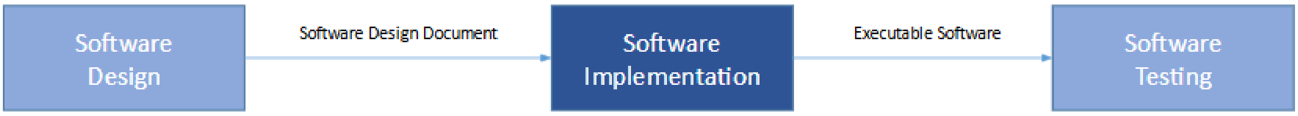
Unit 3 speedrun

26 November 2024 18:14

Lec 1

Context



- Prior to implementation
  - Choose between compiled and interpreted language
  - Decide on development environment
  - Follow configuration management plan
- During implementation
  - Use coding standards and guidelines
  - Follow language syntax
  - Address quality, security, testability
  - Provide unit tested, peer-reviewed functioning code

Software Implementation

- Detailed creation of working software through a combination of coding, reviews and unit testing
- Goals:
  - Minimize complexity
  - Reuse
  - Anticipate change
  - Readable
  - Verifiable
- Characteristics
  - Produces high volume of configuration items
  - Related to software quality
  - Extensive usage of CS knowledge
  - Tool intensive
- Choice of programming language:
  - Assembly language
    - Map directly onto cpu architecture
  - Procedural language
    - Modest level of abstraction from underlying layer
  - Aspect orientation support
    - Allow separation of aspects during development
  - Object oriented languages
    - Allow developer to code in terms of objects
- Choice of development environment
  - Commercial vs open source
  - Support of development process
  - Security
  - Account for future capabilities
  - Integration between tools and environment

Program personalities

- Messy
- Verbose
- Cryptic

- Neat

Programmer personalities

- 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)

Principles of software construction

- Modularity
    - Code should be divided into modules
    - Modules encourage reusability
- Separation of concerns
    - Code should be organized so that each part addresses a distinct concern
- Abstraction and encapsulation
    - Allows developers to hide complexity by focusing on essentials
    - Encapsulation keeps data and operations within a module or class private
- SOLID principles
    - Single responsibility, Open/closed window, Liskov substitution, Interface segregation, Dependency inversion)
    - Set of design principles to ensure software is flexible, maintainable and scalable
- Testability
    - Software should be designed and written in a way that it's easy to test individual components

Tools and best practices

- Version control system
    - Tools like git
- Automated testing and CI/CD pipelines
    - Automated testing tools are CI pipelines help catch defects early
- Code reviews
    - Regular peer reviews ensure code quality, adherence to coding standards, and prevent potential issues
- Documentation
    - Helps new developers understand

<u>Lec 2</u>

Characteristics of code

- Programs should be simple and clear
    - Programmers should use a reasonable amount of:
        - Lines per function
        - Lines/function per file
        - Arguments per function
        - Levels of nesting
        - Conditions
- Structuring
    - Structure: dependencies between software components
    - Dependencies can be highlighted via proper naming and layout
    - File structure:
        - Logically grouped
        - Well partitioned
    - Code and data structure:
        - Well encpsulated and logically grouped
        - Properly initialized

- Naming
    - Naming conventions
        - Names have to be meaningful and descriptive
        - Avoid using names similar to keywords
        - No start with _
        - Use a minimum of 2-3 characters
        - Don't use numeric values
    - Types of naming
        - Pascal: FirstWord
        - Camel: firstWord
        - Underscore: _firstWord
- Easy to read and understand
    - Readability depends on identifier naming and visual layout of statements
    - Comments should concisely and clearly explain the logic of the program

