Software Engineering

Software Implementation, Configuration Management, and Quality

Dr. Jayashree R
Department of Computer Science and Engineering

Teaching Assistant: Aanchal Narendran (Sem VII)

Acknowledgement: Dr. Phalachandra HL, authors of the prescribed textbooks and materials sourced online.

Software Engineering

Software Implementation

Dr. Jayashree R

Department of Computer Science and Engineering

Context

Prior to Implementation

- Choose between compiled/interpreted language
- Decide on development environment
- Following a Configuration Management Plan

During Implementation

- Use Coding Standards and Coding Guidelines
- Follow language syntax
- Address Quality, Security and Testability
- Provide Unit Tested, Peer-Reviewed functioning code

Introduction

Software Implementation: Detailed creation of working Software through a combination of coding, reviews and Unit Testing

GOALS

- Minimize Complexity
- Anticipate Change
- Verifiable
- Reuse
- Readable

Characteristics

Produces High Volume of Configuration items

Eg: Source Files, Test Cases

Extensive usage of Computer Science Knowledge

Eg: Algorithms, Coding Practices

Related to Software Quality

Eg: Clean and Maintainable Code

Tool Intensive

Eg: Compilers, Debuggers, GUI Builders

Choice of Programming Language

Based on level of Abstraction

- **Assembly Languages:** map directly onto CPU architecture
- **Procedural Languages:** modest level of abstraction from underlying layer
- **Aspect Orientation Support:** allow separation of aspects during development
- **Object-Oriented Languages:** allows the developer to code in terms of objects

Choice of Development Environment

Commercial vs Open Source

Support of Development Process

Security

Account for future capabilities (product and team)

Integration between tools and environment

Coding: Journey of a Bug

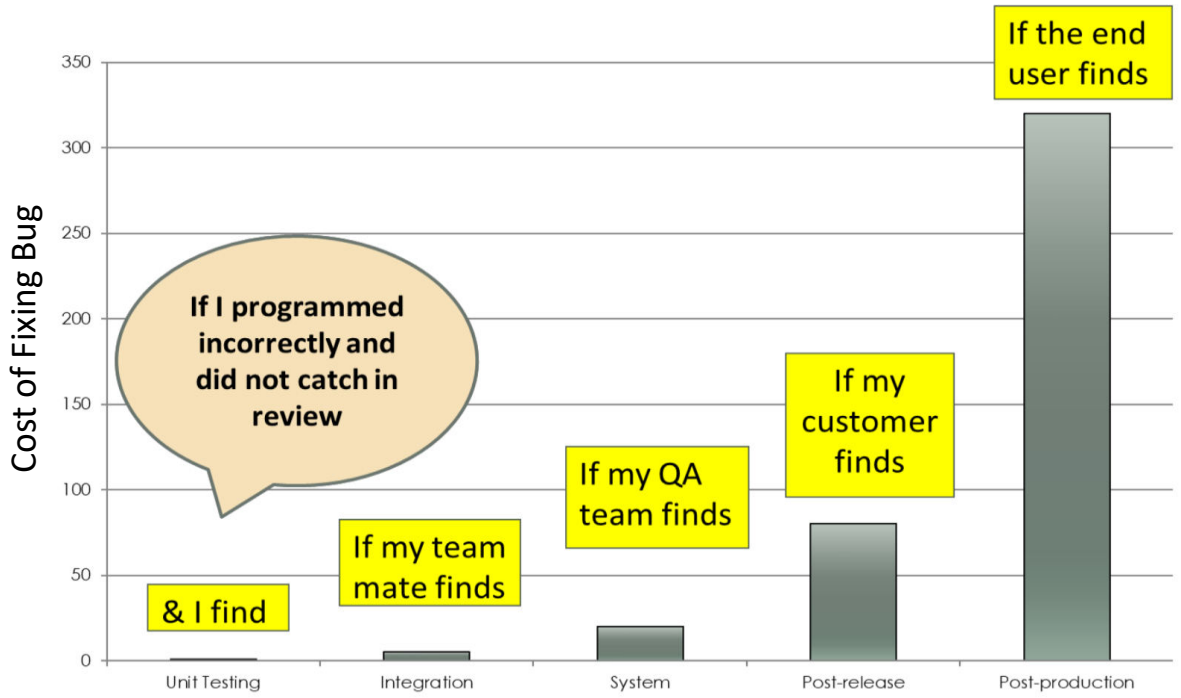
Coding Activity begins once development environment is chosen

Example: Buggy Code

```
if(x<99)
    process_event()
```

- else??? Not Specified and results in unpredictable behavior when x=99

- **Locations to Catch a bug**
  - Coding
  - Code Review
  - Unit Testing
  - Integration Testing
  - System Testing
  - Field Trials
  - Production



Program Personalities

- Programs have personalities: messy, verbose, cryptic, neat

- Reflective of the programmers personality
  - Under documenter
  - CYA Specialist
  - CIO types (pass responsibility to other modules)
  - Dynamic types (create variables on the fly)
  - Fakers (have repetitive code)
  - Multitasker (uses wrappers and glue instead of rewriting)
  - True Believer (Extensive Documentation)

THANK YOU

Dr. Jayashree R
Department of Computer Science and Engineering

jayashree@pes.edu

Software Engineering

Software Implementation, Configuration Management, and Quality

Dr. Jayashree R
Department of Computer Science and Engineering

Teaching Assistant: Aanchal Narendran (Sem VII)

Acknowledgement: Dr. Phalachandra HL, authors of the prescribed textbooks and materials sourced online.

PES UNIVERSITY ONLINE

Software Engineering

Code Characteristics, Standards, Factors for Effective Coding

Dr. Jayashree R

Department of Computer Science and Engineering

Characteristics of Code

Programs Should be Simple and Clear

Naming

Structuring

Easy to Read and Understand

Code Characteristics: Programs should be Simple and Clear

Programs should be well thought out, designed and structured although programmers have a tendency to jumpstart

Programmers should use a reasonable amount of

- Lines per function
- Lines/Functions per file
- Arguments per function
- Levels of Nesting
- Conditions

Code Characteristics: Naming and Naming Conventions

Naming is Important!!

Naming Conventions

- Names have to be meaningful and descriptive
- Avoid using names similar to keywords
- No variables name must start with _
- Use a minimum of 2-3 characters and a maximum of 30 characters in a name
- Do not use Numeric values that are use elsewhere in a name

Types of Naming

Pascal Casing
(first letter Upper Case)

Eg: BackColor

Usage: Class/Variable Names, Method Parameters

Camel Casing
(first letter Upper Case except first word)

Eg: backColor

Usage: Method Names

Underscore Prefix
(after underscore, camel case)

Eg: _backColor

Usage: for internal use in Class/Methods

Code Characteristics: Naming – In Class Exercise

Identify the Naming issue with the below block of code

```
int uglyduck(int timer_count)
{
   if(timer_count<=1)
      return 1;
   else
      return timer_count*uglyduck(timer_count-1);
}
```

Rewrite the following block of code with better Naming

```
for(i=0; i < nelems; i++)
{
    sum += elems[i]
}
```

Follow up question: Would the variable names work in Python?

Code Characteristics: Structuring

Structure: Dependencies between software components (subprograms, parameters, code blocks, etc)

Dependencies can be highlighted via proper naming and layout

For example: dependent subprogram interface/implementation declarations should be closer

File Structure:

- **Logically grouped**
  - #includes, #defines, structures, functions
- **Well partitioned**
  - Decide what goes into header files and C files
  - Which functions/variables are to be exposed

Code and Data Structure:

- Well encapsulated and logically grouped
- Properly initialized

Code Characteristics: Ease of Reading and Understanding

Readability depends on identifier naming and visual layout of statements.

For example: usage of standardized indentation, blank lines and space to illustrate blocks of code and hierarchy

Can you improve these blocks of code

```
a. for(int i=0;i<40;i++)
```

```
b. for(i=0;i<len;i++) if num==array[i] break;
   if (i==len) return -1; else return i;
```

```
c. if (c>='0' && c<='9' || c>='a' && c<='f' || c>='A' && c<='F')
```

Code Characteristics: Ease of Reading and Understanding

Comments should concisely and clearly explain the logic of the program.

They should not be cryptic and misleading.

Must be easy to identify using proper indentation, spacing and highlighting.

