CALIFORNIA STATE UNIVERSITY
FULLERTON™

COLLEGE OF ENGINEERING
AND COMPUTER SCIENCE

Advanced Software Process

Part II: The Repeatable Process

7. Software Configuration Management (I)

Dave Garcia-Gomez
Faculty / Lecturer
Department of Computer Science

Course Roadmap

Part I: Software Process Maturity
- 1. A Software Maturity Framework
- 2. The Principles of Software Process Change
- 3. Software Process Assessment
- 4. The Initial Process

Part II: The Repeatable Process
- 5. Managing Software Organizations
- 6. The Project Plan
- 7. Software Configuration Management (Part I)
- 8. Software Quality Assurance

Part III: Defined Process
- 9. Software Standards
- 10. Software Inspections
- 11. Software Testing
- 12. Software Configuration Management (continued)
- 13. Defining the Software Process
- 14. The Software Engineering Process Group

Part IV: The Managed Process
- 15. Data Gathering and Analysis
- 16. Managing Software Quality

Part V: The Optimizing Process
- 17. Defect Prevention
- 18. Automating the Software Process
- *19. Contracting for Software*
- *20. Conclusion*

Software Configuration Management (Part I)

- The Need for Configuration Management
- Software Product Nomenclature
- Basic Configuration Management Functions
- Baselines
- Configuration Management Responsibilities
- The Need for Automated Tools

Software Configuration Management

- Change management is one of the fundamental activities of SE.
- Changes to the requirements drives the design, and design changes affect the code.
- For even modest-size projects, the number of people involved and the change volume generally require a formal change management system.
  - Software configuration management

Software Configuration Management

- Since a key objective of the software process is to have change activity converge until the final product is stable enough to ship, the management of all changes is important.
- Focus on code control
  - Control of requirements and design changes is also critically important, but these functions can often be handled as enhancements to a code management system.

Software Configuration Management

Change management, more commonly called software configuration management (SCM), is a set of activities designed to manage change by identifying the work products that are likely to change, establishing relationships among them, defining mechanisms for managing different versions of these work products, controlling the changes imposed, and auditing and reporting on the changes made [Pressman 2005].

The Need for Software Configuration Management

- The most frustrating software problems are often caused by poor configuration management.
- The problems are frustrating because they take time to fix, they often happen at the worst time, and they are totally unnecessary.
- Configuration management helps to reduce these problems by coordinating the work products of the many different people who work on a common project.
  - Without such control, their work will often conflict, resulting in problems.

The Need for Software Configuration Management

Problems:

- Simultaneous update
  - When two or more programmers work separately on the same program
- Shared code
  - When a bug is fixed in code shared by several programmers
- Common code
  - When common program functions are modified
- Versions
  - Evolutionary release

The Need for Software Configuration Management

Control system to answer

- What is my current software configuration?
- What is its status?
- How do I control changes to my configuration?
- How do I inform everyone else of my changes?
- What changes have been made to my software?
- Do anyone else's changes affect my software?

Software Product Nomenclature

- Since the role of configuration management is to control the development of the system elements as they are built and then combined into a full system, it is important to use common system terminology.
- The design process starts by successively partitioning the system until a satisfactory level of detail is reached.
  - At this point, implementation begins with these smallest pieces, which are progressively assembled and tested until the total system is completed.

Software Product Nomenclature Figure 7.1

- System
  - The package of all the SW (and HW) that meets the user's requirements.
- Subsystem
  - Large systems can have may subsystems, such as communications, display, and processing.
- Product
  - Subsystem typically contain many products. For example, OS contains a control program, compilers, utilities, and so forth.
- Components
  - A control program could be made of such components as the scheduler and I/O controller.
- Module - Components consist of a number of modules.

Tests Supporting Implementation Process

- Unit test
  - Test of each individual module
- Integration test
  - As the modules are integrated into components, products, subsystems, and systems, their interfaces and interdependencies are tested.
- Function test
  - When integration results in a functionally operable build, it is tested successfully in component test, product test, subsystem test, and finally in system test.
- Regression test
  - At each integration step (spin), a new build is produced.
  - This is first tested to ensure that it has not regressed.

Basic Configuration Management Functions

- Once an initial product level has stabilized, a first baseline is established.
- With each successive set of enhancement, a new baseline is established in step with development.
- Each baseline is retained in a permanent database, together with all the changes that produced it.
- Configuration Management Overview [Figure 7.2]

Basic Configuration Management Functions

- The baseline is thus the official repository for the product, and it contains the most current version.
- Only tested code and approved changes are put in the baseline, which is fully protected.
  - It is the official source that all the programmers use to ensure their work is consistent with that of everyone else.

Basic Configuration Management Functions

Software Configuration Management (SCM)'s key tasks:

- Configuration control
- Change management
- Revisions
- Versions
- Deltas
- Conditional code

Configuration Control

- The task of configuration control revolves around one official copy of the code.
- The simplest way to protect every system revision is to keep a separate official copy of each revision level.
- While this can take a lot of storage, the most serious issue concerns code divergence.

