Software Configuration Management

- Change Control
  - goals and issues (more in project mgt lecture)
- Version Control
  - goals, issues, approaches
  - subversion
- Software Builds
  - goals, tools and environment, automation
  - make
- Introduction to Project #1

change control

- who can make what changes, where, when?
- sometimes, some change is good
  - it represent progress as work is completed
  - such changes should be facilitated
- sometimes, some change is bad
  - changes can be disruptive to the product
  - we need processes to detect & prevent these
- hopefully these processes are adaptive
  - adjusting the burden in response to the risk

change control mechanisms

- may be performed by version control tools
  - may control who can modify which files
  - may notify interested parties of changes
  - these features are usually configurable
- may be managed by human processes
  - publication and objection
  - designated component reviewers
  - change control boards
- should have mechanism/policy separation

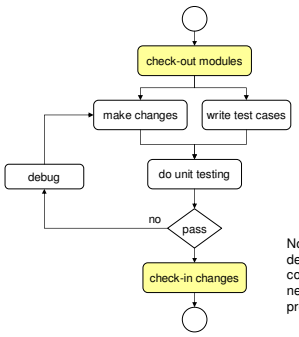
the laws of version control

- All of our work products are versioned
  - we can tell what version we are dealing with
- All official changes are tracked
  - we know exactly what changes were made
  - we know who made each change, when, why
- We can reconstruct any version at any time
  - not just the current version, any prior version
- Files exist in multiple parallel branches
  - each of which has its most current version

version control procedures

- never deliver a work product directly
  - rather, deliver a version-controlled file
  - ensures proper recording of all work
- build from the version controlled files
  - extract specific (or current default) versions
- associate versions with deliverables
  - release has a list of all versions used to build it
  - test/bug reports associated w/specific releases
  - bug fixes are associated w/new file versions
  - work product approvals specify a version

software development process



The checked-out modules may include program source code, test cases, make files, configuration databases, etc.

Note that what development delivers is often not executable code. Rather they check in new versions of their work products.

subversion - keywords

- Automatic module labeling
  - macros that are included in the file text
  - automatically expanded when file is extracted
    (… if they are enabled by the svn **keywords** property)
- Standard keywords

$Date:	date of the last change $
$Revision:	current version number $
$Author:	person who made last change $
$URL:	location of file and repository $
$Id:	all of the above $

subversion - branches

- support parallel code evolution
  - concurrent releases
  - variant products
- implemented as copies of an original tree
  svn copy *original new-copy*
  - perform branch development on new sub-tree
- copy changes from one branch to another
  svn merge –r *first*:*last repository-files*
  - take those changes, make them to local copy

subversion - locking

- proactively prevent conflicting changes
  - only owning user is allowed to change file
- locking commands
  svn lock *file* --message "*text*"
  svn commit *file*
- lock abandoning
  svn unlock *file*
- lock breaking
  svn unlock *file* --force

the laws of software building

- The correctness of software should be a function of the underlying code, not the manner in which it was built.
- All builds (of the same software versions) should yield the same executable bits. No matter who builds them, when they are built, or where they are built.
- Test what you build. Ship what you test.

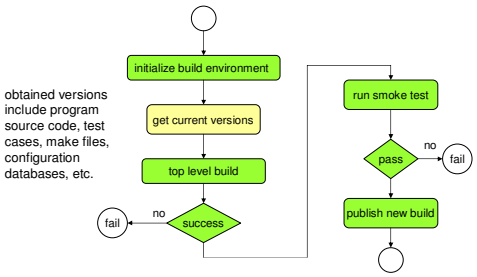
Standard Build Environment

- there is no universal build environment
  - different tool-sets, releases, and platforms
- all of these can affect a built program
  - include files define types, constants, macros
  - libraries contain a huge amount of code
  - different compilers produce different code
- we need a standard build environment
  - compilers, headers, libraries, build tools, …
  - it must be used for all product builds
  - it must be archived for future use

Standard Build Procedures

- software may be very complex to build
  - many components must be built
  - each must be built with correct options
  - pieces must be built in the correct order
  - components must be properly combined
- entire build process must be automated
  - from extraction to installing the binaries
  - to ensure it is done properly/same every time
- build scripts are part of the code

software build process



make – implicit rules

- standard actions for default transformations

```
%o: %.c
        cc –c –o $<
```

- value
  - greatly shorten makefile (dependencies only)
  - put action in one place for easier update
- automatic variables for implicit rules

$@	the desired target
$<	the first prerequisite
$?	the newer-than-target prerequisites
$*	the stem (without suffix)
$%	the target member name (for archives)

make - variables

- user variables
  - form: $(name)
  - can be set explicitly (=, :=, ?=, +=)
  - can be set on command line, in environment
- typical uses
  - lists of file names target: $(objs) $(libs)
  - directory paths cc –I$(include) …
  - commands/options $(CC) -$(COPTS) …

make – control directives

- **include** rules from another file
  - processed as though part of the makefile
  - can be read conditionally, based on variables
  - often used for definitions or modularity
- **if**, **ifdef**, **ifeq**, **ifndef**, **ifneq**, **else**, **endif**
  - conditional processing of rules and directives
- nested **make** invocations
  - actions can include **make** commands
    - with another makefile or in another directory

make – phony targets

- typically operations
  - clean, extract, install, etc.
- if the file does not exist …
  - actions taken when ever target is requested
  - .PHONY directive in case file might exist
- or create a file to record action taken
  - dependency rules can see if it is still up to date
  - e.g. “backup”

Project #1

- Assignment
  - put some source code under subversion control
  - create makefile(s) to automate its construction
  - make, test and put back a few trivial changes
- Purpose
  - Ensure you have a basic level of proficiency with these fundamental tools
- this is an individual project, due Thu 2/1/07
- TAs
  - Christopher Kain ckain@hmc
  - Andrew Glass tglass@hmc

For next lecture

- McConnell 3.1, 20
  - Good overview quality and what affects it
- What is Quality
  - good discussion of an imprecise concept
- What is S/W Quality Assurance
  - excellent background and overview
- What is a post-mortem
  - good introduction to goals and process
- Anatomy of a Retrospective
  - example of a facilitated difficult post-mortem (skimming this briefly will be fine)