Comment the below block of Code

```
function sourceCodeComment () {
  var comment = document.getElementbyID("Code Comment").value;
  if (comment != null && comment != '') {
        return console.log("Thank you for your comment.")
}
```

Food for thought: Is it better to comment each line of code individually or add one comment for the overall function?
Think of a case where in-line comments would be helpful.

Good Programming style

Do's

1. Use a few standards and agreed upon control constructs
2. Use GOTO in a disciplined manner Use user-defined data types to model entities in the problem domain
3. Hide data structures behind access functions
4. Use appropriate variable names
5. Use indentation, parentheses, blank spaces and lines to enhance readability

Don'ts

1. Don't be too clever
2. Avoid null Then statements (dangling if)
3. Avoid Then if statements
4. Don't nest too deeply
5. Don't use the same identifier for multiple purposes
6. Examine routines that have more than five formal parameters

Coding Standards and Guidelines: Introduction

Standards: Rules which are mandatory to be followed

Guidelines: Rules which are recommended to be followed

However, they are often used interchangeably

- Practiced as a checklist, enforces as a part of reviews

Examples of Standards and Guidelines

- Standard headers for different modules
- Naming conventions for local and global variables
- One declaration per line
- Avoid magic numbers

Coding Standards

- Provides a uniform appearance to code written by different engineers
- Improves readability, maintainability and reduces complexity
- Proactively addresses some commonly occurring issues with code
- Helps in code reuse
- Promotes sound programming practices and increases efficiency of programmers

Examples

- Rules for usage of global about which type of data
- Standard headers for different modules
- Naming conventions
- Error and Exception handling conventions

Some common coding practices

Defensive programming	Secure programming	Testable programming

Coding Standard Practices: Defensive Programming

Murphy's Law: If anything can go wrong, it will

- Redundant code is incorporated to check system state after modifications
- Implicit assumptions are tested explicitly

Given the below code block, what modification could be made to adhere to the defensive programming standard?

```
pid = fork();
If (pid==0)
{
    /* child process is created; execute commands */
}
else
{
   /* parent process commands are executed */
}
```

Coding Standard Practices: Secure Programming

Secure Coding: Developing computer software to guard against accidental introduction of security vulnerabilities (Defects, Bugs, Logic Flaws)

Some practices are:

1. **Validate Input:** Validation of input from all untrusted data sources
2. **Heed Compiler Warnings:** Compile using highest warning level and eliminate warnings via code modifications
    1. Use static and dynamic analysis tools to detect additional flaws
3. **Default Deny:** access decisions are based on permissions
4. **Adhere to Principle of Least Privilege:** process should execute with least set of privileges
    1. Elevated permissions for least amount of time
5. **Sanitize Data sent to other systems:** Sanitize all data sent to complex subsystems

Coding Standard Practices: Testable Programming

Testable Coding: Developing computer software that is easy to test, find and fix bugs

Some practices are:

1. **Assertions:** To identify out-of-range values wherever possible
2. **Test Points:** Methods to set/retrieve current module status and variable contents
3. **Scaffolding:** Code to emulate a feature of the system that the object would call
4. **Test Harness:** Code written to drive objects/modules/components as a part of the complete system
5. **Test Stubs:** Code to return known, fixed values to emulate functions that are not being tested
6. **Instrumenting:** Execution logging and messages to visualize execution
7. **Building test data sets:** For both valid and invalid data

Coding Guidelines

- Provides a uniform appearance to code written by different engineers
- Generic suggestions that improves understanding and readability of code
- Reduces cost incurred by early detection of errors
- Reduces complexity and eases maintenance over lifetime of the product

Examples

1. Code should be well Documented
2. Indent code with spaces and lines
3. Minimize the length of functions
4. Reduce the usage of GOTO statements
5. Avoid using identifiers for multiple purposes

THANK YOU

Dr. Jayashree R

Department of Computer Science and Engineering

jayashree@pes.edu

Software Engineering
Software Implementation, Configuration Management, and Quality

Dr. Jayashree R
Department of Computer Science and Engineering

Teaching Assistant: Aanchal Narendran (Sem VII)

Acknowledgement: Dr. Phalachandra HL, authors of the prescribed textbooks and materials sourced online.

Software Engineering

Managing Construction and Tools

Dr. Jayashree R

Department of Computer Science and Engineering

Managing Construction

- Key Issues critical to integrity and functionality of software solution
  - Minimizing complexity
  - Anticipating change
  - Verifiable software solutions
  - Constructing software using standards

- Two Perspectives to Managing construction

Proceeding as Planned?	Technical Quality?
Development progress and Productivity	Ease in debugging, maintaining, extending, etc
Measures: Effort expenditure, Rate of Completion, Productivity	**Measures:** Lines of Code, Number of defects found, Code Complexity
Metrics: LoC/effort days, LoC generated	**Metrics:** No of Errors/KLoC
Adjust plans based on milestones beaten, met or missed	Adjust the construction plans or processes

Managing Construction: Quality for Agile Scrum Project

Sprint Burndown

- Graphically communicated key metric representing completion of work throughout sprint based on story points
- Goal of the team is to consistently deliver all work

Team Velocity Metric

- "Amount" of software or stories completed during a sprint
- Can be measured in story points or hours
- Used for estimation and planning

Throughput

- Indicates total value-added work output by the team
- Represented by units of work completed in period of time

Cycle Time

Total time that elapses from the moment work is started on an item till it's completion

Managing Construction: Construction Technical Quality

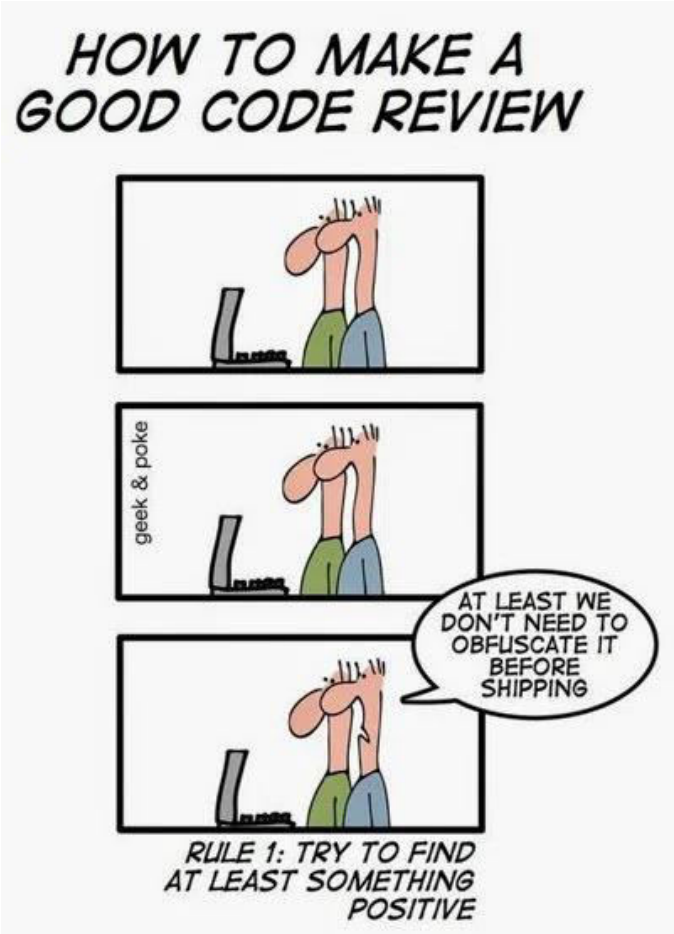
Activities for ensuring construction quality:

1. **Peer Review:** Common and Informal process of developer seeking advice or comments from other developers
2. **Unit Testing:** Writing code that calls individual modules/functions and tests their working
3. **Test-first:** Designing and building test harness before code. (Potential reason for doing so?)
4. **Code Stepping:** Code is executed one statement at a time and the values of variables and flow of control is analyzed
5. **Pair Programming:** Two developers work on same code with one focusing on logic and other on syntax
6. **Debugging:** Analyzing code to locate a known defect
7. **Code Inspections:** formal process of peer review
8. **Static Analysis:** examination of code without execution

Code Review

Code Review/Peer Code Review: consciously and systematically convening with fellow programmers to check each other's code for mistakes

- **What to review for?**
  - Correctness
  - Error Handling
  - Readability
  - Coding standards/guidelines
  - Optimization
- Use Review Checklists



Code Inspection

Software Inspection: examining the source representation with the aim of discovering anomalies and defects

