Configuration Management

Dr. U. Aßmann
Research Center for Integrational Software Engineering

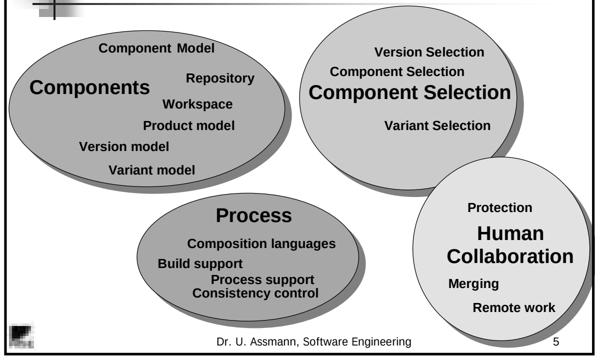
Content

Basic Concepts

Goal

- Constructing software from components and frameworks
- Have several products, but a large reuse
- Have several versions
  - A new version every 2 years
- Have several variants
  - Portability
- Manage work concurrently “programming-in-the-many”

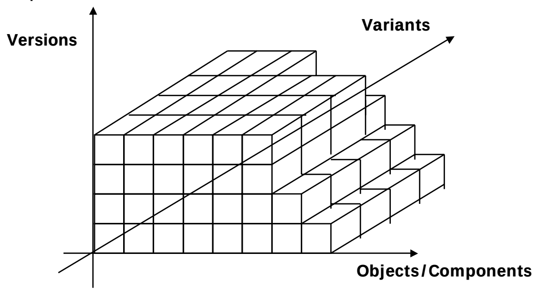
Software Configuration



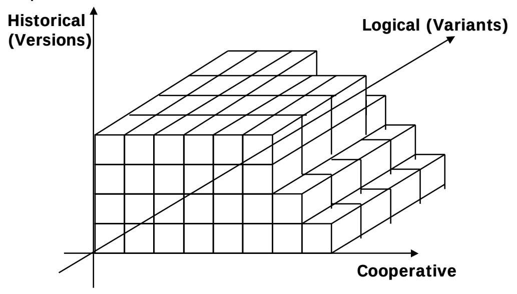
History

- SCCS (Rochkind) 1985
  - Files in versions
  - Deltas (diffs) between versions
  - Locking for collaborative work
- Make (Feldman) 1985
- RCS (Tichy) 1987
- Shape (Mahler) 1992
- Feature Logic (Zeller, Snelting) 1995

The Version/Variant Universe

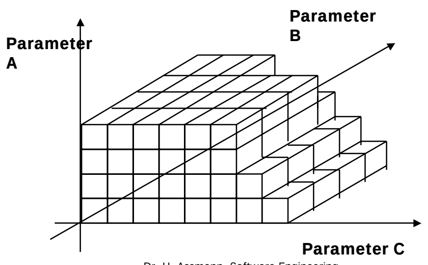


Alternative View



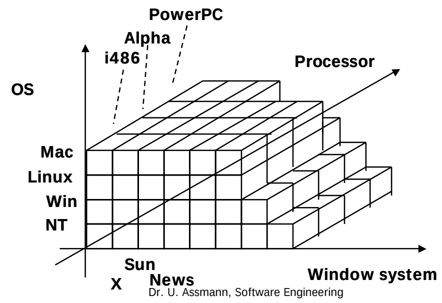
The Variant Parameter Universe

- A n-dimensional space (with n parameters of k variants)



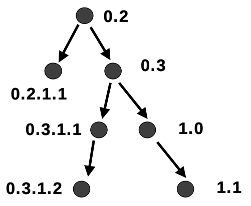
The Variant Parameter Universe

- A n-dimensional space (with n parameters of k variants)



The Version Universe

- A branching tree of unknown depth
- Every new modification either generates a new child
- Or a branch



Components in Repositories and Workspaces

Components

- Component model
  - Files (main case, unfortunately)
  - Objects
  - Documents
  - Often untyped!
- Version model
  - Branching
- Variant model
  - N-dimensional space
  - Often realized by conditional compilation with #ifdef's
- Baseline
  - Identify a product as a set of components with specific version and variant configuration

Repository and Workspace

- To allow programmer modifications, SCM tools always maintain a *repository* and programmer-specific *workspaces*
- Repository realization:
  - File (SCCS, RCS)
  - Set of files
  - Tree of files (cvs, subversion)
  - Object base (Damokles)
  - Data base (ClearCase)
  - Graph of files
  - Special repository on file system level (odin, prcs)

