Software Maintenance and Evolution

Chapter 1: Fundementals of Software Maintenance and Evolution

Definitions

- **Software maintenance**
  - Preventing software from failing to deliver the intended functionalities by means of bug fixing.
  - All support activities carried out after delivery of software. (Bennett and Xu defn)
  - It compromises all activities associated with the process of changing software for the purposes of fixing bugs or/and improving the design of the system so the that future changes are less expensive.
- **Software evolution**
  - Continual change from a lesser, simpler or worse state to higher or better state.
  - All activities carried out to effect changes in requirements are put under the category of evolution.(Bennett and Xu defn)
  - Creating new but related design from existing ones for the purpose of supporting new functionalities, increasing system perfomance and porting it to different operating system.

Rules of Software Evolution (Lehman's laws)

- Continuing change: system needs to be modified to satisfy emerging needs of users.
- Increasing complexity: Additional work must be done to explicitly reduce the complexoty of a system.
- Self-regulation: The measures of products and processes that are produced during the evolution follow close to normal distributions.
- Conservation of organizational stability: The average amount of additional effort needed to produce a new release is almost the same.
- Conservation of familarity: As a system evolves all kinds of personnel, namely devlopers and users, it must gain a desired level of understanding of the system's content and behavior to realize satisfactory evolution.
- Continuing growth: As time passes, the functional content of a system is continually increased to satisfy user needs.
- Declining quality: the quality of a system will appear to be declining unless it is rigorously maintained and adapted to operational environment changes.

- Feedback system: The system evolution process involves multi-loop, multi-agent and multi-level feedback among different kinds of activities.

Nature of Software Maintenance

- Corrective Maintenance: Involves identifying and fixing software defects or errors that are discovered after the system is deployed. It aims to restore the software to it's intended functionality and address any issues reported by users or detected through testing.
- Adaptive Maintenance: Focuses on modifying the software to accommodate changes in the environment. It may includes adapting the software to work with new hardware and software platforms, operating systems, databases or external dependencies.
- Perfective Maintenance: Involves improving the software's functionality, performance and maintainability without altering its external behavior. It aims to enhance the software based on evolving user needs, industry standards or technological advancements.
- Preventive Maintenance: Aims to reduce the occurrence of future defects by proactively identifying and addressing potential issues. Includes code refactoring, performance optimization and security enhancements to mitigate risks and improve overall quality.
- Documentation and Knowledge management: Involves maintaining up-to-date documentation and knowledge about the software is crucial for effective maintenance.
- Impact Analysis: conducting impact analysis before making changes helps to assess the potential effects of modification on the system.
- Regression testing: type of testing that's performed after making modifications or enhancements to the software to ensure that the changes don't introduce new defects or break existing functionality.
- Change Management: Includes activities such as change request management, change prioritization, change approval, and tracking changes to ensure proper documentation and coordination among stakeholders.
- Resource and Time constraints: Maintenance teams must prioritize tasks effectively.
- Continous Improvement: software maintenance continously improves the system.

Need for Software Maintenance

- Bug fix and Issue resolution
- Enhanced performance and Optimization
- Adaptation to Change Requirements
- Security updates and Vulnerability Mitigation
- Compatibility and Integration
- Regulatory and Compliance
- User support and Customer Satisfaction
- Long-Term cost savings
- Knowledge Preservation
- Continous Improvement and Innovation

Maintenance Cost

- Personnel Costs
- Bug fix and Issue resolution
- Enhancements and Updates
- Testing and QA
- Documentation and Knowledge management
- Hardware and Software upgrades
- Security measures
- Downtime and Loss of Productivity
- License and Subscriptions
- Vendor support and Maintenance Contracts
- Compliance and Regulatory costs

Software Evolution Models and Processes

- `Software Maintenance Life Cycle(SMLC)`
    - Features
        - Understanding the code
        - Modifying the code
        - Revalidating the code
    - Change mini-cycle models
        - Change request -> Analyze and plan change -> Implement change -> Verify and Validate -> Documentation change.
    - Staged model of maintenance and evolution
        - Describes how long-lived software evolves
        - Stages
            - Initial development
            - Evoluion
            - Servicing
            - Phaseout

Software Configuration Management

- Discipline of managing and controlling changes in the evolution of software systems.
- The goal of SCM is to manage and control various extensions, adaptations and corrections that are applied to a system over it’s lifetime.
- Elements:
    - Identification of software configurations
    - Control of software configurations
    - Auditing software configurations
    - Accounting software configuration status

Reengineering

- Implies a single cycle of taking an existing system and generating from it a new system.
- Single cycle of software evolution.
- Reengineering = Reverse engineering + Forward engieering.