- Check conformance with a specification but not with customer's real requirements
- Formalized approach to code/document reviews

- **Preconditions**
  - Precise specification must be available
  - Team members must be familiar with organization standards
  - Syntactically correct code or other representations must be available
  - Error checklist must be prepared

Code Inspection

Inspection Procedure	Inspection Checklist
Planning: selecting team, location and having code/documents	Checklist of common errors used to drive the inspection
System overview is presented to inspection team	Programming/language dependent
Code and associated documents are distributed	“weaker” type checking, larger the check list Eg: Initialization, loop termination, array bounds, etc
Prepare such that ever inspector inspects the item	
During inspection logging meeting, inspection takes places and discovered errors are noted	
Owners identify and make necessary modifications	
Re-inspection may/may not be required	

Unit testing tools

Unit Testing Frameworks

- Allow the tester to enter method name, parameters and expected results
- Framework supplies method call, return value comparison and error reporting
- No need to write test harness
- **Examples:** NUnit (for .NET), JUnit (for Java)

Code Coverage Analyzers/Debuggers

- Code Coverage helps identify code that is not covered by test cases
- **Examples:** CoCo (for C), JaCoCo (for Java)
- Debuggers help find bugs
- **Examples:** GDB

Record-Playback Tools

- Lets the tester start a session and records every keystroke and mouse click for a replay
- Done accurately and Quickly
- **Example:** Selenium

Wizards

Tools that generate tests from input parameters

THANK YOU

Dr. Jayashree R
Department of Computer Science and Engineering

jayashree@pes.edu

Software Engineering

Software Implementation, Configuration Management, and Quality

Dr. Jayashree R
Department of Computer Science and Engineering

Teaching Assistant: Aanchal Narendran (Sem VII)

Acknowledgement: Dr. Phalachandra HL, authors of the prescribed textbooks and materials sourced online.

Software Engineering

Software Configuration Management

Dr. Jayashree R

Department of Computer Science and Engineering

Introduction

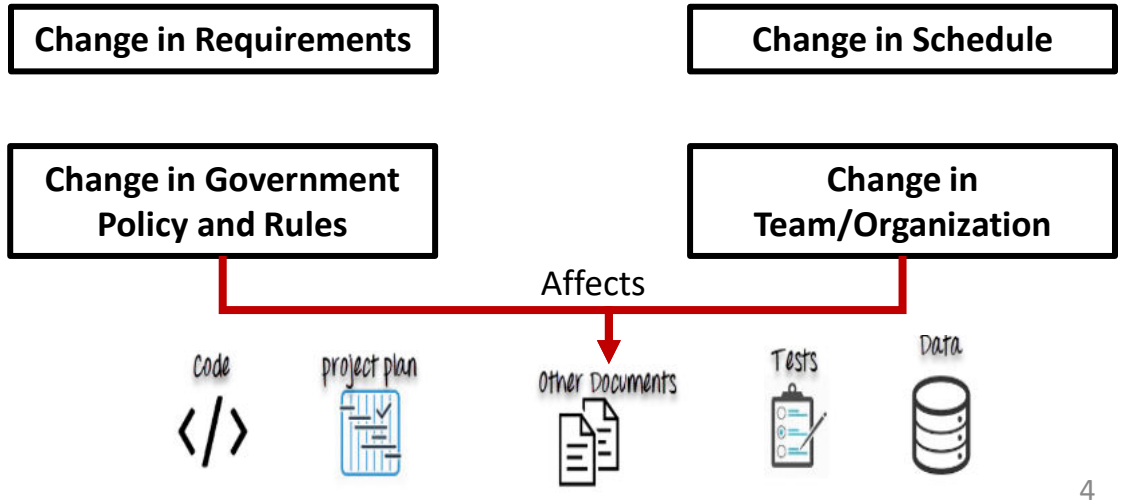
Software Configuration Management: Process to systematically organize, manage and control changes in documents, code and other entities that constitute a software product

- **Goal:** Increase productivity by increased and planned coordination among the programmers and eliminates confusion and mistakes
  - Identifying elements and configurations, tracking changes, version selection, control and baselining
  - Avoid configuration related problems
  - Effective management of simultaneous updating of source files
  - Build management (*Build and release management is the process of managing, planning, scheduling, and controlling a software build throughout its lifecycle.*)(Building an application or software requires build tools like ant, maven, gradle, etc. Build tools compile the source code files into reusable executable files or packages)
  - Defect tracking

Need for SCM

Software engineering in large products/projects with multiple people across locations need to be reliably managed for

- Multiple people working concurrently on the same piece of software
- Working on more than one version
- Working on released systems
- Changes in configuration due to changes in user requirements, budget, etc
- Custom configured systems
- Software must run on different machines
- Coordination among stakeholders
- Controlling costs in making changes



SCM in Scrum-Agile Approach

- SCM is the responsibility of the “whole team” and is automated as much as possible
- The definitive versions of components are held in a shared project repository
- Developers copy versions from the repository into their workspace, make changes to the code and use system-building tools to create a new system on their computer
- Once the changes are tested, the modified components are pushed to the project repository
- Versions of the modified code is available to all team members

Benefits of SCM

1. Permits orderly development of software configuration items
2. Ensures the orderly release and implementation of new or revised software products
3. Ensures only approved changes to both new and existing software products are implemented and deployed
4. Ensures that software changes are implemented in accordance with approved specifications
5. Ensure that the documentation accurately reflects updates
6. Evaluates and communicates the impact of changes
7. Prevents unauthorized changes from being made

Configuration Management Roles

Configuration Manager

- Identify configuration items
- Define procedures for creating and promotions and releases

Developer

- Creates versions triggered by change requests or normal development activities
- Checks in changes and resolves conflicts

Auditor

- Validate processes for selection and evaluation of promotions for release
- Ensures consistency and completeness

Change Control Board Member

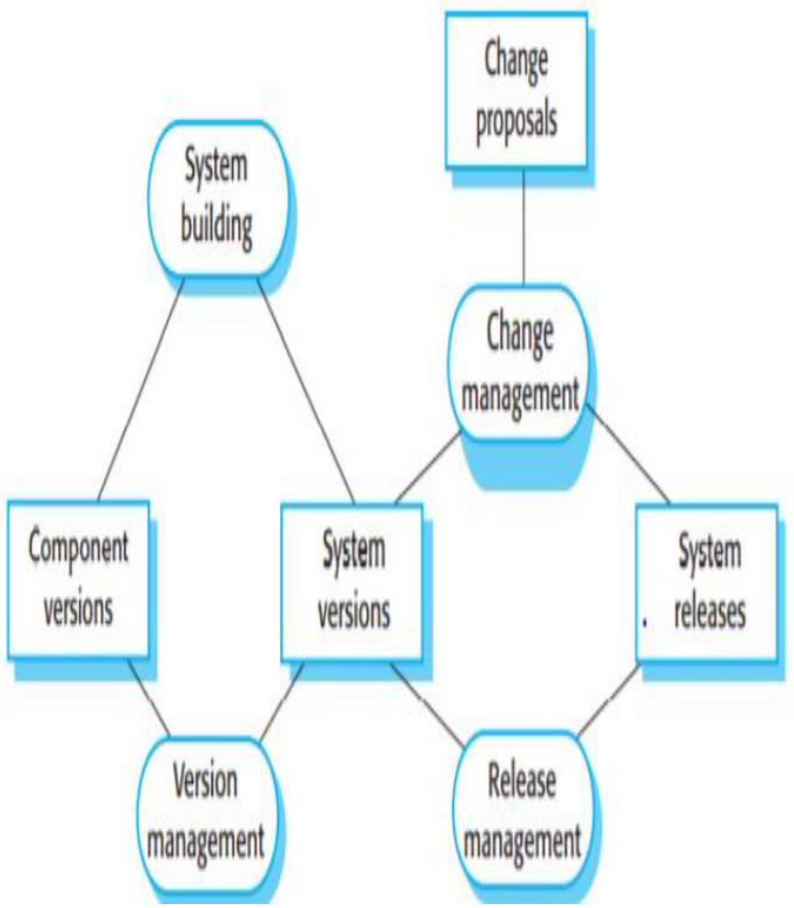
- Responsible for approving or rejecting change requests

Software Configuration Management Planning

- Planning is done by a Configuration manager and starts during the early phases of the project
- Results in a Software Configuration Management Plan
  - May follow a public or internal standard
  - Defines the Configuration Items and a naming scheme
  - Define who takes responsibility for Configuration Management procedures and creation of baselines
  - Defines policies for change control and version management
  - Describes tools to assist in Configuration Management and any limitations
  - Defines the configuration management database