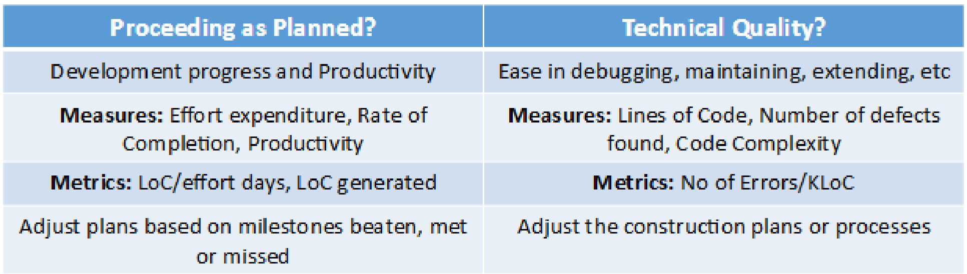
Coding standards and guidelines

- Standards: rules which are mandatory to be followed
- Guidelines: rules which are recommended to be followed
- Provides a uniform appearance to code written by different engineers
- Improves readability, maintainability, and reduces complexity
- Proactively addressed commonly occurring issues with code
- Helps in code reuse
- Common coding practices:
    - Defensive programming
        - Murphy's law: If anything can go wrong, it will
    - Secure programming
        - Developing computer software to guard against accidental introduction of security vulnerability
        - Practices:
            - □ Validate input
            - □ Heed compiler warnings
            - □ Default deny
            - □ Adhere to principle of least privilege
            - □ Sanitize data sent to other systems
    - Testable programming
        - Practices:
            - □ Assertions
            - □ Test points
            - □ Scaffolding
            - □ Test harness
            - □ Test stubs
            - □ Instrumenting
            - □ Building test data sets

<u>Lec 3</u>

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



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

- Quality for agile scrum project:
  - Sprint burndown
    - Goal of team is to consistently deliver all the work
  - Team velocity metric
    - Amount of software or stories completed during a sprint
  - Throughput
    - Indicates total value-added work output by the team
  - Cycle time
    - Total time that elapses from the moment work is started on an item till completion
- Construction technical quality
  - Peer review
  - Unit testing
  - Test first
  - Code stepping
  - Pair programming
  - Debugging
  - Code inspection
  - Static analysis

Code review

- Systematic examination of software code by one or more developers
- What to review for:
  - Correctness
  - Error handling
  - Readability
  - Coding standards/guidelines
  - Optimization
- Software inspection
  - Identify defects early
  - Ensure conformance to specifications
  - Formal process
- Important points:
  - Conformance to specifications, not customer requirements
  - Preconditions for effective inspection
    - Precise specifications must be available
    - Familiarity with standards
    - Syntactically correct code
    - Error checklist
  - Formalized approach to code and document reviews
    - Inspections are structured, documented, and typically include formal meetings where roles are assigned
  - Benefits of software inspections
    - Early detections of defects
    - Improvement in code quality
    - Knowledge sharing

Unit testing tools

- Unit testing frameworks
  - Allows tested to enter method name, parameters and expected results

- Code coverage analyzers/debuggers
    - Code coverage helps identify code that is not covered by test cases
- Record-playback tools
    - Lets tester start a sessions that records every keystroke and mouseclick for replay
- Wizards
    - Tools that generate tests from input parameters