Legacy Systems

- An old program that continues to be used because it still meets the user’s needs.
- Managing legacy systems:
    - Freeze
    - Outsource
    - Carry on maintenance
    - Discard and redevelop
    - Wrap
    - Migrate

Impact Analysis

- Task of estimating the parts of the software that can be affected if a proposed change is made.
- Impact analysis techniques:
    - **Traceability analysis**: requirements, design, code and test cases related to the feature to be changed are identified.
    - **Dependency (source code) analysis**: attempts to assess the afects of change on semantic dependencies between program entities. `Call graph based` and `Dependency graph based` analysis.

Refactoring

- Process of making a change to the internal structure of software to make it easier to understand and cheaper to modify without changing it’s observable behavior.

Softwaare Reuse

- Using existing artifacts or software knowledge during the construction of a new software system.
- Types of reusable artifacts:
    - Data reuse
    - Architectural reuse
    - Design reuse
    - Program reuse

Benefits of Reuse

- Increased reliability
- Reduced process risk
- Increase productivity
- Standards compliance
- Accelerated development
- Improve maintainability
- Reduction in maintenance time and effort

Categories of Software

- System softwares
    - OS
    - Device drivers
    - Utilities
- Application softwares
    - Productivity software
    - Communication and collabration software
    - Multimedia software
    - Graphics and design software
    - Entertainment software
    - Educational software
    - Financial software
    - Enterprise software
    - Embedded software
    - Middleware
    - Development tools

Chapter 2: Issues in Software Maintenance and Evolution

Technical issues

- Code complexity
- Legacy code
- Lack of documentation
- Poor software design
- Inadequate testing
- Dependency management
- Performance and scalability
- Security vulnerability
- Integration challenges
- Data migration and compatability

Management issues

- Resource allocation
- Priortization and planning
- Change and stackholder management
- Collabaration and coordination
- Continous improvement
- Budget constraints
- Documentation and knowledge management
- Risk and vendor management

Cost estimation

- Key factors and techniques to consider:
    - Understanding the scope
    - Historical data
    - Categorizing maintenace tasks
    - Size and complexity
    - Resource avaliability
    - Change management
    - Software metrics

    - Estimation techniques:
        - Expert judgment
        - Analogy based estimation
        - Parametric estimation models (Constructive Cost Model (COCOMO) or Software Lifecycle Management (SLIM))
    - Risk assessment
    - Iterative estimation
    - Contingency planning
    - Docs
    - Monitoring and control
    - Stackholder involvement

Chapter 3: Reverse engineering

- Process of analyzing a subject system to identify the system's components and their interrelationships and create representations of the system in another form or at a higher level abstraction.
- Process of analyzing a software system or component to understand its design, functionality, and implementation details, often without access to its original source code or documentation.
- Extracting information from the compiler or executable form of the software.

Why do we need reverse engineering?

- The original programmers have left the organization
- The language of implementation has become obsolete
- Insufficient documentation
- The business relies on software, which many can't understand
- Company lacks access to all the source code.
- System requires adaptations and/or enhancements
- Software doesn't operate as expected

Key steps in reverse engineering

- IEEE reverse engineering standards for Software Maintenance:
    - Partition source code into units
    - Define the meaning of those units -> identify functional units
    - Create the input and output schematics of the units identified before
    - Describe the connected units
    - Describe the system application
    - Create an internal structure of the system

