// © Lancaster University 1994
//
// Modification history
// Version     Modifier    Date      Change        Reason
// 1.0         J. Jones    1/12/94   Add header    Submitted to CM
// 1.1         G. Dean     9/4/95    New field     Change
//                                                 req. 07/95
```


Version and release management

- Invent identification scheme for system versions
- Plan when new system version is to be produced
- Ensure that version management procedures and tools are properly applied
- Plan and distribute new system releases

Versions/variants/releases

- *Version* An instance of a system which is functionally distinct in some way from other system instances
- *Variant* An instance of a system which is functionally identical but non-functionally distinct from other instances of a system
- *Release* An instance of a system which is distributed to users outside of the development team

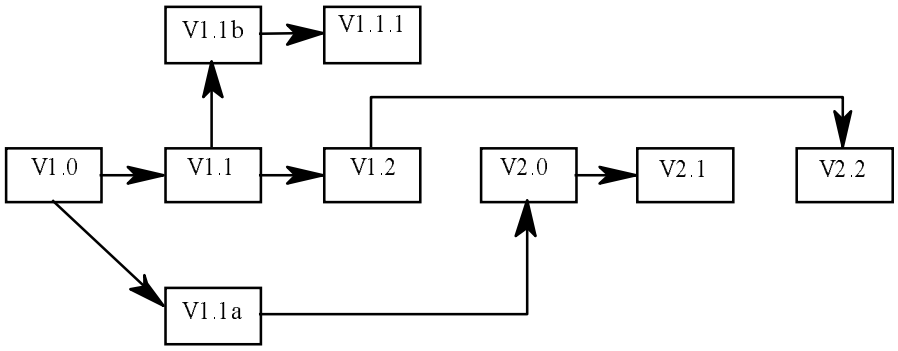
System releases

- Not just a set of executable programs
- May also include
  - Configuration files defining how the release is configured for a particular installation
  - data files needed for system operation
  - an installation program or shell script to install the system on target hardware
  - electronic and paper documentation
- Systems may be released on magnetic tape, floppy disk or CD-ROM

Version identification

- Simple naming scheme uses a linear derivation e.g. V1, V1.1, V1.2, V2.1, V2.2 etc.
- Actual derivation structure is a tree or a network rather than a sequence
- Names are not meaningful.
- Hierarchical naming scheme may be better

Version derivation structure



Attributed version identification

- Attributes can be associated with a version with the combination of attributes identifying that version
- Examples of attributes are Date, Creator, Programming Language, Customer, Status etc.
- More flexible than an explicit naming scheme for version retrieval; Can cause problems with uniqueness
- Needs an associated name for easy reference

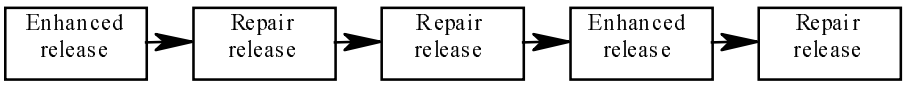
Release management

- Releases must incorporate changes forced on the system by errors discovered by users and by hardware changes
- They must also incorporate new system functionality
- Release planning is concerned with when to issue a system version as a release

Lehman's fifth law

- The incremental system change which can be incorporated in each release of the system is approximately constant
- If too many new features are included at the same time as error repairs, the cost of producing a new release is significantly increased
- If a release has many changes incorporated, it must be followed by a further release fixing problems in the first release

System release strategy



Release problems

- Customer may not want a new release of the system
  - They may be happy with their current system as the new version may provide unwanted functionality (e.g. Word 6)
- Release management must not assume that all previous releases have been accepted. All files required for a release should be re-created when a new release is installed

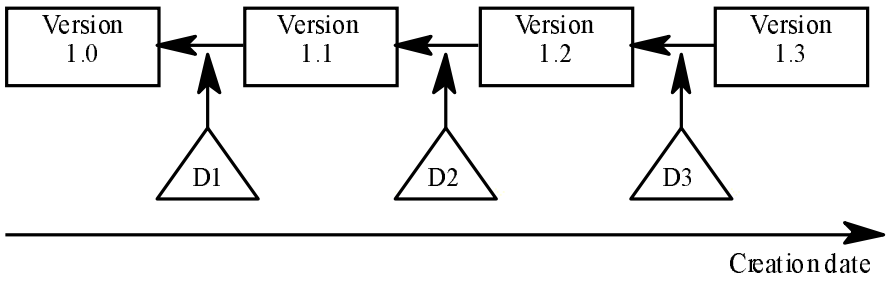
Version management tools

- Version and release identification
  - Systems assign identifiers automatically when a new version is submitted to the system
- Controlled change.
  - Only one version at a time may be checked out for change. Parallel working on different versions
- Storage management.