<u>Lec 4</u>

Software configuration management

- Process to systematically organize, manage, and control changes in documents, and other entities that constitute a software product
- Goal: Increase productivity by increased and planned coordination among programmers
    - Identifying elements and configurations, tracking changes, version selection, control and baselining
    - Avoid configuration related problems
    - Effective management of simultaneous updating of source files
    - Building management
    - Defect tracking
- Need for SCM:
    - Multiple people working on project
    - More than one version
    - Working on released systems
    - Changes in configuration due to changes in user requirements
    - Software must run on different machines
    - Coordination among stakeholders
    - Controlling costs in making changes
- Scrum-agile approach
    - SCM is responsibility of whole team
    - Definitive versions of components are kept in shared project repository
    - Developers copy versions from the repo into their workspace
    - Once changes are tested, modified components are pushed onto project repo
    - Versions of modified code available to all team members
- Benefits:
    - Permits orderly development of software configuration items
    - Ensures orderly release and implementation of new or revised software products
    - Ensures only approved changes to both new and existing software products
    - Ensures that software changes are implemented in accordance with approved specifications
    - Ensure that documentation accurately reflects updates
    - Evaluates and communicates the impact of changes
    - Prevents unauthorized changes
- Roles:
    - Configuration manager
    - Developer
    - Auditor
    - Change control board member
- Planning:
    - Planning initiation
    - SCM plan
    - Standards compliance
    - Configuration items defined and establishes a naming scheme
    - Responsibility assignment specifies who is responsible for configuration management procedures and baseline creation
    - Change control and management policies
    - Configuration management tools
    - Configuration management database

    - Audit and review procedures
    - Communication guidelines
    - Risk management
    - Training and support
    - Ongoing maintenance of the SCMP

Configuration management activities

- Configuration item identification
    - Independent hardware/software that is designated for configuration management
    - Could be:
        - All types of code files and drivers for tests
        - Requirement, analysis, design, test, and other docs
        - User or developer manuals
        - System config
    - What items need config control:
        - Come items must be maintained for the lifetimes of the software
    - When to do it:
        - Start too early: too much bureaucracy
        - Start too late: chaos
- Configuration management directories -> repo to store and organize config items
    - Programmer's directory:
        - Dynamic
        - For holding newly created ot modified software entities
    - Software repo
        - Static
        - Archive for various baselines in general use
    - Master directory
        - Controlled library
        - Manages current baselines and for controlling changes
- Baselining: formally approved snapshot of system's config files
    - Baseline= specification that has been formally reviewed and agreed to by responsible management and serves as a basis and can only be changes in formal review
    - Baselines developed after review
- Branch management: create and manage parallel versions of codebase
    - Codeline = progression of source file and artifact which makeup software components as they change over time
    - Branch= copy or clone of all or portion of source code
    - Reasons for branching:
        - Support concurrent dev
        - Capturing solution configs
        - Support multiple vers of solution
        - Enable experimentation in isolation
        - Ensures overall product is stable
    - Merging is bringing back and integratinf changes over multiple branches
        - Frequent merging helps decrease the likelihood and complexity of a merge conflict
    - Branching strategies
        - Single branch
        - Branch by customer or organization
        - Branch by developer or workspace
        - Branch by module or component
    - Branch management entails having a well defines branching policy
- Version management: tracking and controlling multiple versions
    - Controlling, organizing, and tracking different versions in history of computer files
    - Keeping track of diff versions of software components and systems
        - Git
        - Changes to a version are identified by a number (revision number)- 7.5.2
            - 7: release number (defined by customer)
            - 5: version number (developer)

            - 2: revision number (developer)
    - Key versions:
        - Changes are attributable
        - Change history is recorded and can be reverted
        - Better conflict resolution
        - Easier code maintenance
        - Less software regression
        - Better organization and communication
- Build management: Coordinating and automating process compiling, linking and packaging code
    - Creating the application program for software release by compiling and linking source code
    - Done with apache ant, make, 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:
        - Fetch code from repo
        - Compile code and check dependencies
        - Link libraries
        - Running tests and building artefacts
        - Archive logs and send notification emails
        - May result in version number change
- Install: Deploying and configuring software to ensure it runs correctly
    - First interaction with customer
    - Placing multiple files containing executable code
    - 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, jenkins
- Promotion management: moving code through predefined stages
    - Changes made by programmer is only available in their environment and needs to be promoted to central master directory
        - Promotion is done based on certain promotion policies
    - Promotion could be based on baselining criteria which was planned for (it would involve some amount of verification)
    - It's further be authorized and moved to the master directory
- Change management: structured process for handling modifications to system components
    - Change could result in creation of different version or release of the software
    - Deals with changes in Cis which have been baselined
    - General change process:
        - Change can be requested
        - Unique identification is associated with requested change and logged
        - Change is assessed based on impact with other modules
        - Accept or reject
        - All of these activities are tool driven
        - If accepted: change is implemented and validated
        - Plans done and executed for documentation, versioning, mergind, delivery
        - Audit implemented change
    - Complexity varies with proj
        - Small = change requests informally and fast
        - Complex = detailed change request
    - Information is required to process a change to baseline
        - Description of proposed changes
        - Reasons for making changes
        - Lost of items affected by changes
    - Tools, resources and training are required to perform baseline change assessment
        - File comparison tools to identify changes
        - Other resources depending on size and complexity