Reverse Engineering concepts

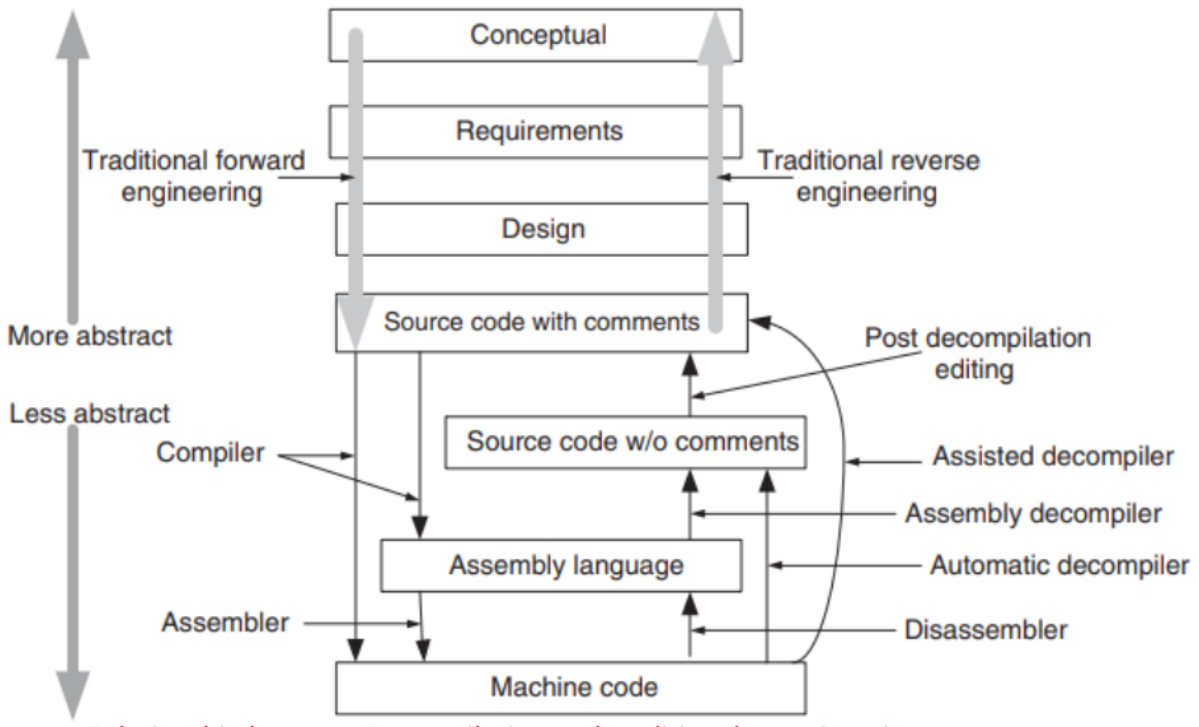
- **Define objectives:** determine what specific information or knowledge you want to extract or understand from the target artificat.
- **Obtain the target artificat:** aquire the target artificat or product that you intend to reverse engineer.
- **Analysis:**
    - `Observation` : observe the physical and digital characteristics, interfaces and external behavior.
    - `Disassembly` : disassemble the physical artifact to examine internal components, connection and circuitry.
    - `Instrumentation` : use specialized tools or equipments to gather data about the artifacts behavior.
    - `Dynamic analysis` : run the artifact and monitor its runtime behavior to gain insights into its operations, data flow and control flow.
    - `Static analysis` : analyze static characteristics of the software without executing the artifact.
- **Documentation:** documenting findings and observations during the analysis phase.
- **Reconstruction:** creating architectural diagrams, flowcharts or pseudo-code based on the information from the analysis stage.
- **Understanding and Enhancement:** identifying areas of optimization, bug fixing or implementing new feature once we have a clear understanding of the artifact.
- **Legal and Ethical considerations:** ensuring that we comply with relevant laws, licensing agreements, and intellectual property rights.

Reverse engineering tools

- Software applications or utilities that aid in the process of analyzing and understanding an artifact.
- Commonly used tools:
    - `Decompilation` provides a limited kind of program comprehension in reverse engineering. However compilation isn’t considered part of the forward engineering since it’s automatic step.

    - Types of decompilers:

        - `Assembly decompiler` : Assembly language -> Source code w/o comments
        - `Automatic decompiler` : Machine code -> Source code w/o comments
        - `Assisted decompiler` : Machine code -> Source code with comments

