Configuration Management

Controlling change & product integrity

What is Configuration Management?

- Managing projects w.r.t. issues such as
  - multiple developers working on the same code
  - targetting multiple platforms
  - supporting multiple versions
  - controlling the status of code
- Process aspect
  - e.g., authorizing and controlling change
- Tools aspect
  - e.g.., tracking and storing version histories

Example 1: Interface Control

Typical software engineering class project

- Scenario:
  - Small team development, tight deadline
  - First incremental build nearing completion
  - Developers note desireable changes that will affect module interfaces
    - Changing now could affect schedule; changing later will cost more
- Issues:
  - How do we decide which changes to make?
  - How do we coordinate the change?
    - esp. for other developers that depend on the current interfaces

Example 2: "Evergreen System"

Based loosely on Northern Telecom digital switch

- Software system
  - Digital telephone switch, millions of lines of code
  - Always three system releases:
    - Current customer release
    - In development (detail design, code, test)
    - In development (requirements and arch. design)
- Scenario
  - Bug found in current customer release
    - requires modification to maintain reliable function
    - do the development versions require revision? If so, how should they be coordinated?

Example 3: Distributed Development

Based loosely on Arcadia research project experience

- Multiple development organizations (A, B, C), geographically distributed
- Scenario: Dependencies
  - Testing tools developed by A,B depend on API of compiler developed by C
  - Compiler depends on object manager developed by A
  - Compiler and object management system are also distributed independently of higher-level packages
  - New version of compiler planned, with revised API, will take approx. 1 year to deliver initial version
- Issues:
  - When should organizations A and B revise their testing tools to use the new interface?
  - How should the test tools be packaged and distributed? Should they include the compiler and object manager packages?

Managing Change: Process

- Establish a baseline
  - Change cannot be managed unless there is a "baseline" configuration. Changes to the baseline require authorization.
  - Scope and authorization procedure depends on stage
    - During implementation cycle, authorization may be required only for changes to design (interfaces) and schedule
    - After release, any change may require a specified regression test and distribution procedure
- Establish authority and procedures

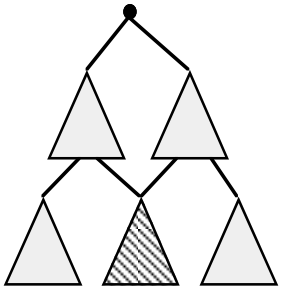
Example change processes

- Ex. 1, small team: proposed interface changes at weekly meeting
  - negotiate and plan considering schedule impact
- Ex. 3, distributed development: "configuration control board" with representatives from A,B,C negotiates and schedules change
  - Establishes 2 baselines: Development and Demo
  - Demo "frozen" 3 months before demo schedule
    - Snapshot of working system saved
    - Only critical fixes allowed; no "improvements"
  - Board schedules "changeover" periods for development baseline

What is in a Configuration?

- Delivered system
  - The "program" or "system" (executables, etc.)
  - User documentation, requirements documents, ...
- Other work products
  - Design, requirements, meeting notes, etc.
  - Source code, including build scripts
  - Test cases, drivers, ...
  - *Complete change history*
- "External" dependencies
  - Tools: compilers, CASE tools, word processors ...
  - Libraries, system configurations, other "environment"

Configurations may overlap



- Overlapping platform builds
  - Example: Windows (3.1, 95, NT), Mac (68k, PPC), and Unix (HP, Sun Sparc, x86 (Linux, Solaris X86))
- Overlapping product features
  - Microsoft Office (Standard, Pro), MS Word
  - FrameMaker, FrameMaker+SGML
- Overlapping revisions
  - Delivered, beta, internal development

Basic Tool Support

- Concurrency control:
  - Prevent (or support) simultaneous changes to the same configuration elements
- Versioning:
  - Maintain a complete log of changes, and recreate any previous version on demand
- Building:
  - Construct derived artifacts (executables, documents, etc.) from a consistent configuration

These are just the basics; higher-level and process-oriented tools can be built on this foundation

Concurrency Control Approaches

- Locking control:
  - Developer "checks out" an element
    - Read only: without a lock
    - Read/Write: with a lock
  - Changes are committed at "check in"
    - and become "current" version for next check-out
- Merging control:
  - Multiple developers may "check out" a version
    - Both may be making changes to local copies
  - Changes "merged" at check-in
    - Overlapping changes may require resolution

RCS: a Revision Control System

Originally by W. Tichy, then at Purdue; still the core of many commercial and free configuration management systems.