Revisions

- Keeping track of revisions is an important task of configuration management.
- In large system, some programs, such as the control program, provide essential function for all the others.
- Once a component's modules have been integrated into a testable unit, it is then tested with the latest control program level.
- When new problems are found, previous tests can be rerun to trace the problem source.

Revisions

- If an early driver is now reproduced using the latest versions of the control program modules, some of these modules will likely be different, so the prior tests will not be exactly reproduced.
- The answer is to keep track of every change to every module and test case.
- There is one latest official version and every prior version is identified and retained.
- These obsolete copies can then be used to assist in tracing problems.

Derivations

- The ability to determine what has changed is one of the most powerful SW testing aids.
- Derivations [Figure 7.3]
- A derivation record would show that Test A was run on control program level 116 and the rerun was attempted on level 117.
- Changes X and Y could then be identified readily.

Derivations

- Information maintained in derivation records:
  - The revision level of each module
  - The revision levels of the tools used to assemble, compile, link, load, and execute the programs and tests
  - The test cases used and their revision levels
  - The test data employed
  - The files used
  - The software and hardware system configuration, including peripherals, features, options, allocations, assignments, and HW change levels
  - The operational procedures
  - If not a stand-alone test, a record of the job streams being executed.

Versions

- Often, several different functions can be implemented by the same module with only modest coding differences.
  - Version [Figure 7.4]
- As a project progresses, many versions of individual work products will be created.
- The repository must be able to save all of these versions to enable effective management of product releases and to permit developers to go back to previous versions during testing and debugging.
  - Revision numbering scheme

Deltas

- The use of versions solves the problem of different functional needs for the same module but introduces multiple copies of the same code.
  - The reason is that most of the code in the two memory management modules would be identical.
  - One, however, would have an additional routine to handle memory limits testing and mode switching.
  - Since they are stored as separate modules, however, there is no way to ensure that all the changes made to one are incorporated in the other.

Deltas

- One way to handle this is to use deltas.
- This involves storing the base module together with the changes required to make it into another version.
- Deltas [Figure 7.5]
- Very useful in some situations, but has some disadvantages
  - Use the delta approach for temporary variations

Conditional Code

- Another way to handle slight variations between modules is to use some form of conditional program construction.
- A source program would contain various different versions constructed depending on different conditions, but none would be included in the final system unless called for at system installation.
- This need for conditional code is often met by using system generation options.

Conditional Code

- These might include conditionally selecting one of several optional modules depending on the particular condition that the program was used.
- All modules would be in the source library, but only one would be used at any time.
- Conditional Code [Figure 7.6]

Baselines

- The baseline is the foundation for configuration management.
- It provides the official standard on which subsequent work is based and to which only authorized changes are made.
- After an initial baseline is established and frozen, every subsequent change is recorded as a delta until the next baseline is set.

Baselines

- It is desirable to establish a baseline at an early point in every project.
- Establishing a baseline too early, however, will impose unnecessary procedures and slow the programmers' work.
- As long as the programmers can work on individual modules with little interaction, a code baseline is not needed.
- As soon as integration begins, however, formal control is essential.

Baselines Scope

- Baseline items included for the implementation phase:
  - The current level of each module
  - The current level of each test case
  - The current level of each assembler, compiler, editor, or other tools used
  - The current level of any special test or operational data
  - The current level of all macros, libraries, and files
  - The current level of any installation or operating procedures
  - If the project involves operating system or control program changes, it may also be necessary to retain data on the computing system configuration and its change level.

Baseline Control

- SCM must ensure appropriate baseline control while providing flexible service to the programmers.
- On the other hand, the baseline must be protected against unauthorized change; at the same time, the programmers should be able to readily modify and test their code.
- This controlled flexibility is accomplished by providing the programmers with private working copies of any part of the baseline.

Baseline Control

- They can thus try out new changes, conduct tests, or make trial fixes without disturbing anyone else.
- When they are ready to incorporate their work into a new baseline, however, care must be taken to ensure that the new changes are compatible and that no new code causes regressions.

Configuration Management Records

- To maintain control over all code changes, several procedures are used.
- First, every change proposal is documented and authorized before being made.
  - Sample Change Request Contents [Table 7.1]
- As the change is made, a record is kept of what was done.
  - Sample Change Log Contents [Table 7.2]
- The results of each test should be documented.
  - Sample Test Report Contents [Table 7.3]

Configuration Management Records

- Detailed test records are particularly crucial when HW and SW changes are made simultaneously.
- Perhaps the single most important project control document is the problem report.
  - Sample Problem Report Contents [Table 7.4]

Configuration Management Records

- Another important report is the expect list.
  - This is a detailed listing of every function and feature planned for every component in each new baseline.
  - All developers who work on program changes for a new baseline should maintain a current list of all the functions, features, fixes, and interfaces to be provided by their program when delivered to integration.
  - This information is made available to all development groups, so they know of any changes affecting them.

