Software Engineering
Unit - III

Compiled by
M S Anand

Department of Computer Science

Software Engineering
Introduction

Text Book(s):

1. “Software Engineering: Principles and Practice", Hans van Vliet, Wiley India, 3rd Edition, 2010.
2. “Software Testing – Principles and Practices”, Srinivasan Desikan and Gopalaswamy Ramesh, Pearson, 2006.

Reference Book(s):

1. “Effective DevOps: Building a Culture of Collaboration, Affinity, and Tooling” by By Jennifer Davis, Ryn Daniels, O’ Reilly Publications, 2018
2. “Software Engineering: A Practitioner’s Approach”, Roger S Pressman, McGraw Hill, 6th Edition 2005
3. “Software Engineering”, International Computer Science Series, Ian Somerville, Pearson Education, 9th Edition, 2009.
4. “Foundations of Software Testing ", Aditya Mathur, Pearson, 2008
5. “Software Testing, A Craftsman’s Approach ", Paul C. Jorgensen, Auerbach, 2008.
6. IEEE SWEBOK, PMBOK, BABOK and Other Sources from Internet.

Software Implementation (Development)

Design activity results in a design document.

This design document is the input to the development (programming) activity.

The output is executable software which is the input to the next phase – Software Testing.

Software Implementation (Development)

Before getting into the implementation (development) stage, the following should be looked into:

- Choice between compiled/interpreted language
- Decision on development environment
- A Configuration Management Plan

During the development (implementation) phase:

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

Software Implementation (Development)

Software development (implementation)
Creation of working software through a combination of coding, reviews and Unit Testing.

What are the goals during this phase?

- Minimize complexity
- Anticipate change
- Verifiable code
- Reuse
- Readable

Software Implementation (Development)

What are the main characteristics of this phase?

1. Produces High Volume of Configuration items
   Eg: Source Files, Test Cases.

2. Extensive usage of Computer Science Knowledge
   Eg: Algorithms, Coding Practices

3. Related to Software Quality
   Eg: Clean and Maintainable Code

4. Tool Intensive
   Eg: Compilers, Debuggers, GUI Builders

Choice of programming language

Questions to Ask When Choosing a Programming Language

1. Does the language have proper ecosystem support? Is it going to work for the long haul? Is vendor support available for the language?
2. What is the environment for the project (web, mobile, etc)?
3. Do we need to consider some infrastructure such as new hardware? What kind of deployment do we need?
4. What's the preference of the client?
5. Any specific requirements for libraries, features, and tools for the programming language?
6. Is the developer available to code in this language or do we need to hire new developers? Are they experienced and comfortable in working with this language, or do they need to learn the language quickly?
7. What are some important constraints of this project? Time, budget, resources?
8. What's the performance consideration and is the languages suitable to accommodate this performance?
9. What's the security consideration and do we need to use any third-party tool?

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
(An aspect is a subprogram that is associated with a specific property of a program)

Object-Oriented Languages: allows the developer to code in terms of objects

Choice of development environment

A development environment in software development is a workspace for developers to make changes without breaking anything in a live environment.

- Commercial v/s 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

What is a software bug?

A **software bug** is an error, flaw or fault in the design, development, or operation of computer software that causes it to produce an incorrect or unexpected result, or to behave in unintended ways.

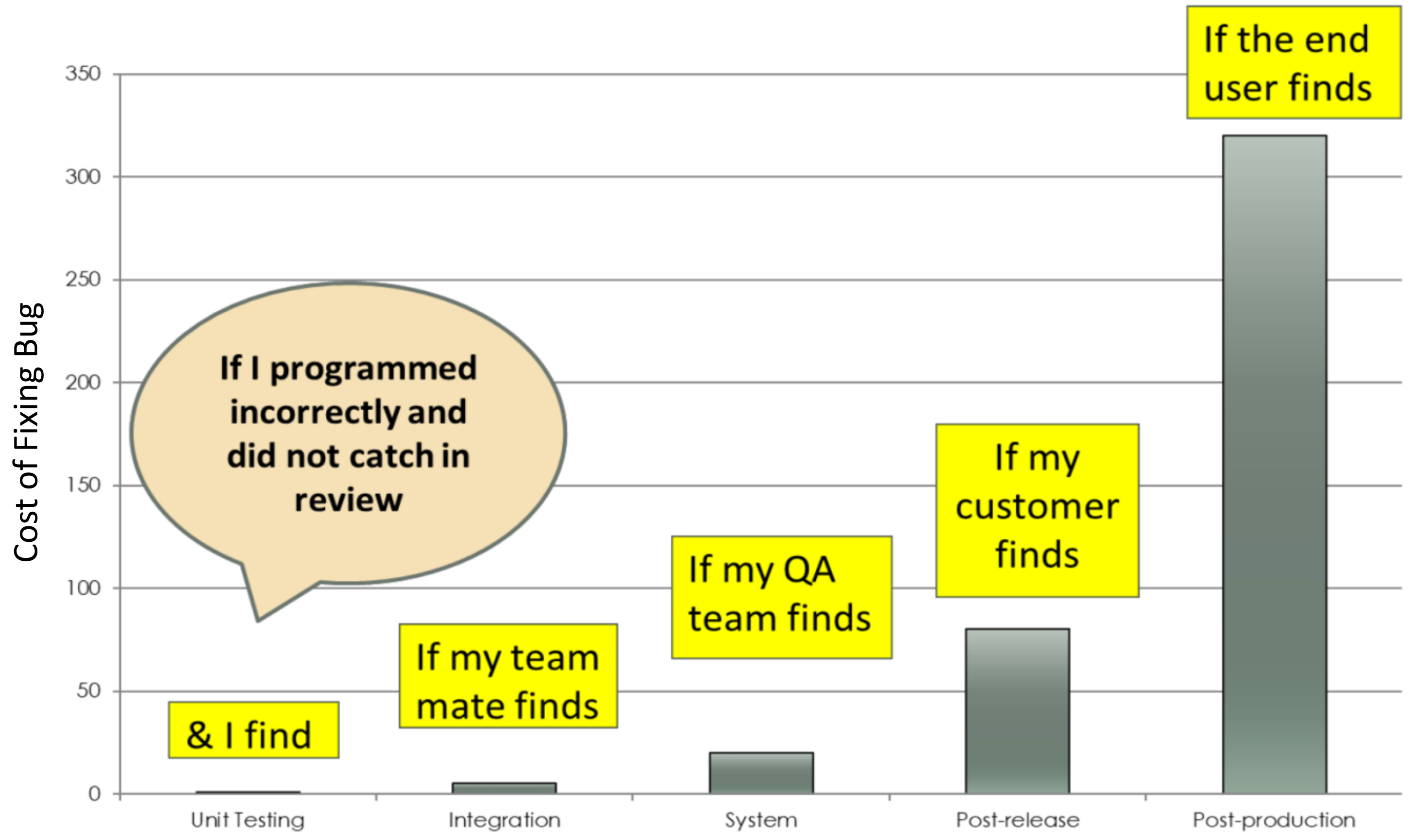
Example: Buggy Code

```
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

Cost of fixing a bug



Program personalities

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

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

Characteristics of code

Programs should be simple and clear

Programs should follow previously agreed upon naming conventions.

Structuring

Easy to read and understand

Characteristics of code – 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

Software Engineering

Characteristics of code – 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 used 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

Characteristics of code – 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?

Characteristics of code – 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

Characteristics of code – 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?

```
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’)
```

Software Engineering

Characteristics of code – 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.

<u>Insert relevant comments in the block of Code shown below:</u>

```
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
3. Use user-defined data types to model entities in the problem domain
4. Hide data structures behind access functions
5. Use appropriate variable names
6. Use indentation, parentheses, blank spaces and lines to enhance readability

Don'ts

1. Don't be too clever
2. Avoid "dangling if"
3. Avoid "dangling else"
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

Dangling if, dangling else, …

Dangling "if"

```
if (condition) { }
        if (condition 1) { }
                if (condition 2) { }
else { }
```

In the above example, there are multiple "*ifs*" with multiple conditions and here we want to pair the outermost if with the else part. But the else part doesn't get a clear view with which "*if*" condition it should pair. This leads to inappropriate results in programming.

Software Engineering

Dangling if, dangling else, …

<u>Dangling “else”</u>

“Dangling else” can lead to serious problems. It can lead to wrong interpretations by the compiler and ultimately lead to wrong results.