Configuration Management Activities

1. Configuration item Identification
2. Configuration Management Directories
3. Baselining
4. Branch Management
5. Version Management
6. Build Management
7. Install
8. Promotion Management
9. Change Management
10. Release Management
11. Defect Management



Configuration Items

Configuration item: Independent or aggregation of hardware, software or both, that is designated for configuration management and treated as a single entity

- Software configuration items could be
  - All types of code files and Drivers for tests
  - Requirement, analysis, design, test and other non temporary documents
  - User or developer manuals
  - System configurations
- Challenges associated with configuration items
  - What items need Configuration control?
    - Some items must be maintained for the lifetime of the software. Similar to object modelling
  - When to place entities under configuration control
    - Start too early, too much bureaucracy
    - Start too late, introduces chaos

SCM Directories

Programmer's Directory
(Dynamic Library)

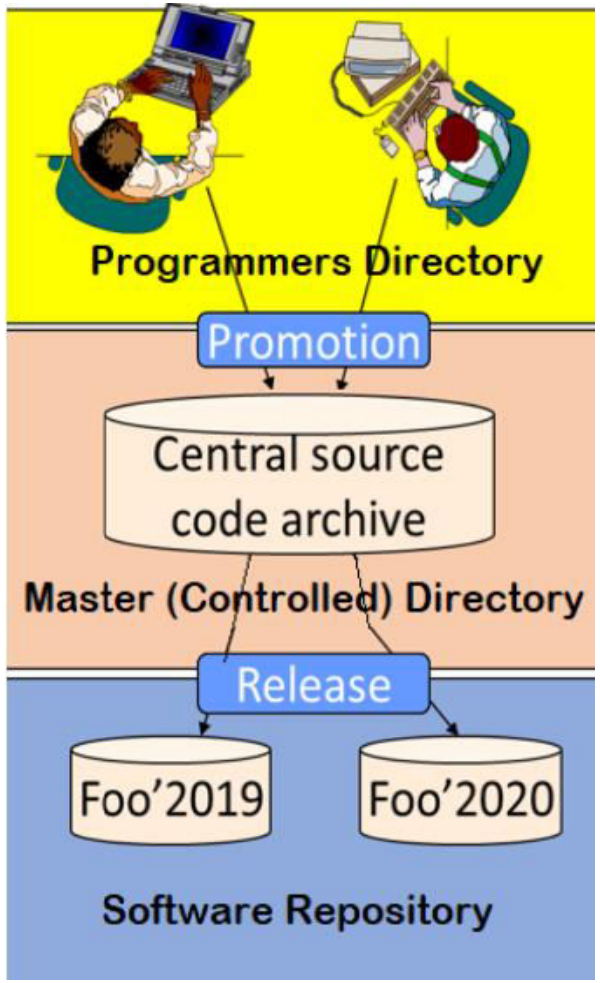
- Library for holding newly created or modified software entities
- Controlled by the programmer

Software Repository
(Static Library)

- Archive for various baselines in general use
- Copies may be made available on request

Master Directory
(Controlled Library)

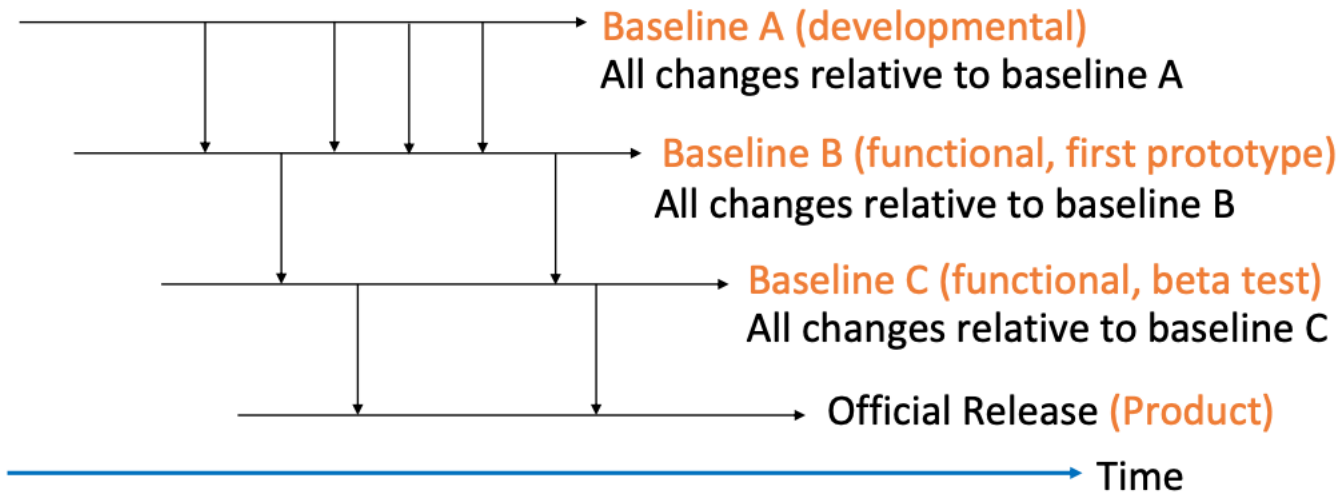
- Manages current baselines and for controlling changes
- Entry is controlled, after verification
- Changes must be authorized



Baselines

Baseline: Specification or product that has been formally reviewed and agreed to by responsible management and serves as a basis and can only be changed on formal review

- As systems get developed, baselines get developed after a review
  - **Examples:**
    - **Baseline A:** APIs have been defined; Bodies of methods are empty
    - **Baseline B:** All data access methods are implemented and tested



Food for thought: Do baselines have to be tied to project milestones?

THANK YOU

Dr. Jayashree R

Department of Computer Science and Engineering

jayashree@pes.edu

Software Engineering

Software Implementation, Configuration Management, and Quality

Dr. Jayashree R

Department of Computer Science and Engineering

Teaching Assistant: Aanchal Narendran (Sem VII)

Acknowledgement: Dr. Phalachandra HL, authors of the prescribed textbooks and materials sourced online.

Software Engineering

Software Configuration Management Activities

Dr. Jayashree R

Department of Computer Science and Engineering

Branch Management

Codeline: Progression of source file and artifact which make up software components as they change over time

Branch: copy or clone of all or a portion of the source code

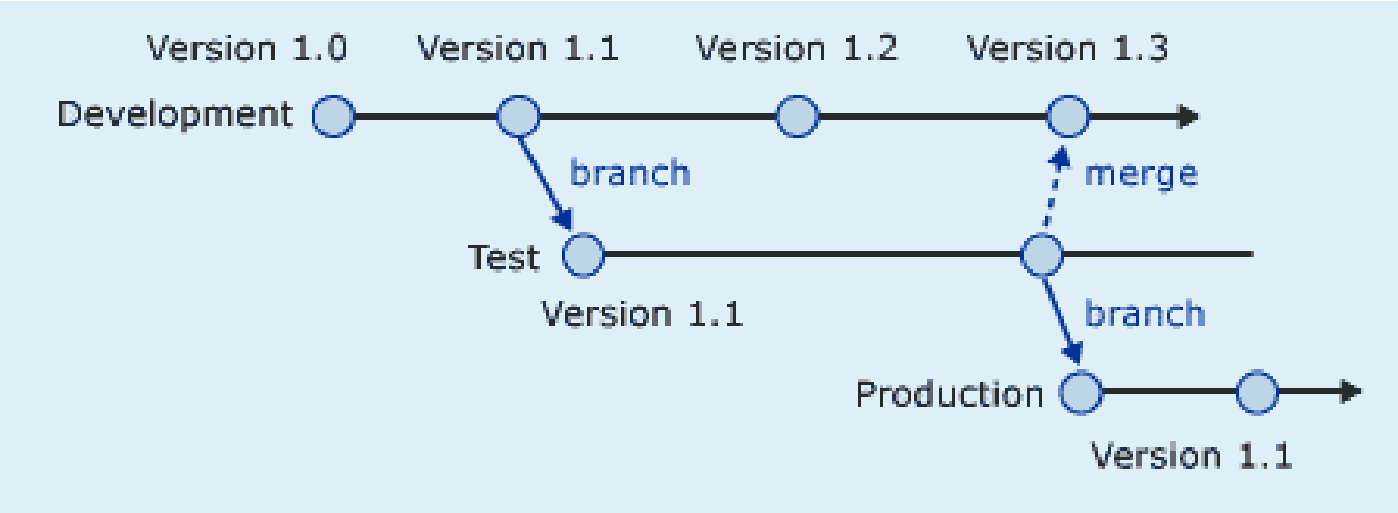
Reasons for Branching

- Supporting concurrent development
- Capturing of solution configurations
- Support multiple versions of a solution
- Enable experimentation in isolation without impacting the stable version