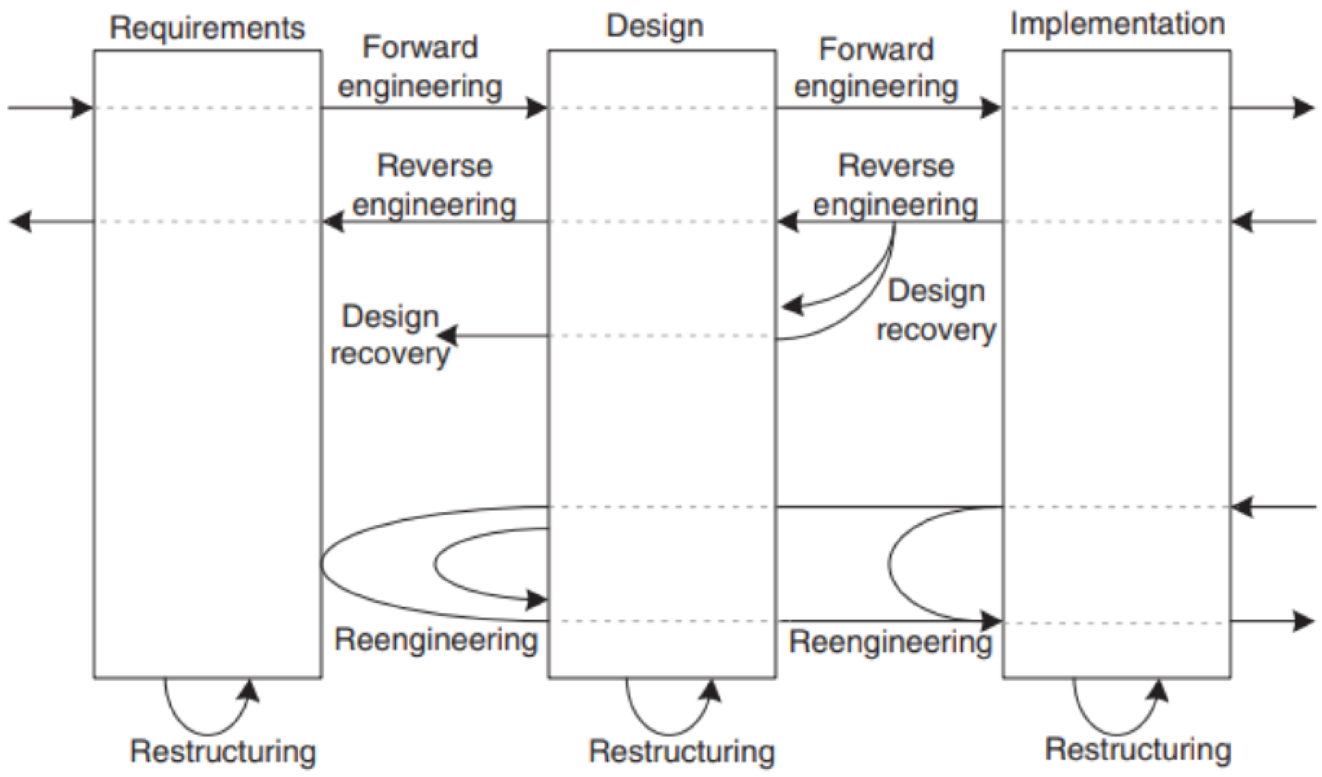


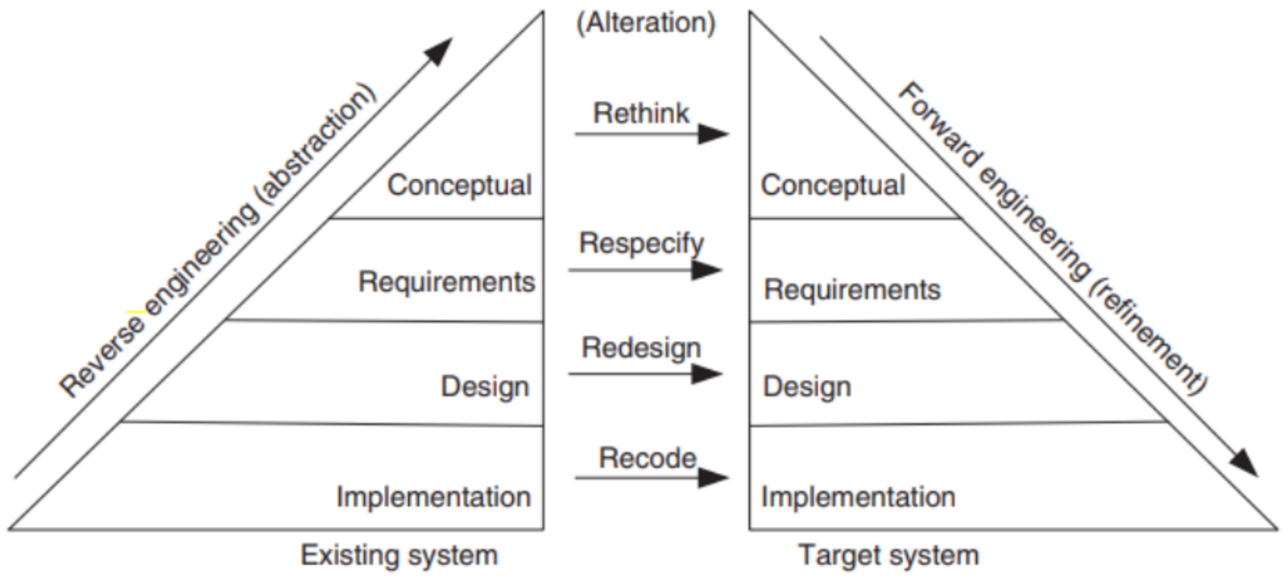
Relationship between Decompilation and traditional reengineering

Name	Type	Open source	Platforms
IDA Pro	Disassemblers and Decompilers	No	Windows, Linux and MacOS
Ghidra	Disassemblers and Decompilers	Yes	Windows, Linux and MacOS
Hopper	Disassemblers and Decompilers	Yes	MacOS and Linux
OllyDbg	Debuggers	No	Windows
WinDbg	Debuggers	No	Windows
GDB	Debuggers	Yes	Windows, Linux and MacOS
Wireshark	Network analysis tools	Yes	Windows, Linux and MacOS
Burp Suite	Network analysis tools	No	Windows, Linux and MacOS
JD-GUI	Decompiler tools	Yes	Windows, Linux and MacOS
jadx	Decompiler tools	Yes	Windows, Linux and MacOS