  - System stores the differences between versions rather than all the version code
- Change history recording
  - Record reasons for version creation

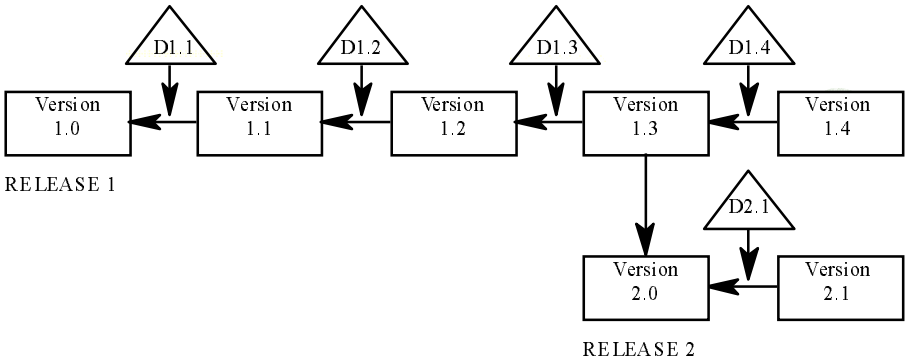
RCS - Revision Control System

- RCS is a relatively old tool but still widely used.
- Minimizes the disk requirements by only storing differences (deltas) from a base version
- Applies deltas to the latest release to re-create earlier system versions
- Allows any named version or release to be generated
- Allows independent development of different releases

Deltas in RCS



Parallel development in RCS



RCS limitations

- Designed as a code control system therefore intended for use with ASCII text
- Cannot be used to manage object code or other documents with non-ASCII representations (e.g. multimedia files)
- Text-based user interface. Version browsing is difficult
- Version retrieval based on the name rather than the version attributes

System building

- Involves taking all system components and combining them into a single executable system
- Different systems are built from different component combinations
- May take several days for large systems if all components are compiled and linked at the same time

System building problems

- Do the build instructions include all required components?
  - When there are many hundreds of components making up a system, it is easy to miss one out. This should normally be detected by the linker
- Is the appropriate component version specified?
  - A more significant problem. A system built with the wrong version may work initially but fail after delivery
- Are all data files available?
  - The build should not rely on 'standard' data files. Standards vary from place to place

System building problems

- Are data file references within components correct?
  - Embedding absolute names in code almost always causes problems as naming conventions differ from place to place
- Is the system being built for the right platform
  - Sometimes must build for a specific OS version or hardware configuration
- Is the right version of the compiler and other software tools specified?
  - Different compiler versions may actually generate different code and the compiled component will exhibit different behaviour

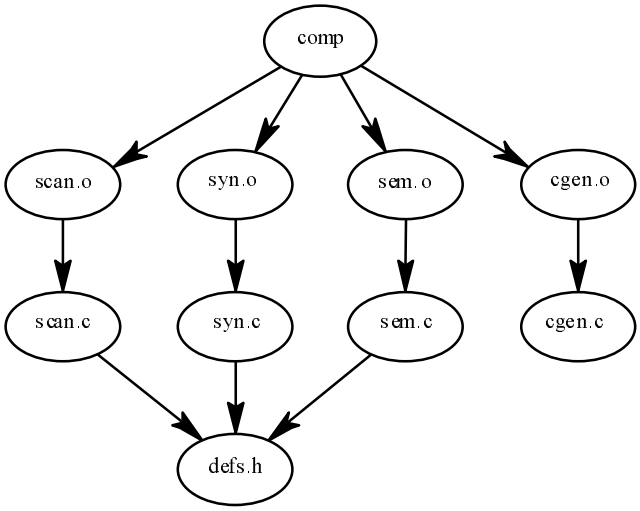
System representation

- Systems are normally represented for building by specifying the file name to be processed by building tools. Dependencies between these are described to the building tools
- Mistakes can be made as users lose track of which objects are stored in which files
- A system modelling language addresses this problem by using a logical rather than a physical system representation

System building with Make

- Most widely used build tool on the Unix system is MAKE. Comparable tools are available on other systems
- User specifies component dependencies and MAKE automatically forces recompilation of required files when it detects that the source code has been changed after the object code was created

Component dependencies



Make problems

- Based on a physical rather than a logical model of dependencies
- Dependency specifications (Makefiles) quickly become large, complex, hard to understand and expensive to maintain
- MAKE uses a simple model of change based on file update times. Source code changes NEED not require re-compilation

Make problems

- MAKE does not (easily) allow versions of tools such as the compiler to be specified
- Not tightly linked to version management tools such as RCS. Manual intervention is usually needed to check out source code from a version management system for building

System modelling in PCL

- PCL is a system modelling language
- Systems are represented as families with stable and variable parts
- Attribute-based component identification
- Component interface definition
- Composition structure of components
- Physical structure - mapping of logical component to physical files
- Component relationships e.g. 'requires' and 'implemented as'

Logical system model

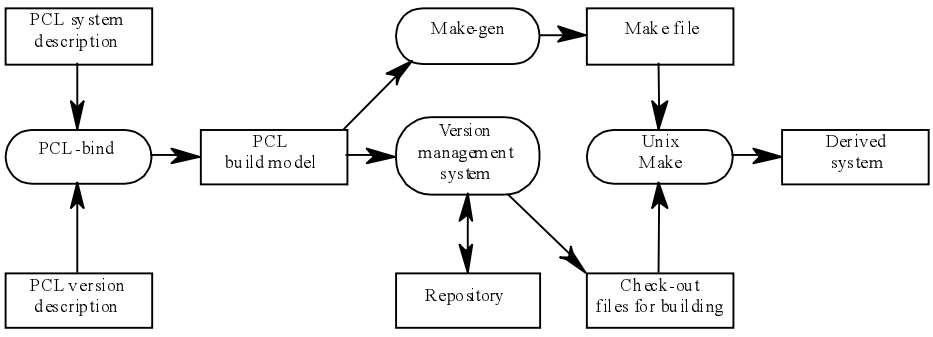
```
family print-server

attributes
        multiple-paper-types: boolean ;
end
interface
        print => print-file ;