For example:

```
Initialize k=0 and o=0
if(ch>=3)
if(ch<=10)
            k++;
else
            o++;
```

In this case, we don’t know when the variable “*o*” will get incremented. Either the first “*if*” condition might not get satisfied or the second “*if*” condition might not get satisfied. Even the first “*if*” condition gets satisfied, the second “*if”* condition might fail which can lead to the execution of the “*else”* part. Thus it leads to wrong results.

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, enforced as a part of reviews

<u>Examples of Standards and Guidelines</u>
Standard headers for different modules
Naming conventions for local and global variables

One declaration per line
Avoid magic numbers

Software Engineering
Coding Standards

- Provide a uniform appearance to code written by different engineers
- Improve readability, maintainability and reduce complexity
- Proactively address some commonly occurring issues with code
- Help in code reuse
- Promote sound programming practices and increase efficiency of programmers

Examples

- Rules for usage of type of data (local, global)
- 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
3. Use static and dynamic analysis tools to detect additional flaws (??)
4. Default Deny: access decisions are based on permissions
5. Adhere to Principle of Least Privilege: process should execute with least set of privileges
6. Elevated permissions for least amount of time
7. Sanitize Data sent to other systems: Sanitize all data sent to complex subsystems

Coding Standard Practices

Note:
Static analysis is a method of debugging that is done by automatically examining the source code without having to execute the program.

Static analysis is generally good at finding coding issues such as:

1. Programming errors
2. Coding standard violations
3. Undefined values
4. Syntax violations
5. Security vulnerabilities

Static code analysis is a form of white-box testing that can help identify security issues in source code. On the other hand, dynamic code analysis is a form of black-box vulnerability scanning that allows software teams to scan running applications and identify vulnerabilities.

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

Scaffolding is **any code that we write, not as part of the application, but simply to support the** process of Unit and Integration testing

Coding Guidelines

- ▪Provide a uniform appearance to code written by different engineers
- ▪Generic suggestions that improve understanding and readability of code
- ▪Reduce cost incurred by early detection of errors
- ▪Reduce complexity and ease maintenance over lifetime of the product

<u>Examples</u>

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 same identifiers for multiple purposes

One of the more commonly used "C coding" guidelines" document is here.

Software 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 projects

Sprint Burndown

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

Team Velocity Metric

- "Amount" of <u>software or stories completed during a sprint</u>
- 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

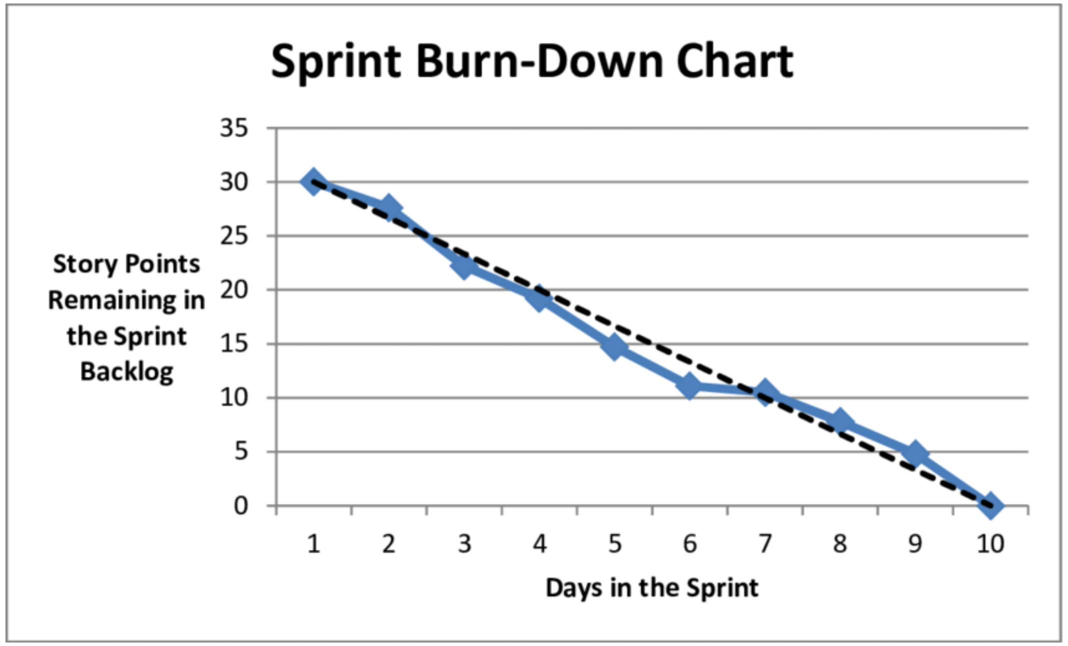
Sprint burndown chart

The Sprint Burndown Chart makes the work of the Team visible.

It is a graphic representation that shows the rate at which work is completed and how much work remains to be done.

The chart slopes downward over Sprint duration and across Story Points completed.

What makes the chart an effective reporting tool is that it shows Team progress towards the Sprint Goal, not in terms of time spent but in terms of how much work remains.



Managing construction – Construction Technical Quality

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

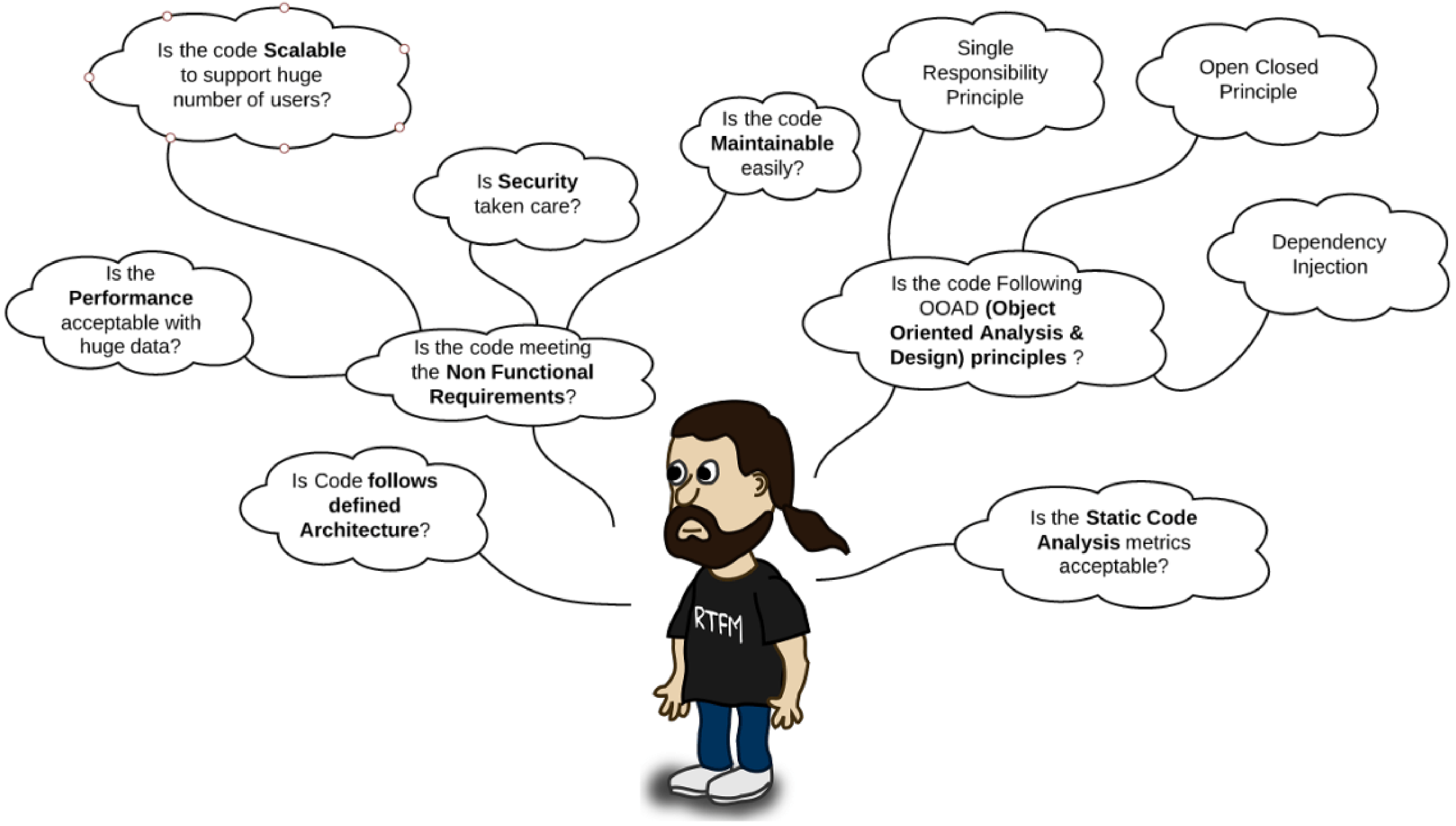
Code review

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

What to review for?