Name	**Type**	**Open source**	**Platforms**
dnSpy	Decompiler tools	Yes	Windows
SonarQube	Static analysis tools	No	Windows, Linux and MacOS
FindBugs	Static analysis tools	Yes	Windows, Linux and MacOS
AFL	Dynamic analysis tools	Yes	Windows, Linux and MacOS
Peach	Dynamic analysis tools	Yes	Windows, Linux and MacOS
Angr	Dynamic analysis tools	Yes	Windows, Linux and MacOS
S2E	Dynamic analysis tools	Yes	Windows, Linux and MacOS
Radare2	Reverse Engineering suite	Yes	Windows, Linux and MacOS
Binary Ninja	Reverse Engineering suite	No	Windows, Linux and MacOS
x64dbg	Reverse Engineering suite	Yes	Windows

Relationship between reengineering and reverse engineering





Benefits of Reverse Engineering

- Understanding legacy systems
- Interoperability and Compatibility
- Security analysis
- Competitive analysis
- Knowledge Transfer and Learning

Risk of Reverse Engineering

- Legal and Ethical concerns
- Incomplete or Inaccurate understanding
- Time and Resource intensive
- Potential damage to the systemi

Reuse

- The practice of utilizing existing software artifacts, such as code modules, libraries, frameworks, designs, or components, in the development of new software systems.
- `Reusability` : the inherit quality of software artifacts that enables them to be easily reused in different contexts. The design and implementation characteristics that facilitate the effective and efficeint reuse of software components.

Benefits of Reuse

- Increased productivity
- Improved quality
- Faster time to market
- Cost savings

Types of Reuse

- **Code Reuse:** Reusing code snippets, libraries, or frameworks can significantly speed up development. Developers can leverage existing functionality, algorithms, or utility functions to achieve common tasks.
- **Design Reuse:** Reusing design patterns, architectural styles, or system structures promotes consistency and best practices. Well- established design patterns, such as MVC or Observer, can be reused across different projects.
- **Component Reuse:** Reusing entire software components or systems, such as microservices or software-as-a-service (SaaS) offerings, allows developers to integrate pre-built functionality into their applications.

Key characteristics of reusable software artifacts

- Modularity
- Abstraction
- Loose coupling
- Standardization

How to design reusable software?

- We can maximize reusability of a software artifact using the following principles and best practices during design phase:

    - Encapsulation
    - Separation of concerns
    - Interface definition
    - Documentation

Maintenance measures

- **Defect Density:** the number of defects found in a given period usually per unit code size. It helps assess the quality of the software and identify areas that require improvement.
- **Mean Time to Repair (MTTR):** average time taken to fix reported defects or issues. It provides insights into the efficiency and effectiveness of the maintenance process and helps in setting realistic expectations for issue resolution.
- **Mean Time Between Failures (MTBF):** calculates the average time between consecutive failures occurring in the software system. It helps assess the system's reliability and provides an indication of its stability and robustness.
- **Maintenance Effort:** quantifies the resources expanded on software maintenance activities. Helps evaluate the overall cost and maintenance and useful for budgeting and resource allocation purposes.
- **Change Request Change:** measures the frequency of change requests or modifications made to the software. Indicates adaptability of the software to evolving needs and level of stackholder engagement.
- **Customer satisfaction:** the satisfaction level of software users or customers with the maintenance activities.
- **Documentation coverage:** extent and quality of documentation avaliable for the software system. Includes user manuals, technical specifications, and support documentation.
- **Regression test coverage:** the percentage of the system covered by regression tests.
- **Mean Time to Discover (MTTD):** the average time taken to detect or discover defects after they occur. Indicates the ability of detecting and reporting defects in the system.
- **Technical Debt:** the accumulated "debt" resulting from shortcuts or suboptimal design decisions made during development or maintenance. It helps gauge the level of effort required to address technical debt and maintain the software's long-term health.

Chapter 4: Configuration Management

What’s Configuration Management (CM)?

- Managing and controlling the numerous corrections, extensions, and adaptations that are applied to a system over its lifetime.

What’s Software Configuration Management (SCM)?

- Process of managing and controlling changes to software systems throughout their lifecycle.
- Managing and controlling the changes made to software systems, including their documentation, code, and related artifacts.
- Systematic management of Software Configuration Items (SCIs) and associated documentation, ensuring that changes are planned, implemented, tested, and tracked in a controlled manner.

Benefits of Software Configuration Management

- Ensures that development processes are traceable and systematic so that all changes are precisely managed.
- Enhances the quality of the delivered system and the productivity of the maintainers.