- Branching ensures that overall product is stable
- Merging is brining back and integrating changes over multiple branches
  - Frequent merging helps decrease the likelihood and complexity of a merge conflict

Branch Management

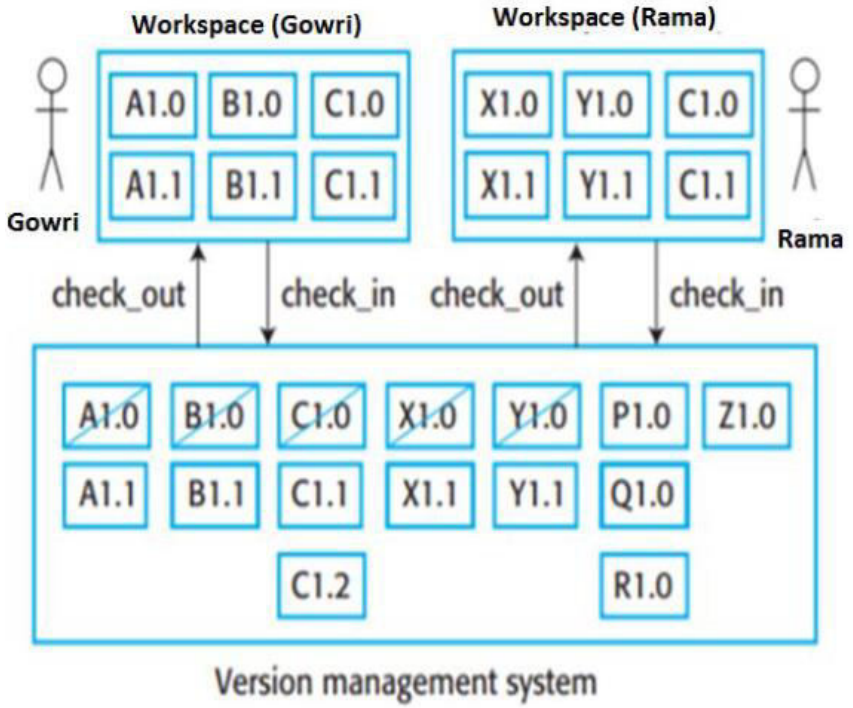
- Many branching strategies which may be applied in combination
  - Single branch
  - Branch by customer or organization
  - Branch by developer or workspace
  - Branch by module or component
- Branch Management entails having a well defined branching policy, owner for the code lines and usage of branches for release



Version Management

Version Management: keeping track of different versions of software components and systems

- Usage of tools like git for version management
- Changes to a version are identified by a number, termed the revision number (7.5.5)
  - 7 – Release number (defined by customer)
  - 5 – Version number (defined by developer)
  - 5 – Revision number (defined by developer)
- Key Features of Version Control System
  - Changes are attributable or traceable
  - Change history is recorded and can be reverted
  - Better conflict resolution
  - Easier code maintenance and quality monitoring
  - Less software regression
  - Better organization and communication



Build Management

Build Management: creating the application program for a software release by compiling and linking source code and libraries to build artefacts such as binaries or executables

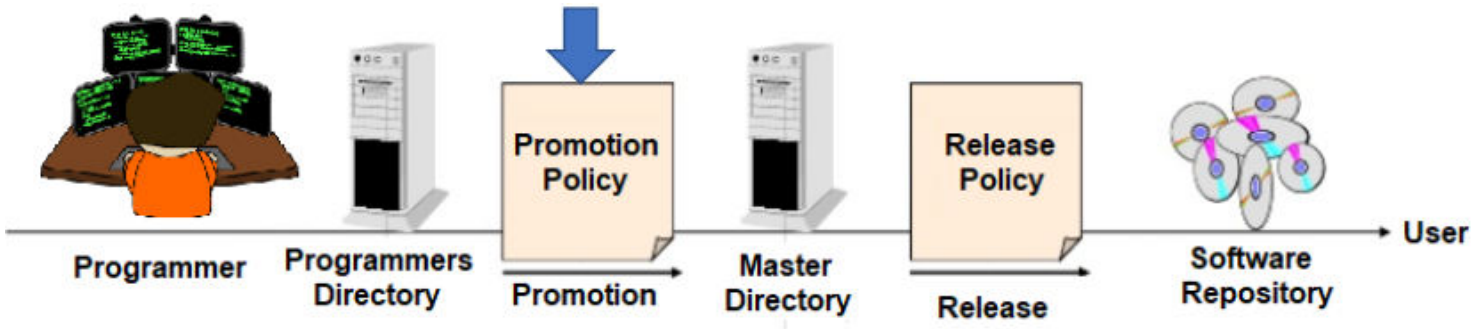
- Done using tools like Make, Apache Ant, Maven, etc
- Compilation and linking of files in the correct order
    - No need to recompile if no change in source code results in shorter build time
- Build Process
    - Fetching code from the source control repository
    - Compiling the code and checking the dependencies
    - Linking the libraries, code, etc
    - Running tests and building artefacts
    - Archive logs and send notification emails
    - May result in version number change

Install Management

- Software installation is the first interaction with the customer
  - If install fails, may result in negative perception
- **Software installation:** placing multiple files containing executable code, downloading or copying from a repository, images, libraries, configuration files from the internet
- Interaction with OS functions for validating the resources needed, permissions, versions, identifiers to ensure enforcement of licenses
- May involve customizations for localizations
- Sometimes may be automated using zip, shell scripts, InstallAware and Jenkins

Promotion Management

- Changes made by a programmer is only available in their environment and needs to be promoted to a central master directory
  - Promotion is done based on certain promotion policies
- Promotion could be based on baselining criteria which was planned for and would involve some amount of verification
- It would further be authorized and moved to the master directory
- **Food for thought:** Could compiled and executable code with a few warnings (not errors) be promoted to the master directory?



THANK YOU

Dr. Jayashree R
Department of Computer Science and Engineering

jayashree@pes.edu

Software Engineering
Software Implementation, Configuration Management, and Quality

Dr. Jayashree R
Department of Computer Science and Engineering

Teaching Assistant: Aanchal Narendran (Sem VII)

Acknowledgement: Dr. Phalachandra HL, authors of the prescribed textbooks and materials sourced online.

Software Engineering

Software Configuration Management Activities

Dr. Jayashree R

Department of Computer Science and Engineering

Change Management

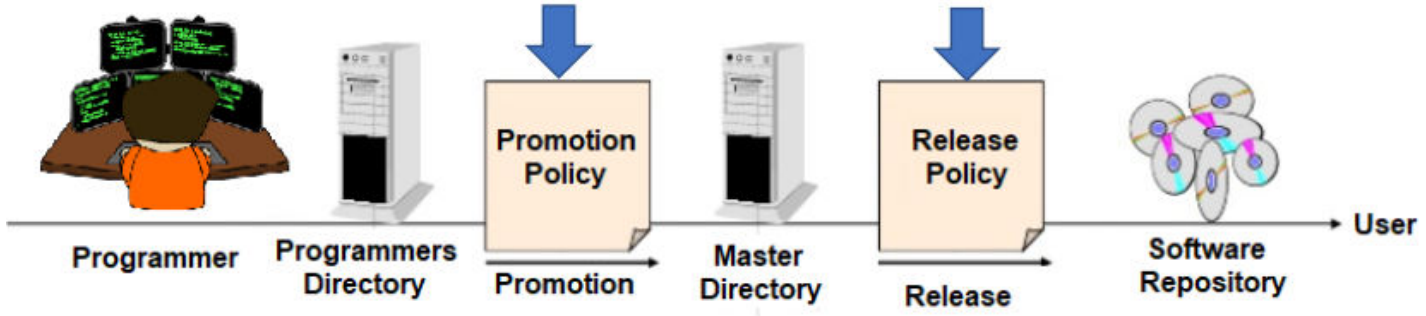
- Change could result in creation of different version or release of the software
- Deals with changes in Configuration Items which have been baselined
- General change process
  - Change can be requested (anytime by anyone)
  - Unique identification is associated with the requested change and is logged
  - Change is assessed based on impact to other modules, branches and categorized
  - Decision on the change – Accepted or Rejected
  - All of these activities are mostly tool driven
  - If accepted, change is implemented and validated
  - Plans done and executed for documentation, versioning, merging and delivery
  - Implemented change is audited