Configuration Management Responsibilities

- To implement the necessary controls and procedure, a number of responsibilities need to be established.
- Depending on the size of the system and the number of people involved, they may be handled by a single individual, several people, or an entire organization.
- The basic responsibilities are:
  - Configuration manager
  - Module ownership
  - Change Control Board (CCB)

Module Ownership

- Maintaining design integrity is a common problem in large systems with periodic enhancements.
- One simple but effective approach is to designate a programmer as owner of each module.
- Since there are often many more modules than programmers, one person generally owns several modules at one time.

Module Ownership

- The module owner's responsibilities are:
  - Know and understand the module design
  - Provide advice to everyone who works on or interfaces with the modules
  - Serve as technical control point for all module modifications, including both enhancement and repair
  - Ensure module integrity by reviewing all changes and conducting periodic regression tests

The Change Control Board

- On moderate to very large projects, a central control mechanism is needed to ensure that every change is properly considered and coordinated.
  - Change Control Board (CCB)
  - Configuration Control Board (CCB)
  - Members from development, documentation, test, assurance, maintenance and release
  - Its purpose is to ensure that every baseline change is properly considered by all concerned parties and that every change is authorized before implementation.

The Change Control Board

- Information typically needed by a CCB on each proposed change:
  - Size
  - Alternatives
  - Complexity
  - Schedule
  - Impact
  - Cost
  - Severity
  - Relationship to other changes
  - Test
  - Resources
  - System impact
  - Benefits
  - Politics
  - Change maturity

The Change Control Board

- If the change is to fix a customer-reported problem, other information may also be needed:
  - A definition of the problem the change is intended to fix
  - The conditions under which the problem was observed
  - A copy of the trouble report
  - A technical description of the problem
  - The names of any other programs affected

The Change Control Board

- Additional information should also be available prior to release:
  - Final source and/or object code for the change
  - Final documentation changes
  - Evidence of successful inspection and test of the code and documentation
- The CCB not only determines whether the change should be made but also the conditions under which it can be released.

CCB Problems

- Since the CCB must consider every change, its workload can become excessive, particularly during the final test and release phases of a large project.
- This is one reason why multiple CCB's are established for specific components or test phases.

CCB Problems

- It is important to select the CCB members with care.
- They have the power to block any part of the project so these assignment should not be treated lightly.
- Generally, in fact, the project manager should personally chair the highest-level CCB.

The SCM Organization

- A further question concerns the dividing line between configuration management and computer operations.
  - These two organizations must coordinate, control, and track every change so that between them they can reproduce every program or test environment.
  - Project-unique changes generally should be handled by configuration management and system-wide changes by computer operations.

The Need for Automated Tools

- The final development stages of even modest-sized projects involve a lot of detail.
- Regardless of the care taken in product design, quality can be destroyed by sloppy last-minute changes.
  - Changes as object patches with the intent of later updating the source may cause serious problems if equivalent source changes are not later made and tested for every case.

The Need for Automated Tools

- If any one change is overlooked or its consequences are not properly considered, unpleasant problems are likely in system test, acceptance test, or customer use.
- Automated system are thus generally needed to handle the large amount of detail.

The Change Management System

- The change management system tracks the change requests, problem reports, CCB actions, and activity log entries.
- Data is recorded on what was done, by whom, when, what remains to be done, any special conditions, and current status.
  - The system should maintain a record of the changes not closed, the changes to a particular component, and the oldest outstanding changes.
  - The system tracks the mean time to change closure and maintains statistics on the change backlog.

The Change Management System

- The change management system tracks the change requests, problem reports, CCB actions, and activity log entries.
- Data is recorded on what was done, by whom, when, what remains to be done, any special conditions, and current status.
  - The system should maintain a record of the changes not closed, the changes to a particular component, and the oldest outstanding changes.
  - The system tracks the mean time to change closure and maintains statistics on the change backlog.

The Change Management System

- Change activity is also a powerful indicator of project status.
- With change data, management can pinpoint problems early and take timely action in those areas where the problems are most severe.
- Module Change Tracking [Figure 7.7]

The System Library

- The system library stores the development work products, including the source and object code for every baseline and change, the test cases, and the development tools.
- It provides locks to prevent unauthorized changes, and it has the capability to build the various system configurations, test drivers, and test scenarios or buckets required by development.

The System Library

- Primary functions of the system library:
  - Retrieving objects by name, date of creation, author, component, or development status
  - Identifying object relationship
  - Executing a make function, such as building a product release, building a private system copy, …
  - Performing such service functions as checking status, verifying the presence of all build items, …
  - Providing the required levels of control and protection
  - Gathering and reporting any desired statistical and accounting information

References

Humphrey, Watts S., *Managing the Software Process*, The SEI Series in Software Engineering, Addison-Wesley, 1989. (29th Printing, May 2003) (ISBN 0-201-18095-2)