Software Project Management

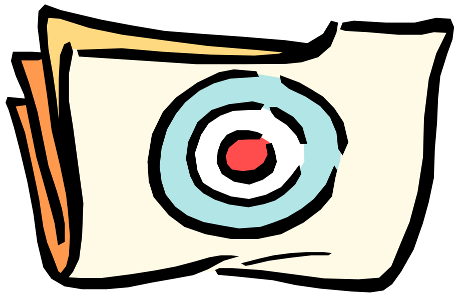
Software Configuration Management

Overview

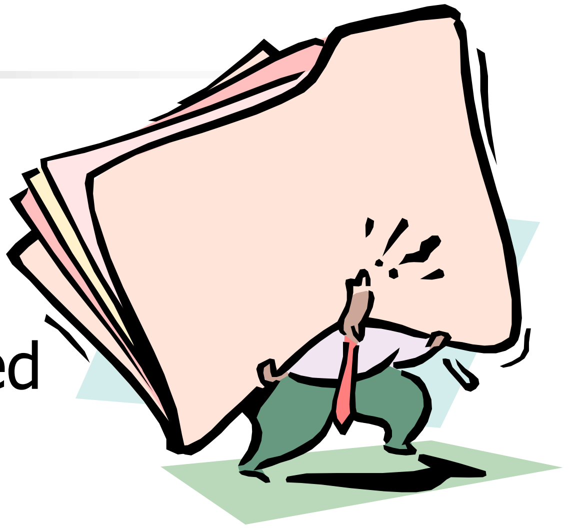
- In this lecture we shall cover,
  - Software evolution (types of changes)
  - Configuration management (need for it)
  - Facets of SCM
  - Change control board
  - Change management
  - Auditing and status accounting

Introduction

- Ideal:
  - Software is developed from stable/frozen requirements
  - The concept is that it is easier to hit a stationary target than a moving target
- Reality:
  - Not applicable for most real-world systems
- The only constant is "CHANGE"
  - An effective software project need to have a strategy to tackle "CHANGE"



Software Evolution

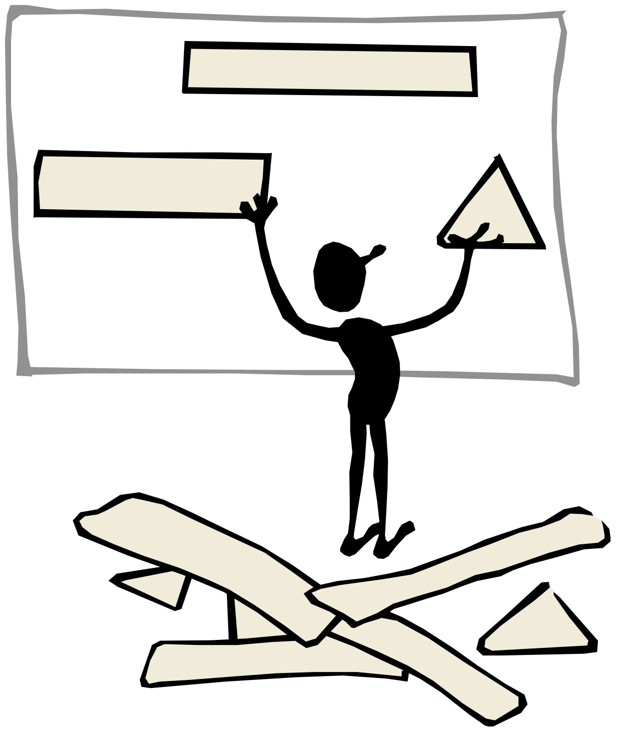


- Software evolves over a period of time
  - Many different items are produced over the duration of the project
  - Different versions are produced
  - Teams work in parallel to deliver the final product
- Software evolution implies a constantly changing system

How software changes…

- The four aspects of software evolution are:

1. Corrective changes
2. Adaptive changes
3. Perfective changes
4. Preventive changes



Corrective Changes

- Required to maintain control over the system’s day-to-day functions
- These changes are made as faults (or) bugs are found during the development time
- Some changes may be long-term and fundamental, some may be patches to keep the system in operation (emergency fixes)

