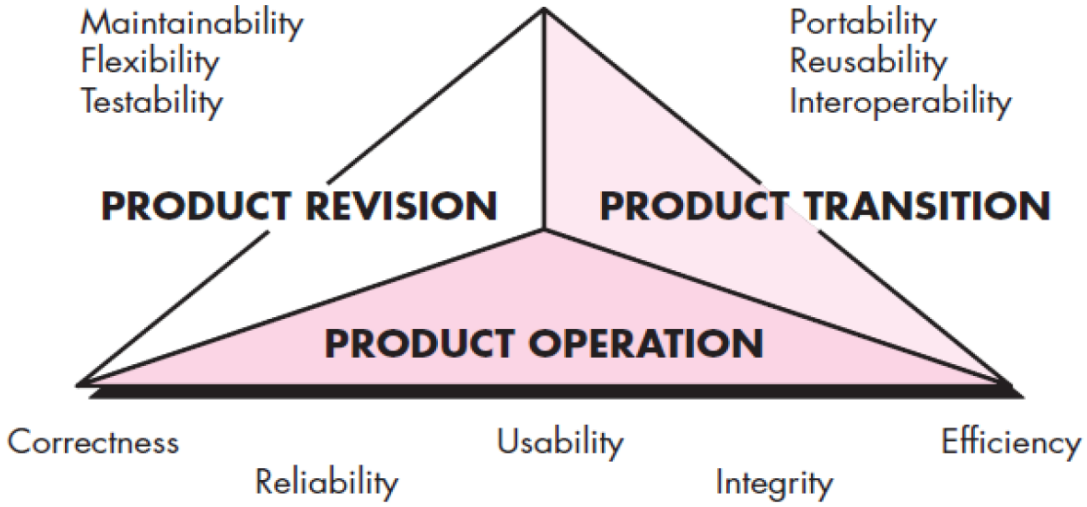
SE Module 6

27-Module 6_ Software Quality_ Software Quality Factors,-07-11-2024

Software Quality Management :

- **Software Quality** refers to software that is reliable, bug-free, meets user requirements, is delivered on time and within budget, and is easy to maintain. It incorporates both functional and structural qualities, ensuring the software performs as expected while being robust and manageable.
- **Quality Dimensions** (David Garvin):
    1. **Performance**: Does the software deliver all required functions and content efficiently while adding value to the user?
    2. **Feature Quality**: Does the software offer surprising and delightful features for first-time users?
    3. **Reliability**: Does it function without failure, and is it available when needed?
    4. **Conformance**: Does it meet local, external, and internal design and coding standards?
    5. **Durability**: Can the software be maintained or updated without causing new errors or degrading performance?
    6. **Serviceability**: How quickly can defects be corrected and changes made to the software?
    7. **Aesthetics**: Does the software provide a smooth and attractive user experience?
    8. **Perception**: How do users perceive the software quality, influenced by their experiences and expectations?
- **Software Quality Factors**:
    1. **McCall's Model** divides quality factors into:
        - **Product Operation**: Focuses on the daily use of the software, including correctness, reliability, efficiency, integrity, and usability.

- **Product Revision**: Concerned with maintainability, flexibility, and testability to ensure the software can evolve without major issues.
- **Product Transition**: Involves portability, reusability, and interoperability to ensure the software adapts well to different environments and projects.



2. **ISO 9126** outlines six key quality attributes:
    - **Functionality**: Meets user needs with suitability, accuracy, security, and compliance.
    - **Reliability**: Available when needed, with minimal faults.
    - **Usability**: Easy to use, understand, and learn.
    - **Efficiency**: Uses system resources optimally.
    - **Maintainability**: Easy to fix and improve, ensuring stability.
    - **Portability**: Easy to adapt to different environments and systems.

- **Software Quality Assurance (SQA)** ensures that software development processes comply with quality standards. SQA includes activities such as process definition, auditing, training, and ongoing improvements. It aims to

identify weaknesses in processes and correct them to ensure continuous quality enhancement throughout the development lifecycle.