1. Correctness
2. Error Handling
3. Readability
4. Coding standards/guidelines
5. Optimization
6. Use Review Checklists

Software Engineering
Code review



Is the code Scalable to support huge number of users?
Is the Performance acceptable with huge data?
Is Security taken care?
Is the code Maintainable easily?
Single Responsibility Principle
Open Closed Principle
Dependency Injection
Is the code meeting the Non Functional Requirements?
Is the code Following OOAD (Object Oriented Analysis & Design) principles ?
Is Code follows defined Architecture?
Is the Static Code Analysis metrics acceptable?
RTFM

Expert Code Reviewer

Software Engineering

Code inspection

Note

Single Responsibility principle:

A module should be responsible to one and only one actor.

Open Closed principle

Software entities should be open to extensions and closed for modifications

Dependency Injection

Having the creation and binding of the dependent objects outside of the class that depends on them.

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 every 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 record every keystroke and mouse click for a replay
- Done accurately and Quickly
- **Example:** Selenium

Wizards

Tools that generate tests from input parameters

Software Configuration Management (SCM)

Software Configuration Management (SCM) is a software engineering discipline consisting of standard processes and techniques often used by organizations to manage the changes introduced into its software products. SCM helps in identifying individual elements and configurations, tracking changes, and version selection, control, and baselining.

SCM is a Process to systematically organize, manage and control changes in documents, code and other entities that constitute a software product.

SCM provides a mechanism for managing all changes in an efficient, cost-effective and timely manner.

SCM also puts into place a framework for maintenance and support of a product.

Software Configuration Management (SCM) …

Goals of SCM

Increase productivity by increased and planned coordination among the programmers and eliminate confusion and mistakes

1. Identifying elements and configurations, tracking changes, version selection, control and baselining

2. Avoid configuration related problems

3. Effective management of simultaneous updating of source files

4. Build management

5. Defect tracking

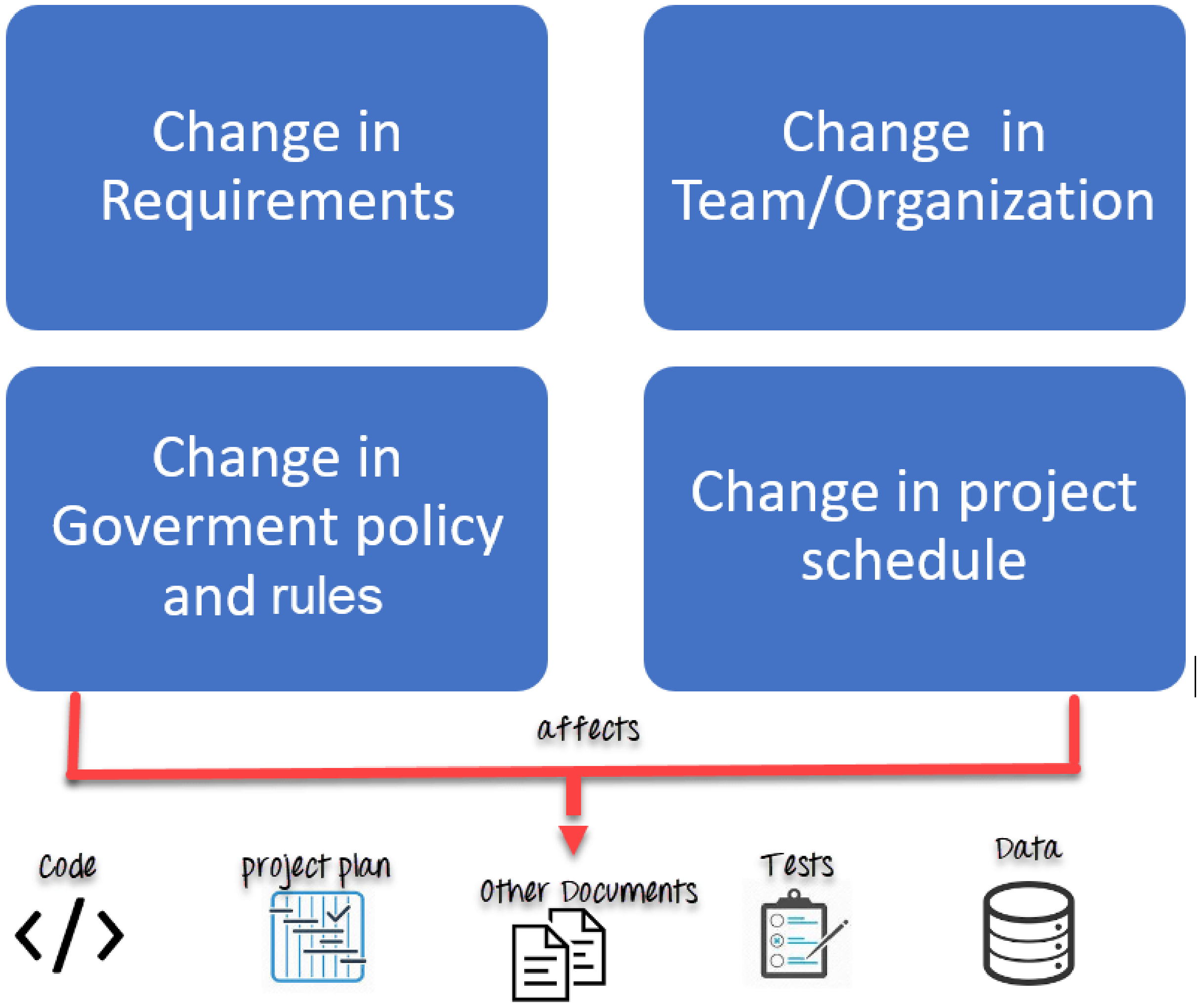
Software Configuration Management (SCM) …