Adaptive Changes

- Essentially maintaining control over system modifications
- As one part of the system changes, other impacted areas will need to be updated
- Examples
  - Database upgrades
  - Use of a new compiler or development tool

Perfective Changes

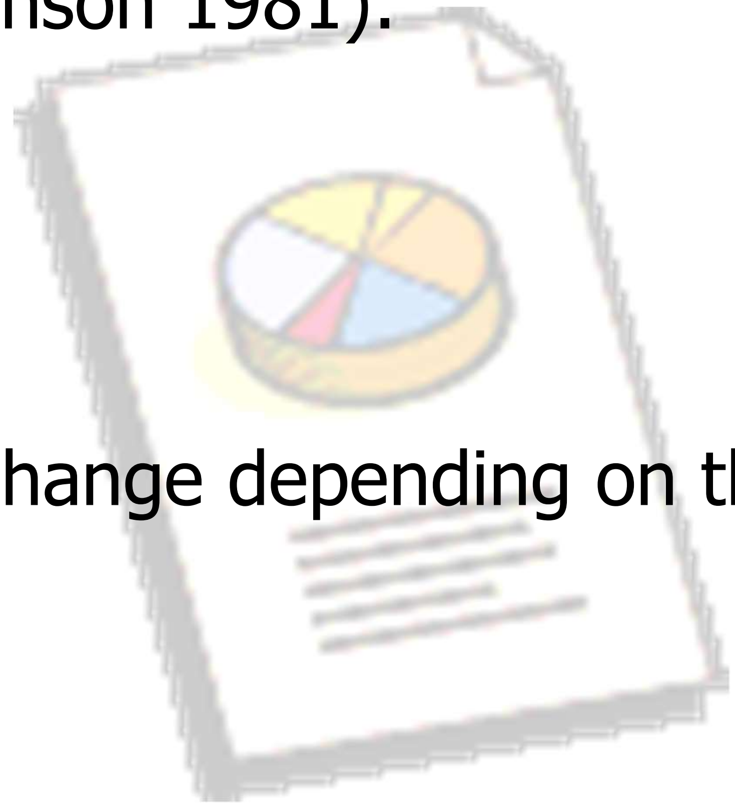
- Perfecting existing acceptable functions
- The domain of Refactoring designs falls into this category
- Perfective changes are done to increase the long-term maintainability or elegance of the solution
  - Involves changes to design or data structures for better efficiency
  - Updates to documentation to improve their quality
  - Enhancing the code to make it more readable

Preventive Changes

- Preventing the system performance from degrading to unacceptable levels
- Involves alterations made to ensure that the system has a defense against potential failures
- Example:
  - Adding extra redundancy modules to ensure that all transactions are properly logged

Types of Changes

- The typical distribution of these changes is (from Lientz & Swanson 1981):
  - Perfective (50%)
  - Adaptive (25%)
  - Corrective (21%)
  - Preventive (4%)
- These figures will change depending on the system and project



Changes and Control

- If changes are not controlled in a project – things can and will get out of hand
- The issue of change management is even more important when multiple people work on a project as well as on the same deliverable
- Without proper strategies and mechanisms to control changes – one can never revert back to an older more stable copy of the software
  - Important as every change introduces risk into the project

So what is the answer??

- The facts:
  - Change is unavoidable in software
  - Changes needs to be controlled
  - Changes need to be managed
- The solution
  - Software configuration management (SCM)

Configuration Management…

- This is the discipline that applies a rigorous approach to ensure
  - Different items produced in software systems are all identified and tracked
  - Changes to the various items are recorded and tracked
  - Completion and proper integration of all the various modules

Configuration Management

- SCM can help determine the impact of change as well as control parallel development
- It can track and control changes in all aspects of software development
  - Requirements
  - Design
  - Code
  - Tests
  - Documentation