Files As Components (SCCS, RCS)

- The earliest configuaration management tools provided versioning on files.
- There is a working version and a repository version of the file
- A change that is commited from the working version to the repository file is called a *delta*
- Users may "undo" changes/deltas and retrieve old versions with version identifiers
- 0.1, 0.2, 0.3, …., 1.0, 1.1, …. 2.0, ….<current>
- If two programmers change concurrently, version numbers branch:
- 1.0, 1.1,
  - 1.2.1.1, 1.2.1.2, 1.2.1.3, ..., 1.2.1.7
  - 1.2.2.1, 1.2.2.2
- Two branches may be also *merged*

RCS commands

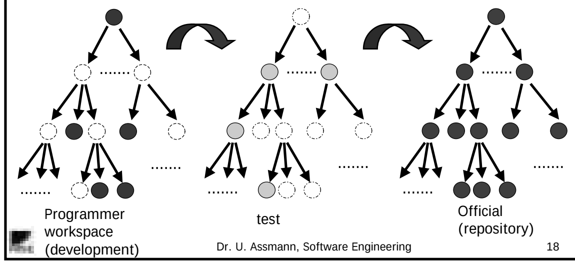
- Checkin (ci)
  - Create a repository file or update it with a working version
- Checkout (co)
  - Create a working version from a repository file
- Rcsdiff
  - Diff two versions
- Rcsmerge
  - Merge two versions
- RCS stores the newest file as such in the repository file and recalculates older versions by undoing deltas (reverse delta technique)
- SCCS stores the oldest file and recalculates newer versions by applying deltas (forward delta technique) Usually slower!

RCS symbolic versions

- RCS can maintain a *baseline*, I.e., a set of versions of files that belong together
  - And assign the baseline a symbolic name, I.e., “newYork”
- RCE (extended RCS) can treat trees and has a visual user interface

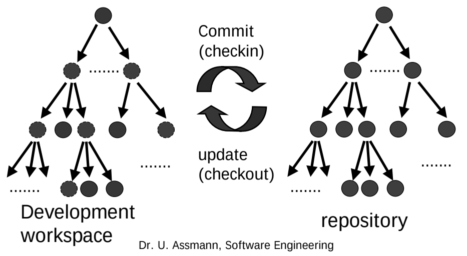
View Pathing on Component Trees (nmake, gmake)

- Programmers have a *view* on the test version which has a *view* on the official version



Simple View Pathing on Trees (cvs)

- Programmers have a *view* on the official version
- All files are physically copied



Operations in CVS

- Checkout
  - Copy a repository (tree) into a development workspace tree.
  - Cvs memorizes the repository location, also on the internet
  - This allows internet collaboration
  - All open source projects use CVS
- Update
  - Update the workspace with newer contents of the repository
- Checkin
  - Copy a development workspace into the repository. Merge differences.
- Diff
  - Differenciate the workspace to the repository

Optimistic Strategy

- CVS uses optimistic commit, i.e., repositories are not locked
- During update operations, it is hoped that the newer contents of the repository can be *merged* with the workspace easily (that's the optimism)
- If two programmers changed the same line in a file differently, a *conflict* occurs that cannot be merged automatically
- The merged file has to be edited by hand to eliminate the "diffed" lines.

Component Selection

Component Selection

- ..also called Artefact or Variant Selection
- Component selection selects a set of components with version and variant from a repository.
- The selected items form a partial baseline or a complete baseline.
- Selection language:
  - Files with piece lists (enumeration of files and version numbers) (cvs, prcs)
  - Attributes of components (variants, versions, branches, symbolic names, states "locked" "frozen", ...)
  - Pattern matching on attributes
  - Logic language to build expressions on patterns
    - Set based language (Adele)
    - Feature logic (Zeller ICE)
  - The more powerful the language is the better conflicts and inconsistencies can be avoided!