Need for SCM

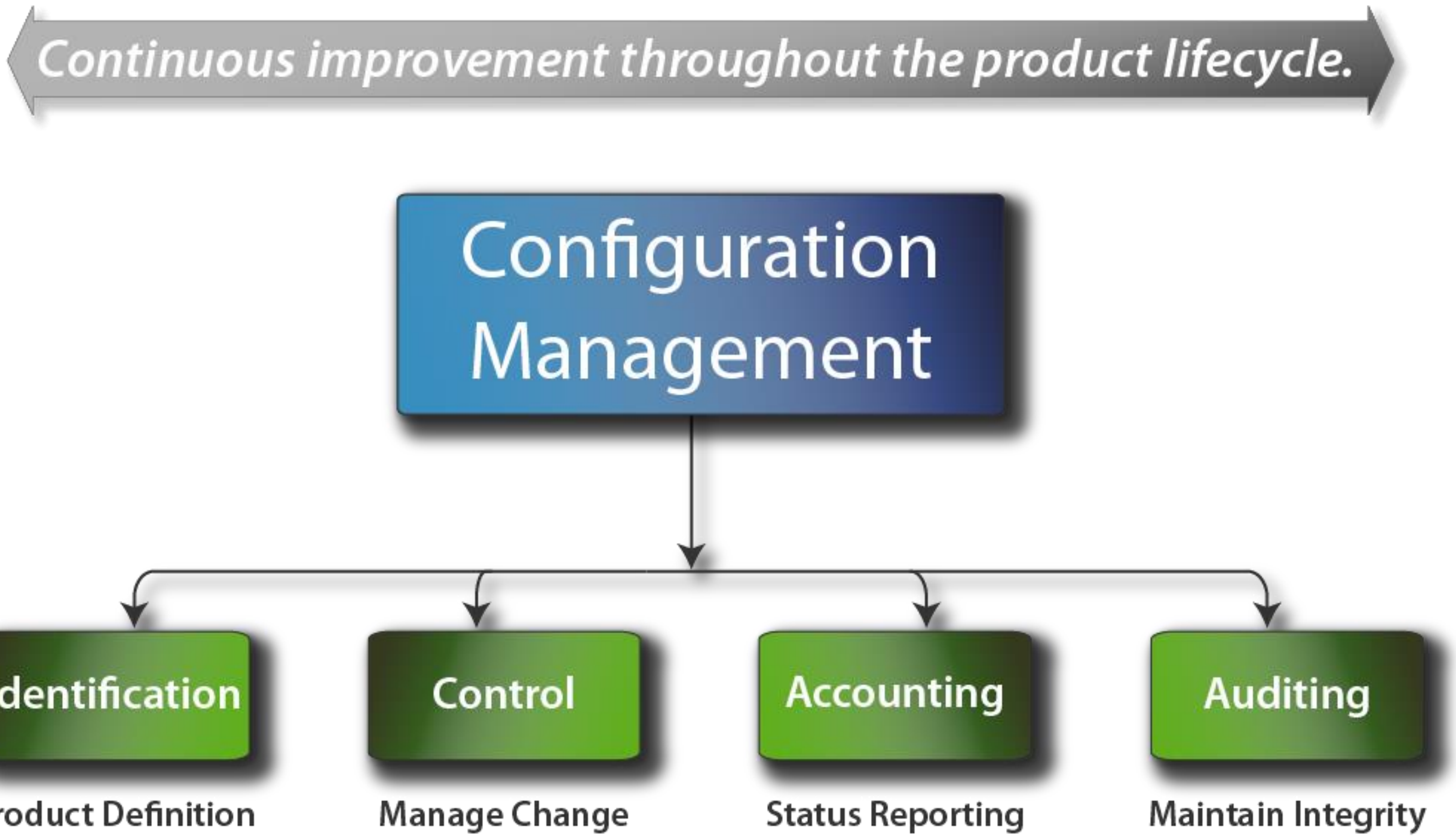
Software engineering in large products/projects with multiple people across locations need to be reliably managed for

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

Software Configuration Management (SCM) …


Change in Requirements
Change in Team/Organization
Change in Goverment policy and rules
Change in project schedule
affects
Code
project plan
Other Documents
Tests
Data

Basic elements of SCM



Basic elements of SCM ..

SCM Element	Method
Software Configuration Item identification	Define baseline components
Software Configuration Control	Mechanism for initiating, preparing, evaluating, approving or disapproving all change proposals.
Software Configuration Auditing	Mechanism for determining the degree that the current state of a software system reflects its baselines (requirements and planning documents)
Software Configuration status accounting	Mechanism for maintaining a record of how a system has evolved, and the state of a system relative to published documents and written agreements.

SCM in Scrum-Agile approach

1. SCM is the responsibility of the "whole team" and is automated as much as possible.

2. The definitive versions of components are held in a shared project repository.

3. 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.

4. Once the changes are tested, the modified components are pushed to the project repository.

5. Versions of the modified code are 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

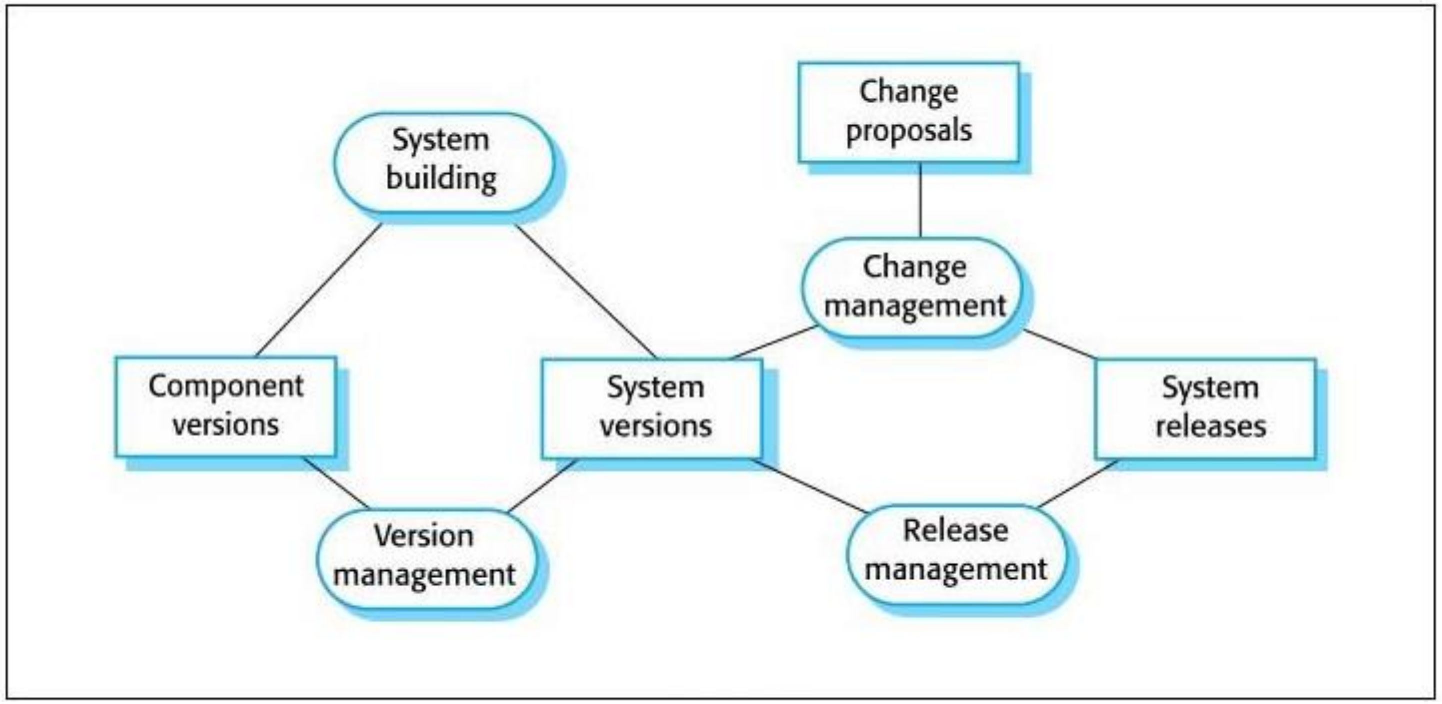
1. Planning is done by a Configuration manager and starts during the early phases of the project

2. Results in a Software Configuration Management Plan
   a. May follow a public or internal standard
   b. Defines the Configuration Items and a naming scheme
   c. Define who takes responsibility for Configuration Management procedures and creation of baselines
   d. Defines policies for change control and version management
   e. Describes tools to assist in Configuration Management and any limitations
   f. Defines the configuration management database

A sample CM Plan is here.

Software 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

Software Engineering
SCM directories

Programmer's Directory
(Dynamic Library)

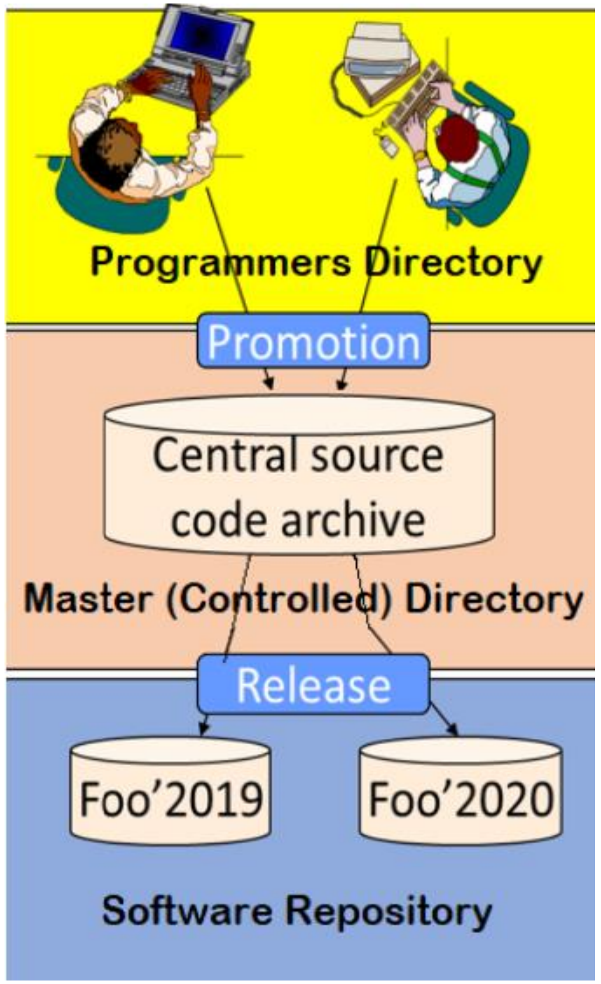
- Library for holding newly created or modified software entities
- Controlled by the programmer