Change Management

- Complexity of change management varies with the project
  - Small projects perform change requests informally and fast
  - Complex projects require detailed change request forms and the official approval of managers or the Configuration Control Board (CCB)
- Information required to process a change to a baseline
  - Description of proposed changes
  - Reasons for making the changes
  - List of other items affected by the changes
- Tools, resources and training are required to perform baseline change assessment
  - File comparison tools to identify changes
  - Other resources and training depending on size and complexity of project

Controlling changes

- Changes are controlled at two points
  - Promotion Policy
  - Release Policy



- **Change Policies:** the promotion and release policy is dictated via environment
  - Informal: good for research type environments and promotions
  - Formal: good for externally developed configuration item and for releases

- Change policies guarantee that each version, revision or release conforms to commonly accepted criteria and a consistent process is applied to how things are done
  - Enforced through engineering processes and tools
  - Audited to ensure conformation

Release Management

- Movement of code to the customer and the software repository is done via release policies
- **Release Policy:** Gating quality criteria that is planned for and includes verification with metrics
  - On meeting metrics, it would be authorized for a release
- **Release Management:** managing, planning, scheduling and controlling a software build through different stages and environments, testing and deploying software releases
- **Version vs Release vs Revision**
  - **Release:** formal distribution of an approved version
  - **Version:** Initial release or re-release of a configuration item associated with a complete compilation or recompilation of the item
    - Different versions have different functionality
  - **Revision:** change to a version that corrects only errors in the design/code, but does not affect the documented functionality

Bug or Defect Management

Bug: Consequence of a coding fault
Defect: variation or deviation from expected business requirements
Driven through defect management tools like Bugzilla

Defect Management Process

1. Discover
2. Reporting or logging into the tool with an unique identifier
3. Validation
4. Analysis and Categorization (Critical, High, Medium, and Low)
5. Request and Approval for fix
6. Resolution
7. Verification by submitter
8. Merging of the code
9. Updating version number
10. Plan for release of fix to customer
11. Closure
12. Reporting

THANK YOU

Dr. Jayashree R

Department of Computer Science and Engineering

jayashree@pes.edu

Software Engineering

Software Implementation, Configuration Management, and Quality

Dr. Jayashree R

Department of Computer Science and Engineering

Teaching Assistant: Aanchal Narendran (Sem VII)

Acknowledgement: Dr. Phalachandra HL, authors of the prescribed textbooks and materials sourced online.

PES UNIVERSITY ONLINE

Software Engineering

Software Configuration Management Tools

Dr. Jayashree R

Department of Computer Science and Engineering

Introduction

Types of Software Configuration Management Tools

Source Code Administration

Used for Source Code control

Software Build

Used for building Source Code

Software Installation

Used for Packaging and Installing the products

Software Bug Tracking

Used for tracking bugs and changes

Source Code Administration tools

RCS

- Very old but in use
- Used for version control system

CVS (Concurrent Version Control)

- Based on RCS, allows concurrent working without locking
- Has a Web Frontend

ClearCase

- Multiple servers, process modeling, policy check mechanisms

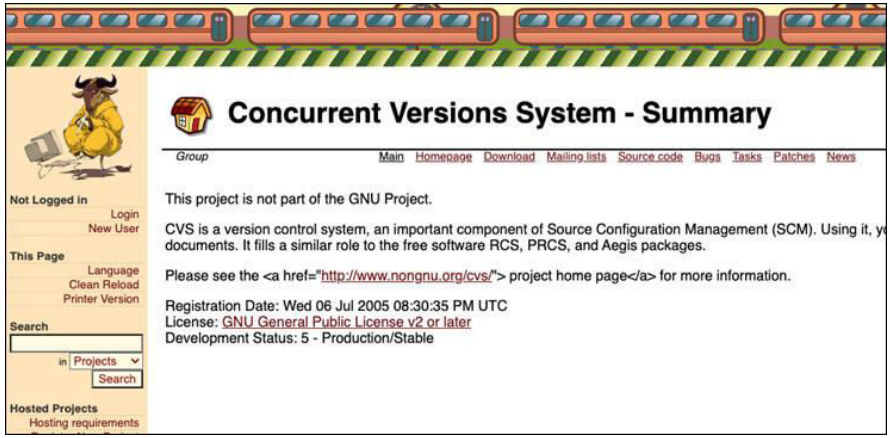
GitHub

- Development platform for version control and project management

RCS is **a software tool for UNIX systems which lets people working on the system control "multiple revisions of text ... that is revised frequently, such as programs or documentation."**

CVS

Best for teams of any size that are looking for basic versioning needs. However, since it's quite old, teams now prefer modern versioning solutions like Git.



Features:

•CVS stands for Concurrent Version Control Systems.

•Simple and sophisticated version control capabilities.

Pros:

•Basic version control needs.

•Conflict resolution is straightforward.

Cons:

•It's slightly outdated as it was one of the first available version control systems, and most modern solutions provide all these features.

•Merging with multiple commits can sometimes be painful.

Pricing:

•Open-source and free to use.

ClearCase is **a software configuration management tool used for version control**. It manages changes across development lifecycles, from design to code to test. It is one of many version control systems available today. (Other options include Perforce Helix Core, Subversion, Git, and TFS.)

GITHUB

Features:

•Enables large development teams to collaborate, review, and manage software/application code.
•Provides integration with Bug tracking tools.
•Built-in support for access control and permissions.

Pros:

•Supports Windows, Mac, and mobile devices.
•Supports cloud-based deployment.
•Can manage multiple projects and permissions at a granular level.
•Great documentation and support.

Cons:

•Security for high-value intellectual property projects/codes can be a concern, as Github is available as a cloud-based setup.
•Reviewing large pull requests can be tedious and time-consuming.

Pricing:

•Free trial is available with limited support for individuals and organizations.
•Other options are the Team version at $4/user/month and Enterprise at $21/user/month.

GitHub Overview

- Can create repositories and add contributors
  - Access the repository using ssh or https
- Changes are pushed as commits
  - Commits are logged with a unique commit number and message
- Master branch is auto cloned
  - Different branches can be created to add code without impacting the stable version
- General contribution method
  - Fork a project
  - Make changes
  - Generate a pull request to merge new code (Are all pull requests merged?)
  - Whenever merged, a difference is done and changes are logged

Some interesting git commands

git init #initialize repo	**git status** #shows untracked files, commits	**git add <files>** #brings untracked files to git	**git commit –m "message"** #commits with message

git remote -v #list of remote origins of your local repository	**git push** #push changes onto remote repository

Git Init command

This command is used to create a local repository.

Syntax

$ git init Demo

The init command will initialize an empty repository.

Git add command

This command is used to add one or more files .

To add one file

$ git add Filename

To add more than one file

git add*

Git commit -m

It records or snapshots the file permanently in the version history with a message.

Git status command

The status command is used to display the state of the working directory and the staging area. It allows you to see which changes have been staged, which haven't, and which files aren’t being tracked by Git.

Git push Command

It is used to upload local repository content to a remote repository. Pushing is an act of transfer commits from your local repository to a remote repo. It's the complement to git fetch, but whereas fetching imports commits to local branches on comparatively pushing exports commits to remote branches. Remote branches are configured by using the git remote command. Pushing is capable of overwriting changes, and caution should be taken when pushing.

Git remote Command

Git Remote command is used to connect your local repository to the remote server. This command allows you to create, view, and delete connections to other repositories. These connections are more like bookmarks rather than direct links into other repositories. This command doesn't provide real-time access to repositories.