- **Capability Maturity Model (CMM)**:
  - A framework used to assess the maturity of software processes and improve their effectiveness. It consists of five levels: **Initial**, **Repeatable**, **Defined**, **Managed**, and **Optimized**. These levels guide organizations in improving their software development processes step by step.
- **CMMI (Capability Maturity Model Integration)** is an advanced version of CMM that integrates several models into one to improve overall process effectiveness. It has five maturity levels:
  1. **Initial**
  2. **Managed**
  3. **Defined**
  4. **Quantitatively Managed**
  5. **Optimizing**

Difference Between CMM and CMMI:

- **CMM** focuses on evaluating whether an organization is following specific tasks in the development process. **CMMI** not only evaluates the processes but also provides a comprehensive approach to continuous improvement across the entire software development lifecycle.

This provides a detailed overview of software quality management, quality models, and improvement frameworks to help organizations optimize their software development processes.

29-Software maintenance, Maintenance Process Models,-14-11-2024

Software Maintenance :

- **Software Maintenance** refers to the modification and updates made to software after its delivery to correct errors, improve performance, and adapt to changes in user requirements, hardware, or software. It ensures that

software continues to function as expected over time and can adapt to changes in the environment in which it operates.

- **Importance**: Maintenance is necessary to keep the system running smoothly and ensure it remains useful. It helps improve system efficiency, enhances features, optimizes performance, and addresses any issues that may arise post-deployment.
- **Categories of Software Maintenance**:
    1. **Correcting errors**: Fixing bugs or issues identified after the software is in use.
    2. **Enhancing features**: Adding new features or improving existing functionality.
    3. **Removing outdated features**: Deleting obsolete functions that no longer serve a purpose.
    4. **Optimizing code**: Refactoring or optimizing the software for better speed and performance.
- **Software Re-engineering**:
    - Re-engineering focuses on improving the maintainability of older or legacy systems without changing the overall functionality. This involves activities like re-documenting the system, restructuring the code, translating it to a newer programming language, and improving data storage structures.
- **Business Process Reengineering (BPR)**:
    - BPR involves redesigning business processes to improve key areas like cost reduction, time management, and quality improvement. The process includes defining business goals, identifying critical processes, evaluating existing processes, redesigning them, prototyping new workflows, and refining them for integration into the system.
- **Re-engineering Steps**:
    1. **Inventory Analysis**: Identifying all software applications in use and selecting those that need re-engineering based on their criticality and maintainability.

2. **Document Reconstruction**: Updating and improving system documentation to ensure clarity and usability for maintenance.
3. **Reverse Engineering**: Analyzing the existing software to generate a higher-level abstraction or representation of its design and components.
4. **Code Reconstruction**: Revising and restructuring the code to remove inefficiencies, improve readability, or update it to newer standards.
5. **Data Reconstruction**: Redefining and organizing the data architecture for improved quality and efficiency.
6. **Forward Engineering**: Using the recovered design information to enhance or rebuild the system, adding new features and improving overall performance.

- **Reverse Engineering**:
  - Involves analyzing an existing software system to understand its components, functionality, and interrelationships. It aims to facilitate maintenance by improving system documentation and helping identify hidden or complex parts of the system that require updating or modification.
- **Forward vs. Reverse Engineering**:
  - **Forward Engineering** is the process of building new software based on client specifications, while **Reverse Engineering** focuses on improving existing systems through modifications and enhancements.
- **Re-engineering Cost Factors**:
  - The costs associated with re-engineering depend on the quality of the current software, available tools for re-engineering, data conversion needs, and the availability of skilled personnel to carry out the work.
- **Benefits of Re-engineering**:
  - Re-engineering reduces risks compared to building new systems from scratch, lowers costs, helps uncover hidden business rules within existing systems, and allows organizations to maintain the expertise of their current staff while upgrading their systems.

- **Cost-Benefit Analysis** for Re-engineering:
  - A cost-benefit model helps evaluate whether re-engineering is worth the investment by comparing the current and predicted costs of maintenance, business value, and the overall re-engineering costs. This model helps organizations make informed decisions about re-engineering legacy systems.

Software Maturity Index

1) A legacy system has 940 modules. The latest release required that 90 of these modules be changed. In addition, 40 new modules were added and 12 old modules were removed. Compute the software maturity index for the system.

Solutions:

SMI = M_T– (F_a + F_c + F_d)/M_T, Where

SMI – is the Software Maturity Index value

M_T – is the number of software functions/modules in the current release

F_c – is the number of functions/modules that contain changes from the previous release

F_a – is the number of functions/modules that contain additions to the previous release

F_d – is the number of functions/modules that are deleted from the previous release.

SMI =(940 - (40 + 90 + 12)) /940

=0.8489

=0.849