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

- Initial system
  - PC version
  - DEC version
  - Sun version
  - Mainframe version
  - VMS version
  - Workstation version
  - Unix version
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

- PCL-TOOLS
  - COMPILE
  - BIND
  - EDIT
  - MAKE-GEN
  - FORM
  - STRUCTURES
  - HELP
    - DISPLAY
      - FORM-SCPECS
      - AST-INTERFACE
      - OBJECTS
        - CODE
        - TESTS
      - QUERY
      - FORM-IO
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.
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
    Assess how change might be implemented
    Assess change cost
    Submit request to change control board
if change is accepted
    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 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 OS.
  - Electronic and/or paper documentation.
- Systems may be released on magnetic tape, floppy disk, CD-ROM, or via the WWW.
Version identification

- Simple naming scheme uses a linear derivation e.g., V1, V1.1, V1.2, V2.1, V2.2 ...
- 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

Diagram:

- V1.0
- V1.1
- V1.2
- V2.0
- V2.1
- V2.2
- V1.1a
- V1.1b
- V1.1.1
- V1.1.2
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

Enhanced release → Repair release → Repair release → Enhanced release → Repair release
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.
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.

- **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

Version 1.0

Version 1.1

Version 1.2

Version 1.3

D1

D2

D3

Creation date
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).
- Simple text-based user interface.
- 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.
  • Is the appropriate component version specified?
  • A system built with the wrong version may work initially but fail after delivery.

◆ Are all data files available?
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 behavior.
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 re-compilation of required files when it detects that the source code has been changed after the object code was created.
Component dependencies

- comp
- scan.o
- scan.c
- defs.h
- syn.o
- syn.c
- sem.o
- sem.c
- cgen.o
- cgen.c
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.
- However, some 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.
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.