Software Repository
(Static Library)

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

Master Directory
(Controlled Library)

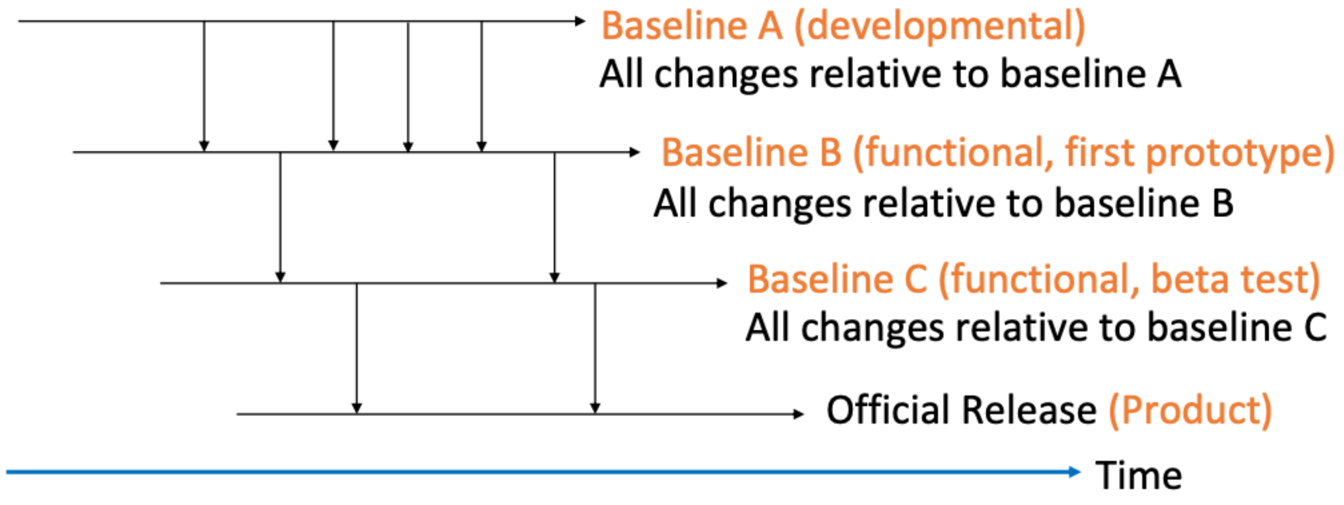
- Manages current baselines and for controlling changes
- Entry is controlled, after verification
- Changes must be authorized



Baselines

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

- 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


Baseline A (developmental)
All changes relative to baseline A
Baseline B (functional, first prototype)
All changes relative to baseline B
Baseline C (functional, beta test)
All changes relative to baseline C
Official Release (Product)
Time

SCM Activities

Codeline: This term commonly describes **a group of files that evolve together over the life of the software application or application component**. A codeline might include source code, build scripts, images, compiled code, and other files that are required to create the application.

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

Reasons for Branching

1. Supporting concurrent development
2. Capturing of solution configurations
3. Support multiple versions of a solution
4. Enable experimentation in isolation without impacting the stable version
5. Branching ensures that overall product is stable
6. Merging is bringing back and integrating changes over multiple branches
7. Frequent merging helps decrease the likelihood and complexity of a merge conflict.

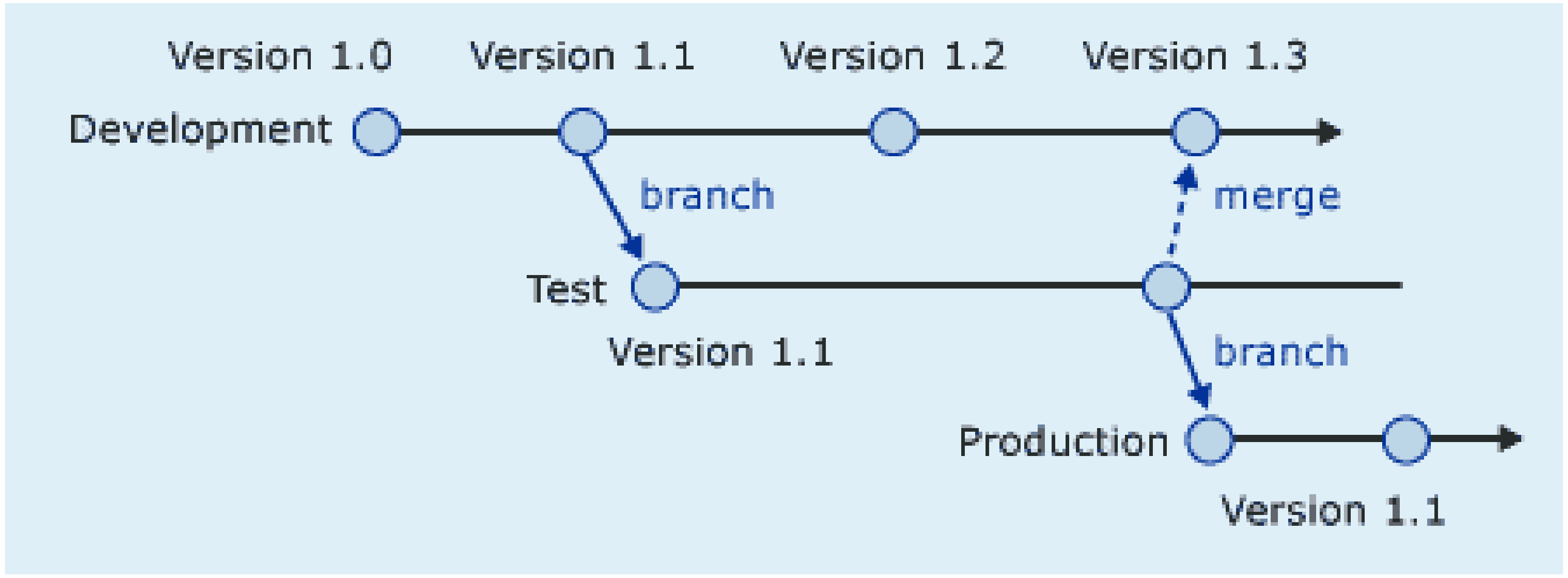
SCM Activities – Branch management

Branching in version control and software configuration management, is <u>the duplication of an object under version control</u> (such as a source code file or a directory tree). Each object can thereafter be modified separately and in parallel so that the objects become different. In this context the objects are called **branches**. The users of the version control system can branch any branch.

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

Branching


Version 1.0
Version 1.1
Version 1.2
Version 1.3
Development
branch
merge
Test
Version 1.1
branch
Production
Version 1.1