Software Build

Software build: process of converting source code into standalone software artefact(s)

- Has a compilation process where build is converted to executable

Make

- Automatically builds executable programs using Makefiles
- Makefiles derive target program

CruiseControl

- Open source tool for continuous software builds

FinalBuilder

- Automate build and release management tool
- Easily define and maintain reliable build process

Maven

- Software project management and comprehension tool
- Based on project object model

CruiseControl is both a continuous integration tool and an extensible framework for creating a custom continuous build process. It includes dozens of plugins for a variety of source controls, build technologies, and notifications schemes including email and instant messaging. A web interface provides details of the current and previous builds. And the standard CruiseControl distribution is augmented through a rich selection of 3rd Party Tools.
CruiseControl is written in Java but is used on a wide variety of projects. There are builders supplied for Ant, NAnt, Maven, Phing, Rake, and Xcode, and the catch-all exec builder that can be used with any command-line tool or script.

Integrated Debugging

A debugging engine is fully integrated into the FinalBuilder IDE. You can step between actions, set break points and set variable watches to debug your build process.

Detailed Logging

Output from all actions in the build process is directed to the build log. The log maintains the same tree structure as the action lists, which makes it easy to navigate the log file. Logs can be exported in HTML, XML or text formats.

Email, MSN, FTP & SFTP Support

Your build process can email the results of the build to you. For example, if the build fails it can export the log, attach it to an email and send it to you. You can announce successful builds on NNTP news servers.

Variables

Using variables is the key to making your build process dynamic. Use variables to pass information from one action to another, and enable actions to build with user specific or machine specific values.

Version Control Integration

FinalBuilder supports more than a dozen version control systems out-of-the-box, so it can get, checkout, tag and perform other build related operations with your existing revision control system.

Scripting Support

Each FinalBuilder action exposes a number of script events, where you have access to all the properties of the action using VBScript, JavaScript, Powershell or Iron Python. In addition, the "Run Script" action allows you to perform specific tasks which might not be covered by the built in action types.

Action Studio

Action Studio is an IDE for writing custom FinalBuilder actions. It provides a way to extend the built-in functionality in FinalBuilder. Action Studio is included in FinalBuilder and can be accessed under the "Tools" menu.

All FinalBuilder Features

For a list of all the features in FinalBuilder, and to compare the Standard and Professional editions, see the Feature Matrix.

Software Build Tools: Makefile

Typical Makefile

Make is Unix utility that is designed to start execution of a makefile. A makefile is a special file, containing shell commands, that you create and name makefile

make could be used to detect a change made to an image file (the source) and the transformation actions might be to convert the file to some specific format, copy the result into a content management system, and then send e-mail to a predefined set of users that the above actions were performed.

```
CC = gcc  #indicates the compiler being used
LDFLAGS =  #to inform the compiler to include any linkers
all: helloworld
helloworld: helloworld.o
        $(CC)$(LDFLAGS) -o $@ $^
        #@ is used to indicate current target (helloworld)
        #Question: what does the -o flag indicate?
```

Refer:https://www.computerhope.com/unix/umake.htm

"**$<**" refers to the first prerequisite and **$^** expands to a space-delimited list of the prerequisites.

```
helloworld.o: helloworld.c
        $(CC)$(LDFLAGS) –c -o $@ $<
```

clean: FRC rm –f helloworld helloworld.o

#Question: Is this command necessary? What is its role

This pseudo target causes all targets that depend on FRC # to be remade even in case a file with the name of the target exists. # This works with any make implementation under the assumption that # there is no file FRC in the current directory. FRC:

Maven

- Simplifies and standardizes the project build process
- Handles compilation, distribution, documentation, team collaboration and other tasks
- Increases reusability and takes care of most build tasks
- Supports multiple development team environments
- Supports creation of reports, checks, build and testing automation
- Standard directory layout and default build lifecycles with environment variables

source code	${basedir}/src/main/java
Resources	${basedir}/src/main/resources
Tests	${basedir}/src/test
Complied byte code	${basedir}/target
distributable JAR	${basedir}/target/classes

Software Installation tools

- Cross platform tools that produce installers for multiple Operating systems
  - **Installer:** installs all the necessary files on the system
    - Could be customized to meet specific needs
- **Bootstrapper:** small installer that does the pre-requisites and updates the big bundle

DeployMaster (Windows)

- Distribute windows software or files via internet/CD/DVD
- Works with different versions of windows

InstallShield

- Simplifies creation of windows installers, MSI packages, and InstallScript installers for Windows
- De-facto for MSI installations

InstallAware (Windows)

- Windows installer platform for MS windows OS
- Supports internet deployment

Wise Installer

- Configure and install Microsoft windows applications

THANK YOU

Dr. Jayashree R
Department of Computer Science and Engineering

jayashree@pes.edu

Software Engineering
Software Implementation, Configuration Management, and Quality

Dr. Jayashree R
Department of Computer Science and Engineering

Teaching Assistant: Aanchal Narendran (Sem VII)

Acknowledgement: Dr. Phalachandra HL, authors of the prescribed textbooks and materials sourced online.

Software Engineering

Software Quality

Dr. Jayashree R

Department of Computer Science and Engineering

Software Quality

- Software systems are complex and evolve leading to need for continuous assessment and evaluation of quality
- Bad quality of software could impact experiences in life
  - Can also lead to dissatisfied customers
- Software quality enhances long term profitability of products

Perspectives of Quality

Transcendent Exceeded normal expectations	User-Based Fitness for use	Manufacturing based Conformance to specs
Product-based Based on attributes of the software	Balancing time, cost and profits Value-Based	

Software product quality perspectives

Product Operation Perspective

- **Correctness:** Does it do what I want?
- **Reliability:** Is it always accurate?
- **Efficiency:** Does it run as well as it can?
- **Integrity:** Is it secure?
- **Usability:** Can I use it?
- **Functionality:** Does it have necessary features?
- **Availability:** Will the product always run when needed?

Overall Environment Perspective

- **Responsiveness:** Can I quickly respond to change
- **Predictability:** Can I always predict the progress?
- **Productivity:** Will things be done efficiently?
- **People:** Will the customers be satisfied?
  - Will the employees be gainfully engaged?

Software product quality perspectives (contd.)

Product Revision Perspective

- **Maintainability:** Can I fix it?
- **Testability:** Can I test it?
- **Flexibility:** Can I change it?

Product Transition Perspective

- **Portability:** Can it be used on another machine?
- **Reusability:** Can I reuse some/all of the software?
- **Interoperability:** Can I interface it with another system?

Approaches to look at quality

- Quality of software product can be visualized as **“Quality of the Product”** vs **“Quality of the Process”**
- Quality actions could include checking whether it conforms to certain norms
  - Verification, Validation and Audits
- Improvement in quality could be achieved by improving the product or process
- **Quality Attributes (FLURPS+)**
  - **Functionality:** features of system
  - **Localization:** Localizable to local language
  - **Usability**: Intuitive, documentation
  - **Reliability**: Frequency of failure in intended time
  - **Performance:** Speed, throughput, resource consumption
  - **Supportability:** Serviceability, maintainability
  - The **+** could include Extensibility and so on

Terminologies associated with software quality

- **Attribute:** measurable physical or abstract property of an entity
- **Measure:** Quantitative indication of extent, amount, dimension, capacity or size
  - Eg: Number of errors
- **Metric:** Quantitative measure of degree to which a system possesses a given attribute
  - Calculation between two measures
  - Eg: number of errors found per person hours
- **Why measure?**
  - **Feedback:** quantifies some of the attribute to help improve the product
  - **Diagnostics:** helps identify issues towards quality
    - Supports evaluating and establishing productivity
  - **Forecasting:** to predict future need and anticipate maintenance
    - Supports estimating, budgeting, costing and scheduling

Characteristics of Software Measures and Metrics

- **Quantitative:** Metrics should be quantitative and expressible in values
- **Understandable:** Metric computation should be defined and easily understood
- **Applicability:** Should be applicable at all stages of software development
- **Repeatable:** Metrics are consistent and same when measured again
- **Economical:** Computation of metric should be economical
- **Language Independent:** Metrics should not depend on programming language

Examples of Measures

Correctness Defects/KLoC, Failures/Hours of operation	Maintainability Mean time to change, Cost to correct
Integrity Fault tolerance, security and threats	Usability Training time, skill level, productivity

Cost of Software Quality

- Measure of quantifying and calculating business value of quality activities
- Costs incurred through meeting and not meeting customer quality
- Isolates the state of quality in the company and helps eliminate waste due to poor quality

Cost of Good Quality	Cost of Bad Quality
Prevention costs: Investments to prevent/avoid quality problems E.g.: Error proofing, improvement initiatives	**Internal failure costs:** costs associated with defects found before the customer receives the product E.g.: Rework, Re-testing
Appraisal costs: costs to determine degree of conformance to requirements and quality standards E.g.: Quality Assurance, Inspection	**External failure costs:** costs associated with defects found after customer receives product E.g.: Support Calls, Patches
Management Control costs: costs to prevent or reduce failures in management functions E.g.: contract reviews, gating/release criteria	**Technical debt:** cost of fixing a problem, which left unfixed, puts the business at risk E.g.: Structural problems, Increased Complexity
	Management failures: costs incurred by personnel due to poor quality software Eg: Unplanned costs, customer damages

THANK YOU

Dr. Jayashree R
Department of Computer Science and Engineering

jayashree@pes.edu

Software Engineering
Software Implementation, Configuration Management, and Quality

Dr. Jayashree R
Department of Computer Science and Engineering

Teaching Assistant: Aanchal Narendran (Sem VII)

Acknowledgement: Dr. Phalachandra HL, authors of the prescribed textbooks and materials sourced online.