- Locking change control
  - Developers may "break" a lock and send email
- Revision tree
  - Revisions other than the "current" may be checked out and changed
  - Revisions may be "merged" into the trunk
- Efficient (text) storage through deltas
  - Complete copy of newest version, deltas (edit commands) for all previous versions. Any version can be recreated on demand applying deltas.
- Other functions
  - Manage version log in source file comments
  - Inquire about status of any library component
  - Compare any two versions, or compare local copy to historical version

Using RCS: Example (next slides)

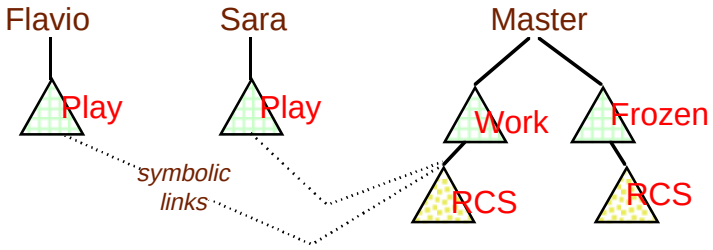
- A revision control process suitable for a small team, with a single shared file system, using Unix
  - Can be adapted to Windows or Mac (e.g, using Visual Source Safe and/or Metrwerks Code Manager)
- Not suitable for larger or smaller projects
  - I use RCS in a simpler way for single-person developments, including papers
  - Larger projects require more elaborate tool support and, especially, more elaborate procedures

A simple revision control process

- Typically three "builds" are current:
  - Frozen: The "demo" version (shared)
  - Work: The current integrated version (shared)
  - Play: Individual developer's version
- Steps:
  - Programmer checks out module to "play", makes changes and tests against "work" modules of others
  - Programmer checks in module when it has been tested against the "work" version (this may require coordination)
  - On a regular schedule, "Work" version is tested and moved to "Frozen" version

Directory Structure for Version Management

- Using RCS or similar for revision and concurrency control (locking) in a shared file system
  - Have a policy on holding locks: e.g., 24 hours or less
  - May need multiple RCS directories, or a protocol for indicating the components of "work" vs "frozen" versions



Distinguish "derived" from "source"

- All "ultimate source" should be under version/ revision control
- All "derived" objects should be produced automatically (e.g., when you run "Make")
  - Never edit derived objects
    - Examples: Object code (obvious?), lex output, generated web pages
- When generating components, consider revision procedure
  - If post-generation changes are necessary, they should be saved and applied to revised version

Version Building

- The basics: Make (or a Make-oid)
  - Generate all derived objects in a consistent configuration
  - May interact with revision control, e.g.,
    - The Makefile is versioned under RCS control
    - The Makefile checks all other components out from RCS
- Beyond Make ...
  - mkMake, and other Makefile generators
    - Analyze source code for dependencies
  - config, autoconfig, ...
    - Adapting to an environment

Configuration Policy Support

- RCS, Make, and related tools provide basic mechanism, but do not enforce policies
- Higher level process can be built on them
  - Example: Work-flow enforcement including successful regression test and management approval before accepting changed version
  - Sometimes integrated with problem report tracking system

Information resources

- Usenet news: comp.software.config-mgmt
- Web sites to start at:
  //www.iac.honeywell.com/Pub/Tech/CM/CMFAQ.html
  //www.iac.honeywell.com/Pub/Tech/CM/CMTools.html
- Books, tools: See the web sites for references
  - Many tools are available, ranging from free (RCS) to very expensive
- Current research: *Workshop on Configuration Management*, before *ICSE'97* (Boston, April 97)

Summary: Configuration Control

- Important for producing and maintaining quality software, on schedule
  - Not just code: Reports, web sites, ...
  - Can be simple or complex, depending on the system and organization
  - A "key capability" in SEI Capability Maturity Model
- Management (process) aspects, and technical aspects
  - Neither management alone, nor tools alone, are enough
  - Management policy and procedures must be supported by technical capability to manage configurations