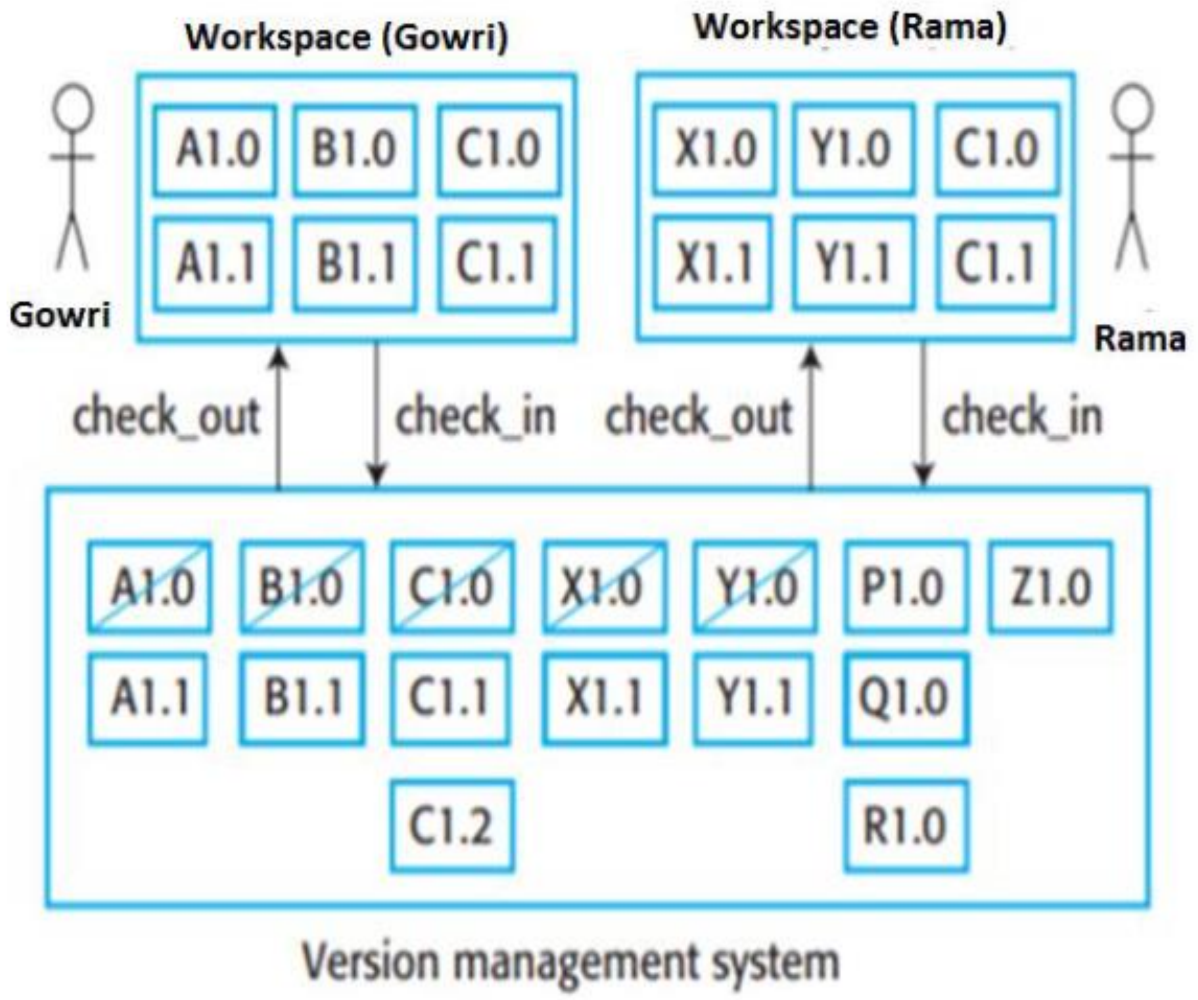
SCM Activities – version management

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)

<u>Key Features of Version Control System</u>

1. Changes are attributable or traceable
2. Change history is recorded and can be reverted
3. Better conflict resolution
4. Easier code maintenance and quality monitoring
5. Less software regression
6. Better organization and communication

SCM Activities – Version management



Software Engineering

Food for thought

```
str="\u00B2“

print(len(str))

??

str=“00B2”)

print(len(str))

??
```

SCM Activities – Build management

Build Management: creating the application program for a software release by compiling and linking source code and libraries to build artifacts 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

1. Fetching code from the source control repository
2. Compiling the code and checking the dependencies
3. Linking the libraries, code, etc
4. Running tests and building artifacts.
5. Archive logs and send notification emails
6. May result in version number change.

SCM Activities – 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

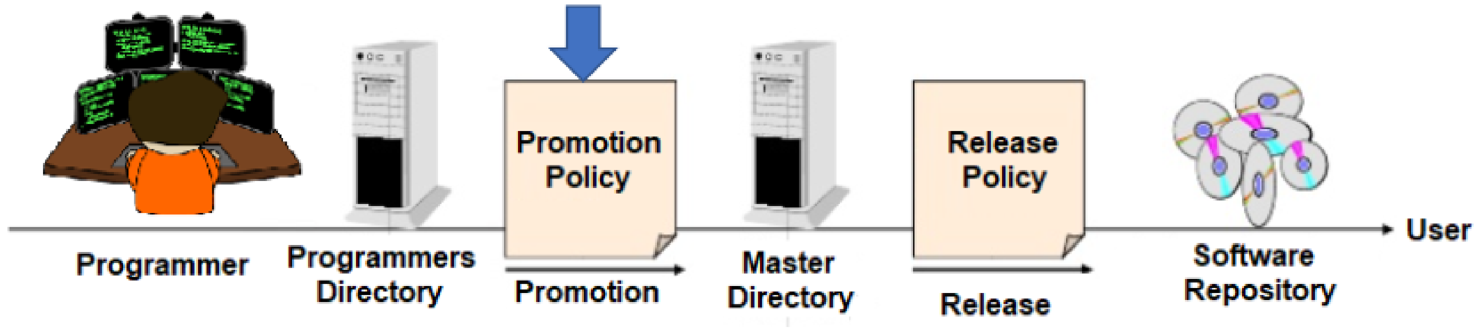
Sometimes may be automated using zip, shell scripts, InstallAware and Jenkins

SCM Activities – Promotion management

Changes made by a programmer are only available in their environment and need 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?



Software Engineering

SCM Activities – 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.

<u>General change process</u>

- ❖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

SCM Activities – 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

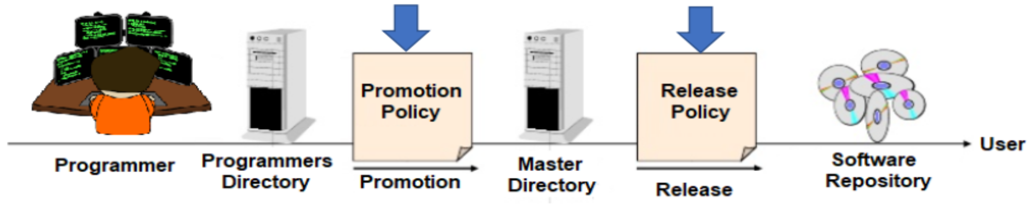
Tools, resources and training are required to perform baseline change assessment

- ❖File comparison tools to identify changes
- ❖Other resources and training depending on size and complexity of project

SCM Activities – Controlling changes

Changes are controlled at two points

1. Promotion Policy
2. Release Policy



<u>Change Policies</u>: 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 <u>guarantee that each version, revision or release conforms to commonly accepted criteria and a consistent process is applied to how things are done</u>

- ✓Enforced through engineering processes and tools
- ✓Audited to ensure conformation

SCM Activities – 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

<u>Version vs Release vs Revision</u>

Release: formal distribution of an approved version

Version: Initial **release** or **re-release** of a configuration item <u>associated with a complete compilation or recompilation of the item</u>

Different versions have different functionality

Revision: change to a version that <u>corrects only errors in the design/code</u>, but <u>does not affect the documented functionality</u>.

SCM Activities – 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

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

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, changes made since the last merge are found out (through the tool) and logged

You can find most of the git commands in the document here.

Software Build

Software build: process of converting source code into standalone software artifact(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

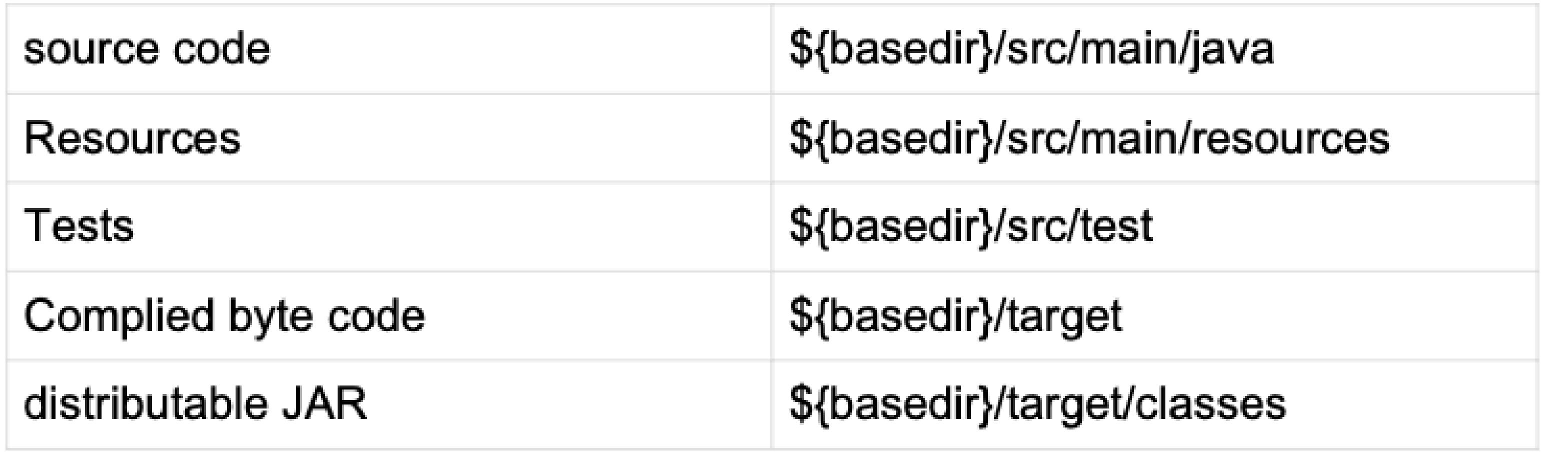
- Software project management and comprehension tool
- Based on project object model