Adele

- Estublier, Universite de Grenoble
- Entity-relationship database
- Dependency relations
- Symbolic variant names
- Set expression based variant selection on attribute values, constraints, and preferences
- Query example

  Baseline = (window-system=x11 $\wedge$
  (current $\vee$ status $\neq$ experimental))
- Check on incomplete and inconsistent configurations

Shapetools

- Lampen, Mahler, University of Berlin
  - On top of the UNIX file system
  - Or the object-oriented data base DAMOKLES
- Attributed file system, I.e., nicely integrated with OS
  - Variants are expressed as attributes
    - Name=myfile, type=c, generation=4, revision=3.0, state=saved, mode=06444, testlevel=low, project=newYork
  - Files have states *busy, saved, proposed, published, accessed, frozen*
- Configuration selection by attribute patterns
  - , expresses AND ; expresses OR . expresses end-of-rule:

```
doAnAction:
    *.c, attr(author,andy), attr(state, busy) ;
    *.c, attr(mode=755), attrmax(version).
```

ICE

- ICE builds on feature logic (A. Zeller)
- In feature logic terms with attributes

  screen1 = [object:screen, resolution:high, speed=high]

  screen2 = [object:screen, resolution:medium, speed=low]
- Query example

  Baseline = [window-system:x11,
  {current: $\top$ ‾, status: ~ experimental}

- Feature logic unification and resolution can be used for deduction
- Easy consistency checking

Collaboration

Collaboration Strategies

- Pessimistic:
  - Programmer has to lock parts or the whole repository
  - Others may read, but not change
  - Too restrictive for groups
- Optimistic:
  - Updates are allowed speculatively
  - Merge problems must be solved by hand
  - Almost all tools are now optimistic
  - Only way for internet based open source projects

Combination with Regression Tests

- A regression test feeds a product with a large test suite, and compares the results with that of the previous version
- Regression tests are often poorly supported by SCM tools

Internet Development

- Cvs has a browser interface, cvsweb
- That is the basis of almost all large open source projects now
- Large sites offer space for open source projects and use cvs as the base configuration tool
  - www.sourceforge.net

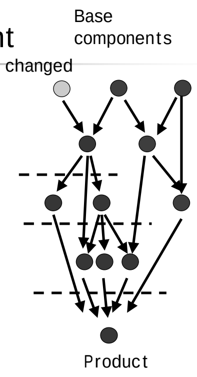
Process Support

Build Support

- Main tool: make (Feldman 1975)
  - Exists in about 30 variants
  - Gmake (gnu) is one of the powerful ones
  - Cook, odin, are other nice ones
- Rule based
  - If component A is newer than its dependent component B
  - Then redo B's production
- Lazy system builds
  - Systems are rebuilt (recompiled, relinked, reconfigured) only if something has changed

Make as Wavefront

- It builds a graph of dependencies between components from the rules
  - Annotates the actions from the rule to the edges
- That must be acyclic
- Make detects which nodes in the graph have changed, I.e., are newer as their successors
- And then executes a wavefront on the dag
  - Executing the production actions annotated to an edge



Auto Configuration

- Gnu autoconf is a makefile generator
  - Generates a *configure* script
  - *configure* generates at installation time makefiles for different machines, operating systems, environments automatically
- Based on macro processing and a macro library
- Other tools
  - Amake
  - Perl tools

Other Tools

PRCS

- Hilfinger, Berkeley
- Is a nice improvement of cvs
- Usually faster since after a commit it does not update workspace files (which cvs does)
- Maintains information about the product centrally in a project file *.prj
- Simpler management of the projects
- Internet version is being worked on

Rational ClearCase

- One of the market leaders in commercial SCM
- Good quality control over complete projects
- Stores attributes with files
- Files are stored in a database
- Support of subsystems (packages): identifiers, releases
- Activity table for merge of subsystems
- Smart recompilation of parts

Subversion

- Newest hit of the internet open source projects
- http://subversion.tigris.net
- Allows everything based on html browsers
- Fully internet based
- Automatic conversion from cvs

Literature

- J. Estublier. Software Configuration Management: A Roadmap. Future of Software Engineering, Limerick, Ireland. ACM 2000.
- A. Midha. Software Configuration Management for the 21st Century. Bell Labs Technical Journal 1997.
- S. Dart. Concepts in Configuration Management Systems. CMU.
- A. Zeller. Unified Versioning through Feature Logic. ACM transactions on Software Engineering and Methodology. Vol 6, No 4, 1997.
- A. Mahler, A. Lampen. An Integrated Toolset for Engineering Software Configurations. IEEE Sup. On Practical Software Development Environments. SE Notes 13, No 5.
- R. Conradi, B. Westfechtel. Version Models for Software Configuration Management. ACM Transactions on Programmming Languages and Systems.