Software Engineering

Software Quality: Metrics, SQA, CMM

Dr. Jayashree R

Department of Computer Science and Engineering

Software Metrics Categorization

Direct Measures
(internal attributes)

- Depends only on value
- Other attributes are measured with respect to these
- E.g.: Cost, Effort, LoC, Duration of testing

Indirect Measures
(External Attributes)

- Derived from direct measures
- E.g.: Defect density, productivity

Size Oriented
(size of software in LoC)

E.g.: Errors/KLoC, Cost/LoC

Complexity Oriented
(LoC – function of Complexity)

- Fan-In, Fan-out
- Halstead's software science (entropy measures)
- Program length, volume, vocabulary

Software Metrics Categorization (contd)

Product Metrics

- Assessing the state of the project
- Tracking potential risks
- Uncovering problem areas
- Adjusting workflow or tasks
- Evaluating teams ability to control quality

Project Metrics

- Number of software developers
- Staffing pattern over the lifecycle of software
- Cost
- Schedule
- Productivity

Process Metrics

- Insights of software engineering tasks, work product or milestones?
- Long term process improvements

Usage of Metrics

- Understand the environment/product
- Formulate and/or select appropriate metrics for a problem
- Educate
- Collect
- Analyze
- Interpret
- Course-Correction
- Feedback

Food for thought: Why are we concerned with analyzing and interpreting metrics?

Software Quality Assurance

Software Quality Assurance: methods to monitor software engineering process to ensure quality

- Involves planning (setting up of goals, commitments, activities, measurements and verifications)
- Encompasses
  - Entire software development processes and activities
  - Planning oversight, record keeping, analysis and reporting
  - Auditing designated software work to verify compliance
  - Ensuring deviations from documented procedure are recorded and noncompliance is reported

Who is involved in SQA?

Project Managers
- Establish processes and procedures
- Plan and provide oversight

Software Engineers
- Apply technical methods and measures
- Conduct review and testing

SQA Group
- QS planning oversight, record keeping, analysis and reporting
- Customers in-house representative

All Stakeholders
- Perform actions relevant to quality of product

Contents of the SQA plan

- Quality cannot be plugged in and needs to happen throughout the lifecycle
- Addresses the following
  - Responsibility Management
  - Document Management and Control
  - Requirements Scope
  - Design Control
  - Development Control and Rigor
  - Testing and Quality Assurance
  - Risks and Mitigation
  - Quality Audits
  - Defect Management
  - Training Requirements

The SQA Plan [IEEE94]

Some basic items

- Purpose of plan and scope
- Management
  - Organization structure, SQA tasks and placement
  - Roles and Responsibilities
- Documentation
  - Project, technical and user documents, models
- Standards, practices and conventions
- Reviews and audits
- Test plan and procedure
- Problem reporting and corrective measures
- Tools

- Code Control
- Media Control
- Supplier Control
- Records collection, Maintenance and Retention
- Training
- Risk Management

SEI - CMM

- Developed by Software Engineering Institute of Carnegie Mellon University
- Tool for objectively assessing the capability of vendor to deliver software
- Maturity model: set of structured levels that decide how well the behaviors, practices and processes of an organization can reliably and sustainably produce outcomes
- Evolutionary improvement path for software organization
- Benchmark for comparison of software development processes and an aid to understanding

Characterizing CMM Process terminologies

Process
- Activities, methods, practices and transformations to develop and maintain software

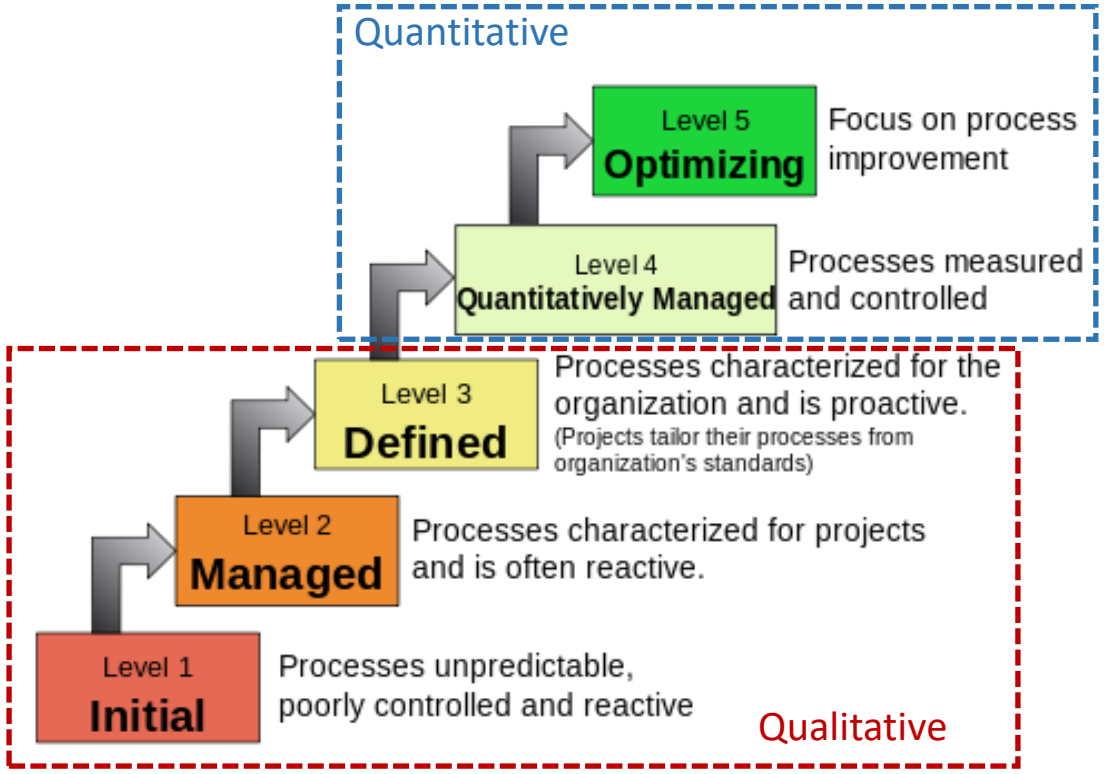
Process Capability
- Ability of process meet specifications
- Indicates range of expected results
- Predictor of future project outcomes

Process Performance
- Measure of results for a specific activity by following a process

Process Maturity
- Extent to which process is defined, managed, measured, controlled and effective

Aspects of the CMM Model

Maturity Levels: 5-level process of maturity continuum where 5th level is most ideal stage



Process maturity perspective

1. Initial (just do it)
2. Repeatable (focus on project management)
3. Well defined (organized assets)
4. Analyzed, improved and managed (quantitative control)
5. Improved and Optimized (continuously improving)

Organization maturity perspective

1. Work accomplished according to plan
2. Practices consistent with processes
3. Processes updated as necessary
4. Well-defined roles/responsibilities
5. Inter-group communication and coordination
6. Management formally commits

Benefits, Risks and Limitations

Benefits

- Establishes a common language and vision
- Build on set of processes and practices developed with input from software community
- Framework for prioritizing actions
- Framework for reliable and consistent appraisals
- Supports industry wide comparisons

Risks

- Models are a simplification of real-world
- Models are not comprehensive
- Interpretation and tailoring must be aligned to business objectives
- Judgement and insight to use correct model

Limitations

- No specific way to achieve the goals
- Helps if used early in the software development process
- Only concerned with improvement of management related activities

THANK YOU

Dr. Jayashree R

Department of Computer Science and Engineering

jayashree@pes.edu