Objectives of Software Configuration Management

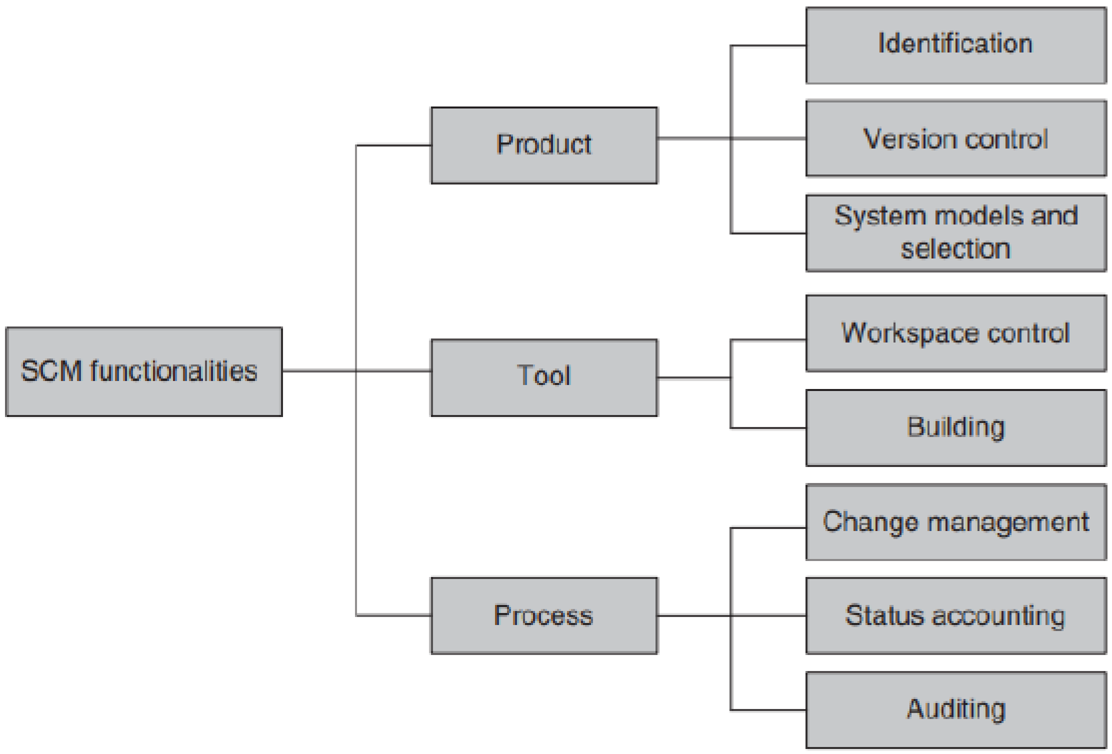
- Uniquely identify every version of every software at various points in time.
- Retain past versions of documentations and software.
- Provide a trail of audit for all modifications performed.
- Throughout the software life cycle, maintain the traceability and integrity of the system changes.

Benefits of effective SCM

1. Confusion is reduced and order is established.
2. To maintain product integrity, the necessary activities are organized.
3. Correct product configurations are ensured.
4. Quality is ensured and better quality software consumes less maintenance efforts.
5. Productivity is improved, because analysts and programmers know exactly where to find any piece of the software.
6. Liability is reduced by documenting the trail of actions.
7. Life cycle cost is reduced.
8. Conformance with requirements is enabled.
9. A reliable working environment is provided.
10. Compliance with standards is enhanced.