Need for SCM…

- As software evolves – many resources make changes to the system
  - CM prevents avoidable errors that arise from conflicting changes
- Often many versions of the software are released and require support
  - CM allows a team to support many versions.
  - CM allows changes in sequential versions to be propagated
- CM allows developers to track changes and reverse any fatal changes to take a software system back to its last known safe state

Need for SCM

- Good SCM increases confidence that we are:
  - Building the right system
  - Testing the system enough
  - Changing it correctly and carefully
- It also:
  - Restrains non-essential changes
  - Ensures that decisions and changes are traceable
  - Increases accountability
  - Improves overall software quality
  - Provides a fall back position when things do not work

Facets of SCM

- The four core aspects of SCM are:
  1. Configuration identification
  2. Configuration control and change management
  3. Configuration auditing
  4. Status accounting

Configuration Identification…

- A large number of items are part of the software development process:
  - Source and binary modules
  - Hardware and operating systems
  - Documentation
    - Requirements
    - Design
  - Test cases
  - Etc…
- Key is to identify the items that need to be under SCM

Configuration Identification…

- Configuration identification is the process of establishing a baseline from which system changes are made – allows for control.
- So what needs to be under SCM?
  - Items where changes need to be tracked and controlled
  - If in doubt, add it into the inventory of items under SCM
- Common items under SCM are:
  - Source code, documentation, hardware/OS configuration

Terminology Review – 1

- **Configuration items** - any single atomic item for which changes need to be tracked
  - Source code file
  - The project plan
  - The documentation standard
  - …
- **Baseline** - A product that has been formally approved, and consists of a well-defined set of consistent configuration items

Terminology Review - 2

Baseline

"A specification or product that has been formally reviewed and agreed to by responsible management, that thereafter serves as the basis for further development, and can be changed only through formal change control procedures"

[Bruegge]

Baseline Types

- As the system is developed a number of baselines are created:
  - *Developmental baseline* (RAD, SDD, integration test, ...).
    - Goal: coordinate engineering activities.
  - *Functional baseline* (prototype, technology preview, alpha, beta release).
    - Goal: obtain customer experiences with functional system.
  - *Product baseline* (GA with a version - win95, word 2000).
    - Goal: coordinate sales and customer support.

Managed Directories

- Programmer's directory (IEEE: dynamic library).
  - Library for holding newly created or modified software entities. The programmer's workspace is controlled by the programmer only.
- Master directory (IEEE: controlled library).
  - Manages the current baseline(s) and for controlling changes made to them. Entry is controlled, usually after verification. Changes must be authorized.
- Repository (IEEE: static library).
  - Archive for the various baselines released for general use. Copies of these baselines may be made available to requesting organizations.

Configuration Identification

- A few notes…
  - Starting too early can add too many items that may really not require full configuration management.
  - Starting too late will result in a disaster
  - It is common to have 1000+ items under SCM
- A good start
  - Place all documents under SCM
  - Add code as it starts to be available
  - Remove older items and archive them
  - Remove items where the changes are minor, rare and need not be under the purview of a complete SCM

Version Allocation…

- Once a configuration item (CI) has been identified – a proper version number must be allocated
- The best option is to start with a major-minor versioning scheme
  - Major version numbers are between 0 – n
  - Minor version numbers should be between 0 – 100

Version Allocation…

- Examples:
  - Report.Java (version 1.23)
    - Major version: 1.0
    - Minor version: 23 (indicative of number of revisions to this file)
  - Project plan (version 6.34d)
    - Major version: 6
    - Minor version: 34
    - The "d" is indicative of "draft"
- Versioning scheme is developed by the company to suite their needs

Version Allocation…

- Often many companies prefix the configuration item based on its type.
  - Documentation may be prefixed “doc”
  - Source code can be “src”
  - Example: doc-pmp-2.34
    - Project management plan document (version 2.34)

Version Allocation

- New versions of software can be:
  - Maintenance releases
  - Minor upgrades
  - Technology refresh or major upgrades
  - Technology insertion