Typical Makefile

```
CC = gcc  #indicates the compiler being used
CFLAGS = -g
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?

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

clean: FRC rm –f helloworld helloworld.o
            #Question: Is this command necessary? What is its role?
```

Software Engineering
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 Engineering
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

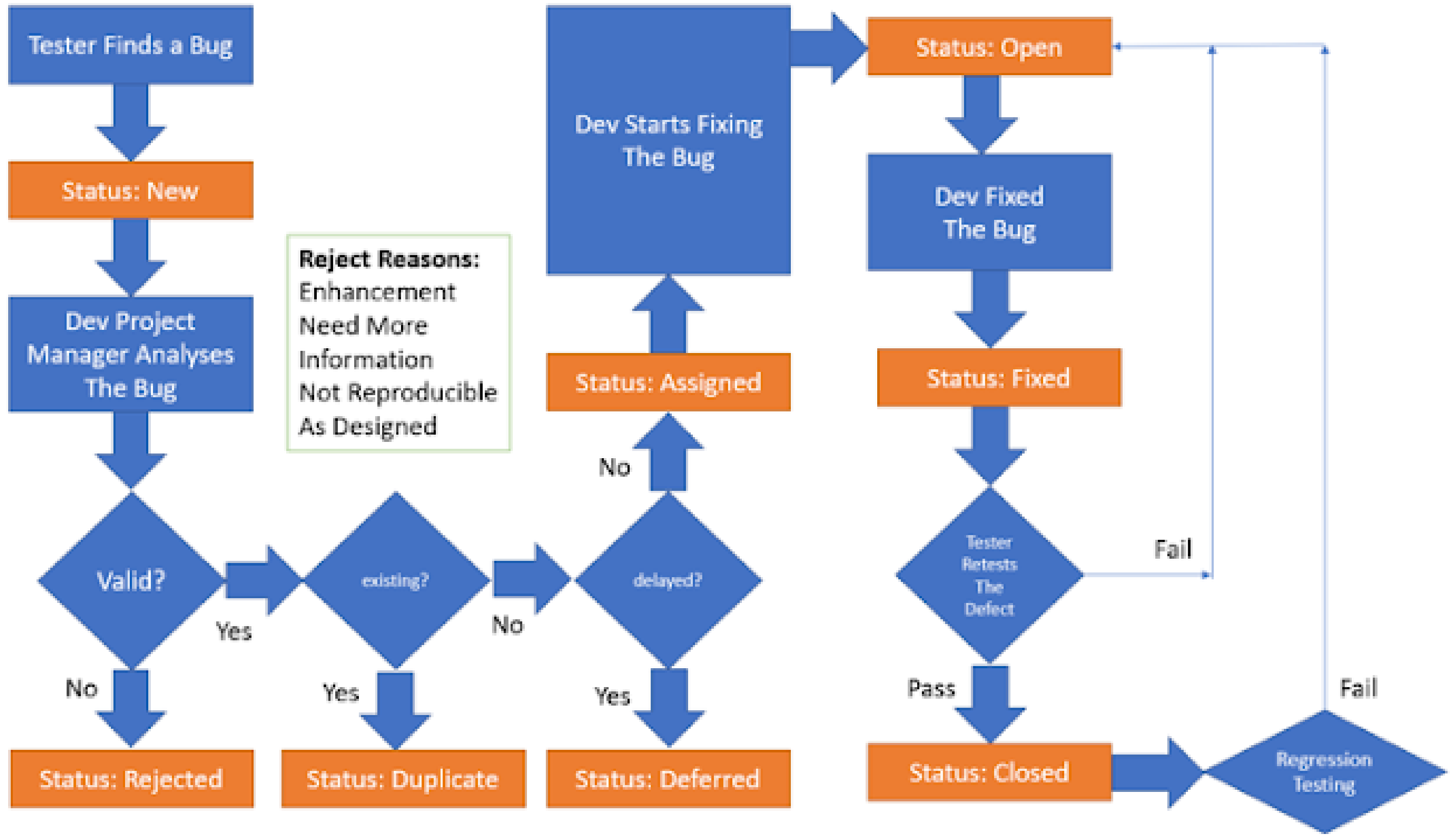
Software Engineering
Software Bug or Defect Tracking tools

Software application that keeps track of reported software bugs in a software project
Clear and centralized overview of development requests (bugs and improvements)

Question: In terms of the MS word app, can you think of a bug and an improvement?

Bug tracking systems support the lifecycle of a bug from logging to resolution
Examples: Bugzilla, FogBugz, Trac Edgewall, Backlog, etc

Software Engineering
Software Bug or Defect Lifecycle



22/10/2024

Software Quality

Software systems are complex and evolve, hence there is a need for continuous assessment and evaluation of quality

Bad quality of software could have serious repercussions on life if we are looking at life-critical applications.

Can definitely lead to dissatisfied customers

Software quality enhances long term profitability of products

Software Quality – Different perspectives

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 perspective

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 perspective ….

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 a 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

Software quality - Terminology

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 measure 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

Characteristics of software measure and metrics

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 quality

Cost of quality (COQ) is defined as a methodology that allows an organization to determine the extent to which its resources are used for activities that **prevent poor quality**, that appraise the quality of the organization's products or services, and that result from internal and external failures.

Having such information allows an organization to determine the potential savings to be gained by implementing process improvements.

Cost of Good Quality (COGQ) and Cost of Poor Quality (COPQ)
COGQ consists of the cost of quality conformance, including any associated costs with both appraisal and prevention, whereas COPQ involves all the nonconformance costs that are both internal and external to the company.

Software Engineering

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 (forewarning of errors), 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

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

Note:
Halstead's metrics are based on the number of distinct operators and operands in the program and are used to estimate the effort required to develop and maintain the program.

Details on Halstead's metrics can be found at:
https://www.geeksforgeeks.org/software-engineering-halsteads-software-metrics/

Software metrics categorization …

Product Metrics

- Assessing the state of the project
- Tracking potential risks
- Uncovering problem areas
- Adjusting workflow or tasks
- Evaluating team's 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

Software Quality Assurance …

Who is involved in the SQA activities?

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

An SQA Plan template (as per IEEE standards) is here.