11. Accounting of status is enhanced.

SCM Spectrum of Functionality



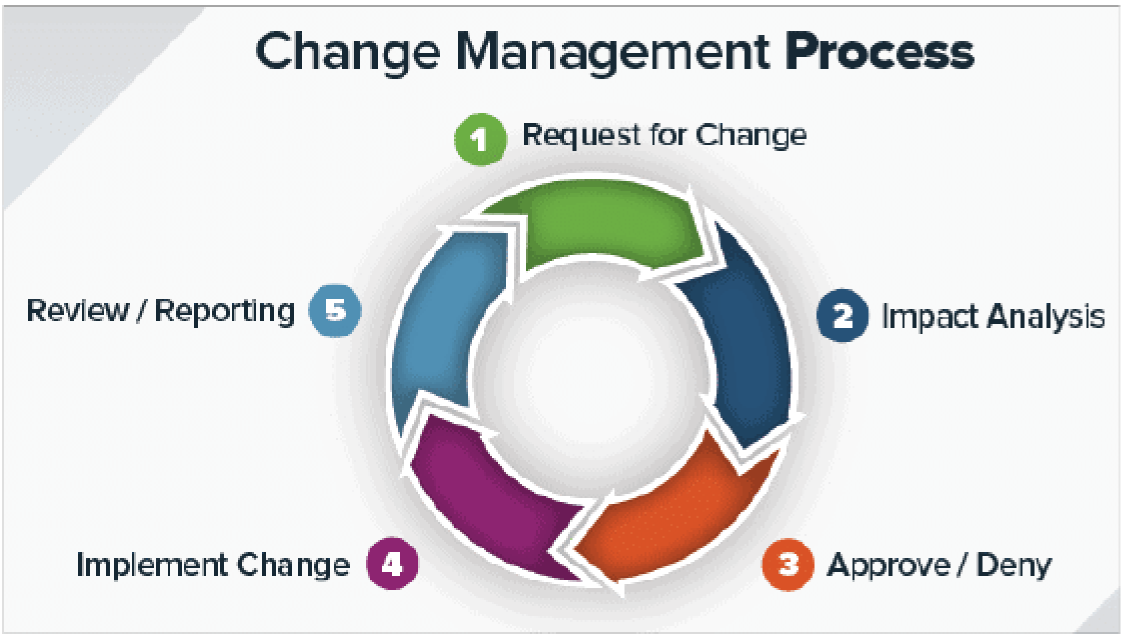
What are the things included Change Request history?

- Why are changes made?
- When are the changes made?
- Who make the changes?
- What changes are made?

Change Management and Change Control

- **Change Management**:
    - The process of managing and controlling changes made to software systems after they have been deployed and are in use.
    - It involves planning, evaluating, implementing, and tracking changes to ensure they are carried out in a controlled and systematic manner.
    - The goal of change management in software maintenance is to minimize the risks associated with changes, ensure the stability and reliability of the software, and meet the

evolving needs of users and stakeholders.



- Change Control:
    - The process of managing changes to software systems, applications, or codebases.
    - It involves a systematic approach to controlling modifications to software to ensure that they are properly evaluated, approved, implemented, and tracked.
    - The primary objective of change control is to minimize risks, maintain software quality, and ensure that changes are effectively managed throughout the development lifecycle.
    - Process:
        - `Change identiifcation`
        - `Change request submission`
        - `Change evaluation and impact analysis`
        - `Change approval or rejection`
        - `Change implementation`
        - `Change testing`
        - `Change deployment`
        - `Change monitoring and review`
        - `Change documentation and communication`
        - `Change closure and post-implementation review`

Management and organizational issues

- Lack of clear Configuration Management Plan
- Inadequate Change Control Processes
- Lack of Collabration and Communication
- Resistance to Change
- Tooling and Infrastructure challenges
- Scalability and Flexibility
- Training and Skill gaps

Management Responsibilities

- Establishing Configuration Management Policies
- Developing a Configuration Management Plan
- Resource Allocation
- Providing Training and Education
- Enforcing Change Control Processes
- Monitoring and Auditing
- Continous Improvement
- Collabration and Communication
- Risk Management

Chapter 5: Building and Sustaining Maintainability

Maintenance Tools

- **Criteria for selecting tools**:
    - Functionality
    - Integration
    - Usability
    - Scalability
    - Customization and Extensibility
- **Taxonomy of tools**:
    - Bug Tracking and Issue Management Tools:
        - Jira
        - Bugzilla
    - Code Ananlysis and Quality Assurance Tools:
        - SonarQube
        - EsLint
    - Performance Monitoring and Profiling Tools:
        - New Relic
        - Apache JMeter
    - Documentation and Generation Tools:
        - Doxygen
        - Sphinx

Quality Assurance

- The process and activities undertaken to ensure that software systems continue to meet the desired quality standards and perform as expected after deployment.
- It involves monitoring, evaluating, and improving the system to identify and address any issues or deficiencies that may impact its functionality, performance, security, or user experience.

- **Key Aspects**:
    - Defect Tracking and Management
    - Regression Testing
    - Performance Monitoring and Optimization
    - Security Assessment
    - User Experience Evaluation
    - Documentation Review and Maintenance
    - Continous Improvement

Program Understanding and Reverse Engineerig Testing

- **Program Understanding**:
    - Process of comprehending the source code, architecture, and design of a software system.
    - It involves analyzing the codebase to gain insights into how the system works, identifying its components and their interactions, and understanding the overall system flow.
    - Program understanding is crucial for maintenance activities, as it helps developers and testers identify areas that require modification or improvement.
    - Techniques:
        - `Code Reading`
        - `Code Documentation`
        - `Static Analysis`
        - `Dynamic Analysis`
- **Reverse Engineering Testing**:
    - Reverse Engineering: analyzing and understanding a software system by examining its executable code, binary files, or other artifact. It is commonly used when the source code is unavailable, incomplete, or poorly documented.
    - Reverse Engineering Testing: process of uncovering system's functionality, behavior, and underlying algorithims.
    - Techniques:
        - `Decompilation`
        - `Disassembly`
        - `Dynamic Analysis`
    - Benefits and Applications:
        - `Legacy System Understanding`
        - `Security Analysis`
        - `Interoperability and Integration`