Baseline Levels

- The software system can be tagged at various stages of its evolution with a baseline number
  - **Development baseline** "n" (where the "n" can be indicative of the 10% of functionality implemented)
  - **Testing baseline** (where a specific build is created for the specific purpose of testing)
  - **Release baseline** (where the software is built for GA)
- There is no rule on when to baseline – but a good guideline is to have one a week

Terminology Review – 3

- **Version** – an *initial* release or re-release of a configuration item (*ideally different versions should have different functionality*)
- **Revision** – minor *changes* to a version that correct errors in the design/code (*typically revisions do not affect expected functionality*)
- **Release –** the *formal distribution* of a ***baseline***

Configuration Control and Change Management

- Review of change activity can highlight what is changing and what is not.
    - Impact of change can be measured over time
- Issues to consider are:
    - Keeping track of changes (deltas or separate files)
    - Allows for parallel development on a single item (many developers updating the same file)

Deltas Vs Separate Files

- After the initial baseline has been established – the item is said to be under SCM.
- Changes can be tracked as:
  - Deltas: *only the changed portion is stored*
  - Separate file: *changes are stored in a new file*
- Deltas work best for text files
- Separate files is a good idea for binary file formats.

Parallel Development

- Many members can often work on the same item:
  - Two developers update the same code file (working on different functions)
  - A number of engineers may be working on a single word document containing the specifications
- Changes are tracked by each user and often merged regularly to create a synchronized version:
  - Merge conflicts are resolved via normal channels of communications
  - Effective management can reduce merge conflicts

Change Management – 1

- For best results changes should be handled formally
  - A change control board (CCB) is necessary
- CCB consists of all key stakeholders
  - Customers
  - Developers
  - Designers and architects
  - Management
  - Business strategists and financiers

Change Request

- Changes are required because:
  - A problem is discovered (bug?)
  - An enhancement is required
- Once a change is required – a "change request" is raised
- A change request (CR) will outline:
  - Current operation, nature of problem/enhancement, expected operation after system is changed

Change Control Board – 1

- All change requests (CR) are reported to the CCB for review
- CCB discusses all open CRs at regular meetings (frequency is determined by nature of project)
- CCB determines if CR identifies a “problem” or an “enhancement”
  - This is done to identify who “pays” for the change

Change Control Board – 2

- Once the change has been categorized, it is discussed in detail,
  - Probable source of problem
  - Impact of the change
  - Time and resource requirements (estimates)
- CCB will assign a priority and severity for all CRs (CRs may also be rejected)
- The CRs are assigned to development management for further action

Impact Analysis

- Before changes are made often a deep or shallow impact analysis is performed
- Impact analysis makes full use of software metrics
- Managers often track
  - Increases in complexity measures as system evolved over a period
- Trend analysis is performed based on modules and change requests to ensure flexibility

Change Management – 2

- Development managers will assign a CR to a developer (or a team)
- The requested change is made as per the plan and a full regression test suite is executed
- Configuration manager reviews the changed system
  - Ensures that all required documentation is changed
  - Ensures that the impact does not exceed estimates (too much)

Configuration Management

- To ensure proper tracking the following information needs to be collected:
  - When was the change made
  - Who made the change
  - What was changed (items modified)
  - Who authorized the change and who was notified
  - How can this request be cancelled
  - Who is responsible for the change
  - Priority and severity
  - How long did the change take (vs estimate)

Change Management – 3

- Changes will need to be modified or cancelled
  - This is required if the team assigned the change request need more resources and time than estimated
- These decisions are delegated to CCB with the extra information added along with the CR.
- A CR can be *assigned*, *deferred*, *rejected*, *closed*.
- All changes are reviewed and modified

Change Control Board – 3

- The complexity of the change management process varies with the project
- Small projects can perform change requests informally
- Complex projects require detailed change request forms and the official approval by one or more managers
- For safety critical projects – change management is very rigorous
  - Example: software changes to an airplane avionics system
- Change request management is supported by robust software packages

Change Management – 4

- To ensure effective change management:
  - Each working version is assigned an identification code
  - As a version is modified, a revision code or number is assigned to each resulting changed component
  - Each component's version and status as well as a history of all changes are tracked