Assignment #1

Subject: Software Quality Assurance

Name: Eman Shahbaz

Roll No.: BITF22M516

Boehm's Quality Model in Software Quality Assurance

Introduction

Software Quality Assurance (SQA) ensures that a software product meets defined quality standards and fulfills both user expectations and technical requirements. Since quality is a broad and multi-dimensional concept, various models have been proposed to define and measure it systematically. One of the earliest and most influential models was introduced by **Barry W. Boehm in 1978**, known as **Boehm's Quality Model**.
This model provides a hierarchical framework that defines software quality in terms of high-level goals, intermediate attributes, and primitive measurable characteristics. It connects what users expect from the software with what developers can measure and improve.

Structure of Boehm's Quality Model

Boehm's model consists of three hierarchical levels:

1. **High-Level Characteristics (User-Oriented Goals)**
2. **Intermediate-Level Characteristics (Developer-Oriented Attributes)**
3. **Primitive Quality Attributes (Measurable Factors)**

1. High-Level Characteristics

At the top level, Boehm identified three main quality goals from a user's perspective:

- **As-is Utility**: Refers to the extent to which the software fulfills its intended purpose and provides the required functionality to the user. For example, an accounting software must correctly handle financial transactions.
- **Maintainability**: Refers to the ease with which software can be corrected, modified, or enhanced in response to changing requirements. A maintainable system allows developers to add new features or fix bugs without introducing further errors.
- **Portability**: Refers to the ability of software to be adapted and run in different hardware or software environments with minimal effort. For example, a web application should be usable across multiple browsers and operating systems

2. Intermediate-Level Characteristics

These are quality attributes that contribute to the achievement of high-level goals:

- **Portability**: Ensures the adaptability of the software to different environments such as operating systems, hardware platforms, or networks.
- **Reliability**: Refers to the ability of the software to consistently perform its required functions without failure. For instance, a medical monitoring system must work without interruption.
- **Efficiency**: Measures how well the software utilizes system resources such as processing power, memory, and response time.
- **Usability**: Represents how easy and user-friendly the software is for end-users. This includes factors like simplicity of navigation, clear interfaces, and quick learnability.
- **Testability**: Refers to the extent to which software can be tested effectively. Well-structured and modular systems are easier to test for correctness.
- **Understandability**: Refers to how easily the software design and code can be understood by developers. Good documentation, coding standards, and clear design improve understandability.
- **Flexibility**: Represents the ability of the software to adapt to future changes or new requirements without excessive rework.

3. Primitive Quality Attributes

At the bottom level are measurable attributes that provide the foundation for intermediate characteristics. These are directly observable and can be assessed in practice:

- **Accuracy**: The degree to which the software produces correct and precise results.
- **Completeness**: Ensures that all required functions and features are fully implemented.
- **Consistency**: Maintains uniformity in behavior and avoids contradictions in functionality.
- **Conciseness**: Refers to the avoidance of unnecessary complexity in code, design, or documentation.
- **Traceability**: The ability to trace requirements throughout different stages of development, such as design, implementation, and testing.
- **Modularity**: Breaking the system into smaller, independent, and reusable components that simplify development and maintenance.
- **Instrumentation**: Providing facilities for monitoring, tracking performance, and diagnosing errors within the system.

Importance of Boehm's Quality Model in SQA

- It provides a comprehensive view of software quality by considering both user expectations and developer concerns.
- It serves as a framework for quality evaluation, linking measurable attributes to high-level goals.
- It helps identify weak areas in software (for example, low usability due to poor documentation) and provides guidance for improvement.
- It laid the foundation for later quality models such as McCall's Quality Model and ISO 9126**.**

Conclusion

Boehm's Quality Model is one of the earliest and most significant frameworks for defining and measuring software quality. It organizes quality into **high-level goals**, **intermediate characteristics**, and **primitive attributes**, making it easier to evaluate whether a software product is useful, reliable, maintainable, and portable. In Software Quality Assurance, this model provides a systematic method to ensure that software meets both technical standards and user satisfaction, making it an essential tool in the field of software engineering.

Difference Between Agile and Kanban

Agile

- **Definition**:
  Agile is a software development and project management methodology that focuses on iterative progress, collaboration, and flexibility. It delivers software in small increments, known as **sprints**, typically lasting 1–4 weeks.
- **Approach**:
  Agile works in time-boxed iterations where teams plan, develop, test, and review software features regularly.
- **Roles**:
  Agile (for example, Scrum) defines specific roles such as:
  - **Product Owner**: responsible for requirements.
  - **Scrum Master**: facilitates the process.
  - **Development Team**: builds the product.
- **Planning**:
  Each sprint requires upfront planning of tasks that should be completed within that iteration.
- **Metrics**:
  Agile teams measure performance using velocity, sprint burndown charts, and sprint goals**.**

- **Flexibility**:
  Changes are typically allowed at the end of each sprint, not during.
- **Focus**:
  Agile emphasizes customer collaboration, frequent delivery of working software, and adaptability to changing requirements.

Kanban

- **Definition**:
  Kanban is a workflow visualization and management method that originated from Lean Manufacturing and is widely used in software development. It can be applied within Agile or independently.
- **Approach**:
  Kanban is based on a continuous flow of work. Tasks are represented as cards on a **Kanban board**, moving through columns such as *To Do → In Progress → Done*.
- **Roles**:
  Kanban does not require specific roles; it can be used with existing team structures.
- **Planning**:
  Kanban requires **minimal upfront planning**. Tasks can be added to the board anytime as work capacity allows.
- **Metrics**:
  Performance is measured with cycle time (time taken to complete a task) and **throughput** (number of tasks completed in a period).
- **Flexibility**:
  Highly flexible – new tasks or changes can be introduced at any time without waiting for an iteration to end.
- **Focus**:
  Kanban focuses on visualizing workflow, limiting work-in-progress (WIP), identifying bottlenecks, and improving efficiency.