Software Engineering Institute - CMM

Capability Maturity Model (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

Software Engineering Institute – CMM …

Characterizing CMM Process terminology

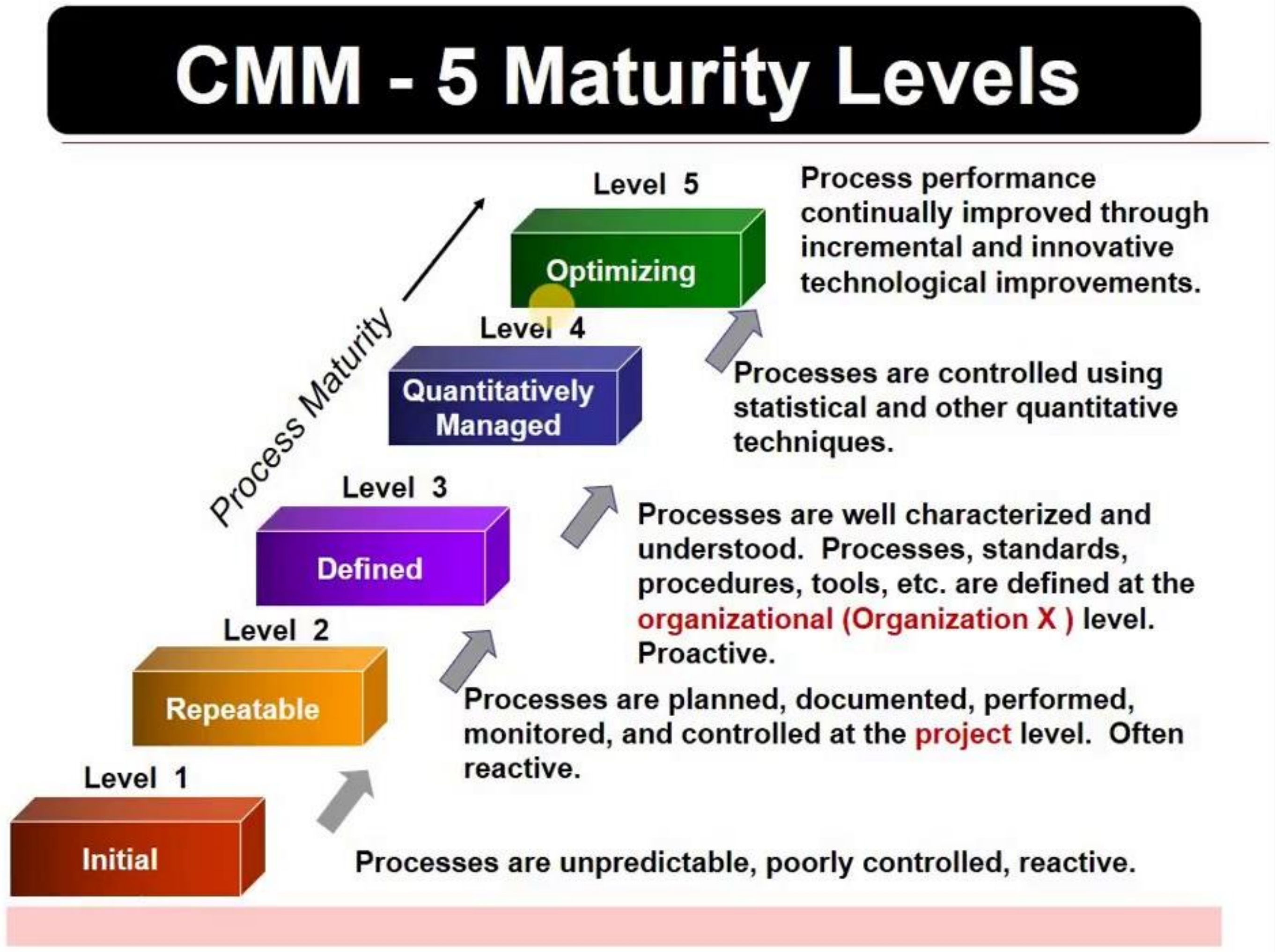
Process
- Activities, methods, practices and transformations to develop and maintain software

Process Capability
- Ability of process to 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

Software Engineering Institute – CMM …



Process performance continually improved through incremental and innovative technological improvements.

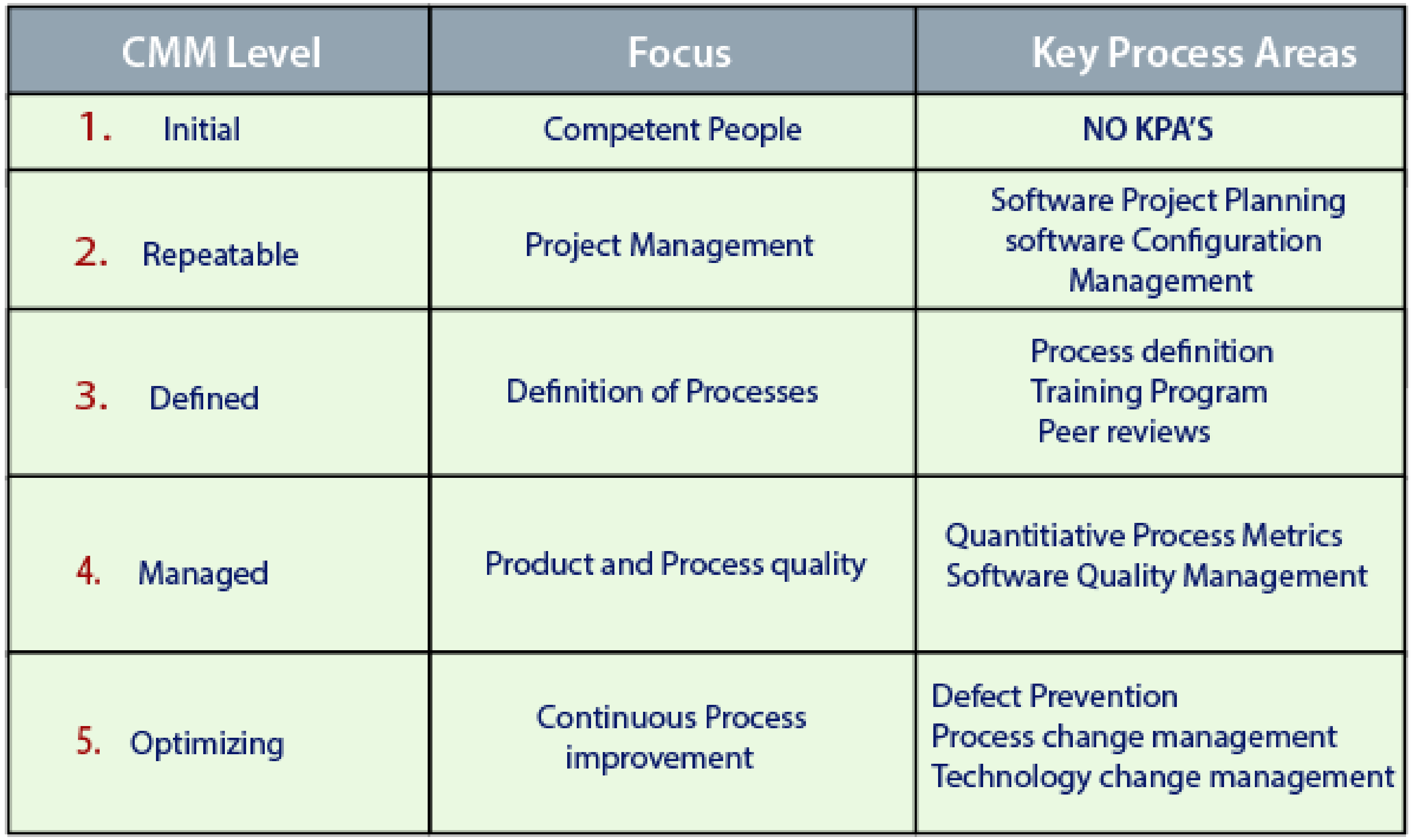
Processes are controlled using statistical and other quantitative techniques.

Processes are well characterized and understood. Processes, standards, procedures, tools, etc. are defined at the organizational (Organization X) level. Proactive.

Processes are planned, documented, performed, monitored, and controlled at the project level. Often reactive.

Processes are unpredictable, poorly controlled, reactive.

Software Engineering Institute – CMM …



CMM Level	Focus	Key Process Areas
1. Initial	Competent People	NO KPA'S
2. Repeatable	Project Management	Software Project Planning software Configuration Management
3. Defined	Definition of Processes	Process definition Training Program Peer reviews
4. Managed	Product and Process quality	Quantitiative Process Metrics Software Quality Management
5. Optimizing	Continuous Process improvement	Defect Prevention Process change management Technology change management

The focus of each SEI CMM level and the Corresponding Key process areas.

Software Engineering Institute – CMM …

The first 3 levels (1,2 3) are **qualitative**

Levels 4 and 5 are **quantitative**

Process maturity perspective

Initial (just do it)

Repeatable (focus on project management)

Well defined (organized assets)

Analyzed, improved and managed (quantitative control)

Improved and Optimized (continuously improving)

Organization maturity perspective

- ➢Work accomplished according to plan
- ➢Practices consistent with processes
- ➢Processes updated as necessary
- ➢Well-defined roles/responsibilities
- ➢Inter-group communication and coordination
- ➢Management formally commits

Software Engineering

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
- Judgment 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

M S Anand
Department of Computer Science Engineering

anandms@pes.edu