    - Controlling changes:
        - Promotion policy
        - Release policy
    - Change policies: Promotion and release policy is dictated via environment
        - Informal: research type env
        - Formal: externally developed config items and their releases
- Release management: Planning, scheduling, and controlling the deployment of new or updated software versions
    - Movement of code to customer and software repo
    - Release policy: gating quality criteria that is planned for and includes verification with metrics
        - If metrics met, release
    - Release management: managing, planning, scheduling and controlling a software build through diff stages anad env
    - Release: formal distribution of an approved version
    - Version: different version with different functionality
    - Revision: change to a version that corrects only errors in design/code but no effect on documented functionality
- Defect management: process of tracking, prioritizing and resolving software defects throughout software lifecycle
    - Bug = consequence of coding fault
    - Defect = deviation from expected business req
    - Bugzilla

<u>**Lec 7**</u>

Types of software configuration management tools

- Source code admin
    - RCS: version control
    - ClearCase: multiple servers, process modelling, policy check
    - Concurrent version control: Based on RCS, allows concurrent working, Web frontend
    - Github: development platform for version control
        - Can create repos and add contributors
            - □ Access using ssh or https
        - Changes are pushed as commits
            - □ Commits get unique number + message
        - Master branch is auto clones
        - General contribution method:
            - □ Fork project
            - □ Make changes
            - □ Pull request to merge new code
            - □ Whenever merged, difference is done and changes are saved

git init
#initialize repo

git status
#shows untracked files, commits

git add <files>
#brings untracked files to git

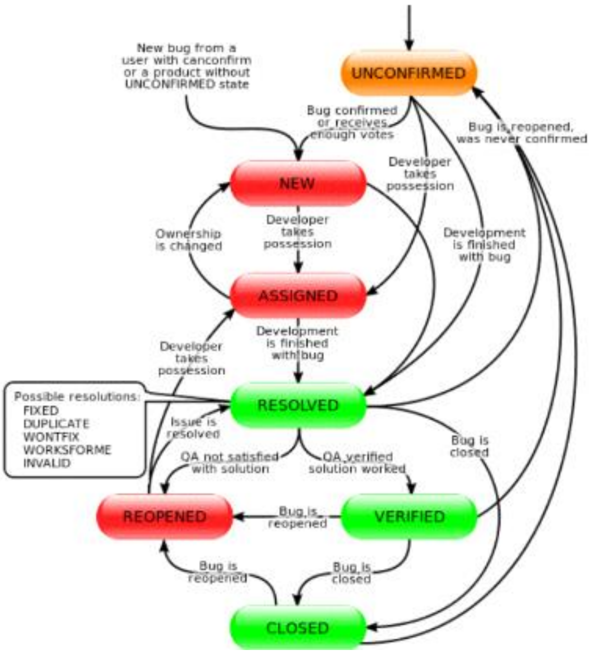
git commit –m "message"
#commits with message

git remote -v
#list of remote origins of your local repository

git push
#push changes onto remote repository

- Software build
    - Source code to standalone software artefact
    - Compilation process: build converted to executable
    - Make: automatically builds executable
    - CruiseCOntrol: opensource for cont software builds
    - FinalBuilder: automate build and release
    - Maven: software project management and comprehension tool
        - Simplifies and standardizes build project
        - Handles compilation, distrib, docum, team collaboration
        - Increases reusability
        - Supports multiple dev team env