        display-queue => show-print-queue ;
        dequeue => delete-print-job ;
        select-printer => set-printer ;
        select-paper-type =>
            if multiple-paper-types then
                set-input-tray endif ;
end
```

Logical system model

```
parts
        PRINT => printer-controller ;
        QUEUE => queue-manager ;
        select-paper-type =>
            if multiple-paper-types then
                set-input-tray endif ;
end
physical
    source => "print_server" repository "/usr/src" ;
    EXECUTABLE => Print-man binary
                        "/usr/utils/bin" ;
end
end // print-server
```


PCL system building



System building from PCL

- Variability is removed from the PCL description by PCL-bind
- The system model is analysed and dependencies are identified. A make file is generated
- The required versions of components are identified by their attributes and checked out automatically from RCS
- Make is called to build the system

Parallel system building

- DSEE is an example of a system which supports networked building
  - Parallel building of different system versions
  - Parallel compilation on different network nodes
  - Integrated source code control and system building
  - Configuration identification based on a system model called a configuration thread
  - Derived object management where all built objects are maintained until they are needed or space is exhausted

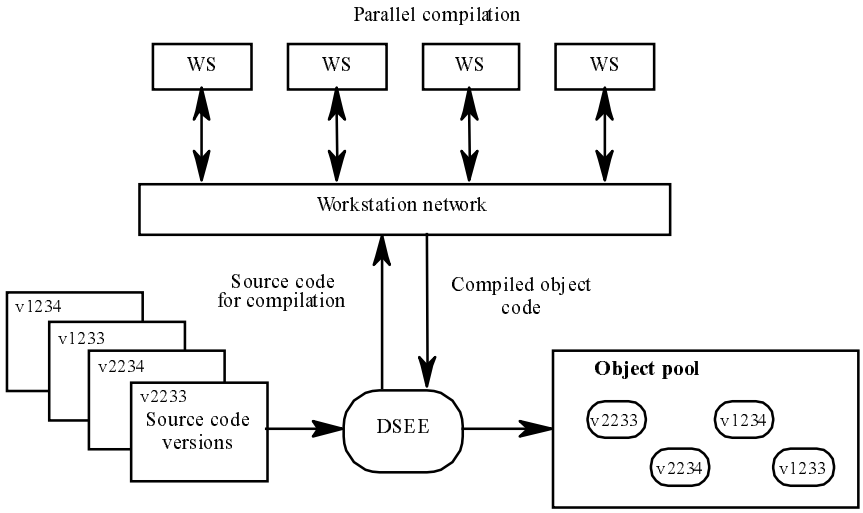
DSEE characteristics

- No automatic recompilation of object code after a source code change (unlike Make)
- Only recompile a component when a demand is made for the object code
- Supports attribute-based version identification through 'configuration threads'
- Object code related to source code through attributes. Compiled objects have the same attribute set as their sources

Parallel building

- Generally, on a network of workstations, some are idle at any one time
- DSEE finds an idle machine and builds component on that machine
- Leads to performance improvement of several hundred %
- System works well when a network is composed of common platforms so components are interchangable.

Network-oriented building



Configuration threads

- Identify the version to be built by specifying its attributes (e.g. Build the version with identifier attribute, R3; build the version with status attribute 'beta test' etc.)
- Identify the version of the compiler and support tools along with its parameters which is to be used in building a system version

Integrated CM databases

- Integrate information about the configuration with the entities which are themselves being managed.
- Generally based on ERA semantic model where entities participate in relationships and have associated attributes.
- Include process information e.g.. an entity attribute may have a value which is a process description showing how that entity was created.

CM support with an integrated DB

- CM tools may be implemented using database query and browsing facilities
- Using process information, other tools such as editors/compilers etc. can be integrated and can automate version creation
- Information dissemination re versions and changes can be incorporated in the process information

Database-oriented CM tools

- Rely on a powerful semantic or object-oriented database
- Store both process and product information
- Provide an integrated set of tools for configuration management
- Integrated with other tools through the database
- Graphical user interface
- Still research prototypes

Key points

- CM is the management of system change to software products
- Effective CM is essential in large software projects
- CM activities include CM planning, change management, system building, and version and release management
- A formal document naming scheme should be established and documents should be managed in a database

Key points

- System releases should be phased with releases fixing problems interleaved with releases offering new functionality
- System building involves assembling components into a system. It is always supported by system building tools such as Make
- Integrated CM tools such as DSEE combine support for system building and version management