Chapter 6: Software Administration

- **System Logs**:
  - Record important `events`, `activities`, and `errors` that occur within a software system.
  - They serve as a valuable source of information for troubleshooting issues, identifying errors, and monitoring system behavior.
- **Updates and Patches**:
  - `Updates` often include bug fixes, performance improvements, and new features.
  - `Patches` specifically focus on addressing security vulnerabilities.
- **Configuration changes**:
  - Involve modifying the settings and parameters of a software system to adapt it to evolving requirements or optimize its performance.
- **Backups**:
  - Essential for data protection and disaster recovery.
  - It involve creating copies of critical system data and storing them securely in separate locations.
  - Automating backup process:
    - `Define backup requirements`: Start by determining what data and system components need to be backed up.
    - `Choose a backup solution`: Select a backup solution that meets your requirements.
    - `Establish a backup schedule`: Define a backup schedule that suits your needs.
    - `Automate backup execution`: Set up automation scripts or tools to initiate and manage the backup process.
    - `Include error handling and notifications`: Implement error handling mechanisms and notifications to ensure the reliability of the backup process.
    - `Monitor backup status`: Regularly monitor the status of backup operations to ensure they are running as expected.
    - `Test backup recovery`: Periodically test the backup recovery process to verify the integrity and reliability of the backup data.
    - `Secure backup storage`: Ensure that the backup data is securely stored in a separate location from the production system.
    - `Document the backup process`: Maintain clear documentation of the backup process, including the automation scripts, schedules, and any configurations or customizations.

Installing and Configuring Hardware and Software

- **Hardware installation**: involves physically setting up the `hardware`, `connecting cables and peripherals,` and `ensuring proper power supply and cooling`.
- **Operating system installation**: Software administrators are often involved in installing and configuring the operating system (OS) that will serve as the foundation for running the software system.
- **Dependency installation**: Software systems often have dependencies on additional `software components`, `libraries`, or `frameworks`.
- **Software installation**: Software administrators handle the installation of the actual software system they are managing.
- **Configuration and customization**: After the software installation, software administrators configure the system to align with the specific requirements of the organization or users.
- **Integration with other systems**: it involve `setting up communication protocols`, `establishing data exchange formats`, `configuring APIs or web services`, or `implementing authentication` and `authorization mechanisms`.
- **Testing and validation**: Once the hardware and software components are installed and configured, software administrators perform testing and validation to ensure that the system functions as intended.
- **Ongoing maintenance and updates**: Software administrators are responsible for managing ongoing maintenance tasks, such as `applying software updates`, `security patches`, and `bug fixes`.

Hardware Installation Best Practices

- Plan and prepare
- Follow manufacturer guidelines
- Proper physical setup
- Label and document
- Regular maintenance

Software Installation Best Practices

- Read documentation
- System compatibility
- Secure installation
- Centralized management
- Standardize configurations
- Regular updates and patches
- Testing and validation
- Backup and recovery
- Document changes and configurations
- Training and knowledge sharing

System Performance Tuning

- System performance tuning is an important aspect of software administration that involves optimizing the performance of a computer system to ensure efficient and smooth operation.
- **Key areas**:
    - Resource Monitoring and Analysis
    - System Configuration
    - Disk I/O Optimization
    - Memory Management
    - CPU and Process Optimization
    - Network Optimization
    - Application-Level Optimization
    - Regular Maintenance and Updates
- **System monitoring tools**:
    - top/htop
    - vmstat
    - iostat
    - sar
    - nmon
    - Windows Task Manager
    - Resource Monitor
    - Performance Monitor

Performing Routine Audits of Systems and Software

- **Steps**:
    - Define Audit Objectives
    - Establish Audit Criteria
    - Perform Security Assessments
    - Review Access Controls
    - Check Configuration Settings
    - Assess Patch Management
    - Review Logs and Monitoring
    - Evaluate Backup and Disaster Recovery
    - Check Compliance Requirements
    - Document Findings and Remediate
    - Follow-Up and Periodic Audits

How often should organizations conduct routine audits?

- The frequency of routine audits for systems and software can vary depending on factors such as `the size and complexity of the organization`, `industry regulations`, and `the level of risk associated with the systems`.
- **General guidelines for conducting routine audits**:
    - Regular Intervals

- Significant Changes
- Incident Response Triggers
- Vendor Software Updates
- Regulatory Changes
- Compliance Requirements
- Risk Assessment