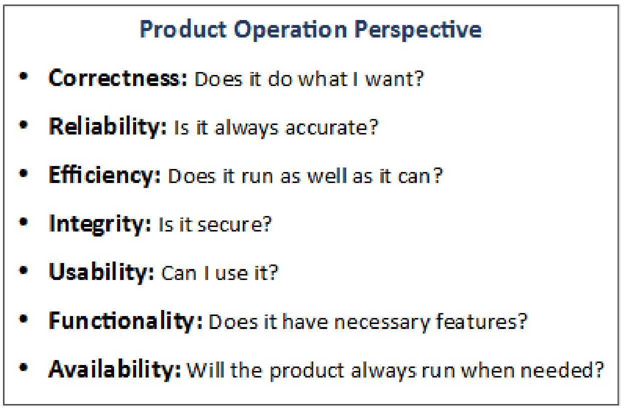
- Software installation
    - Cross platform tools that produce installers for multiple OS
    - Installer: installs all necessary files
    - Boostrapper: small installer that does prerequisites
    - DeployMaster: windows
    - InstallShield: simplifies creation of windows installers
    - InstallAware: windows installer for windows
    - Wise Installer: configure and isntall MS windows
- Software bug tracking


New bug from a user with canconfirm or a product without UNCONFIRMED state
UNCONFIRMED
Bug confirmed or receives enough votes
Bug is reopened, was never confirmed
Developer takes possession
NEW
Ownership is changed
Developer takes possession
Development is finished with bug
ASSIGNED
Developer takes possession
Development is finished with bug
Possible resolutions:
FIXED
DUPLICATE
WONTFIX
WORKSFORME
INVALID
RESOLVED
Issue is resolved
QA not satisfied with solution
QA verified solution worked
Bug is closed
REOPENED
Bug is reopened
VERIFIED
Bug is reopened
Bug is closed
CLOSED

Lec 8

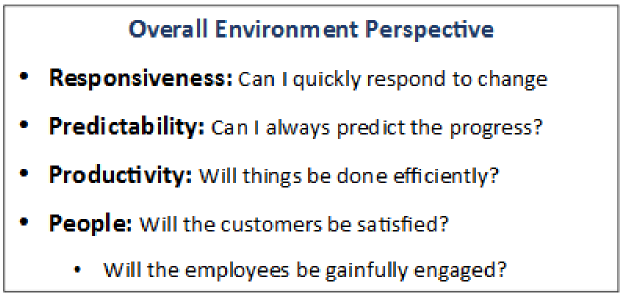
Software quality

- Perspectives:
    - Transcendent (exceeded normal expectation)
    - User based
    - Manufacturing based
    - Product based
    - Balancing time, cost, profits



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?

Quality Attributes (FLURPS+) – *Important!*

- **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

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 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

<u>Lec 9</u>

Software metrics

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

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

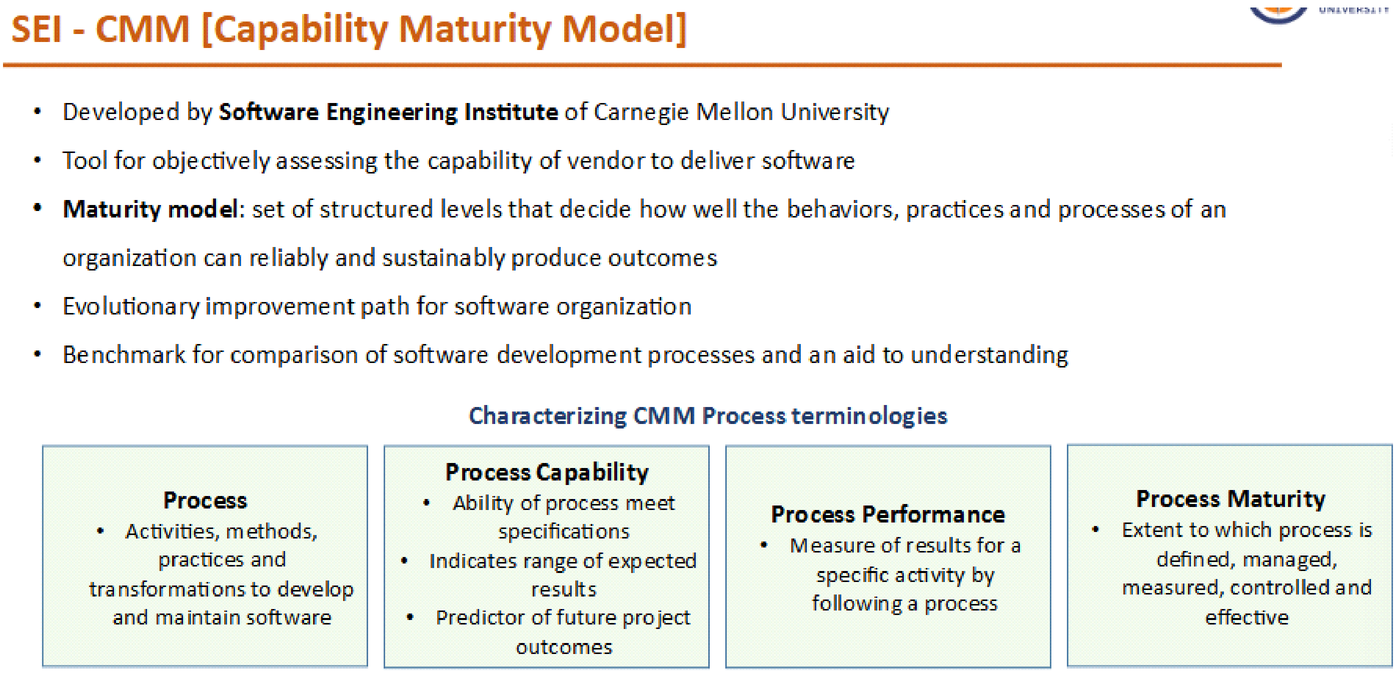
Process Metrics

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

Software quality assurance

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



SEI - CMM [Capability Maturity Model]

- 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

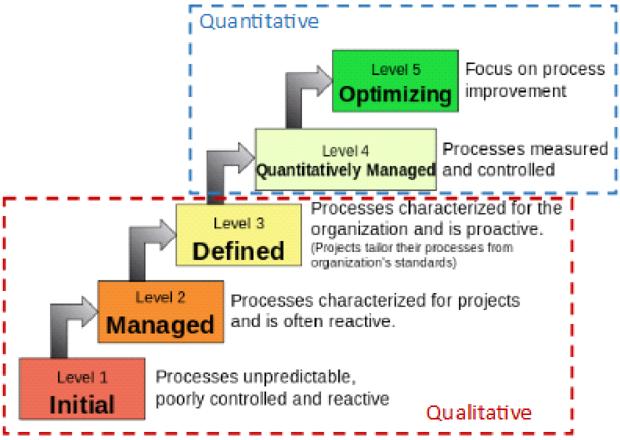
- Measure of results for a specific activity by following a process

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


Quantitative
Level 5
Optimizing
Focus on process improvement
Level 4
Quantitatively Managed
Processes measured and controlled
Level 3
Defined
Processes characterized for the organization and is proactive.
(Projects tailor their processes from organization's standards)
Level 2
Managed
Processes characterized for projects and is often reactive.
Level 1
Initial
Processes unpredictable, poorly controlled and reactive
Qualitative

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

- 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