Slide 02

The term "quality" is often used in various contexts, and it can mean different things to different people. This slide presents five different views or perspectives on what quality means:

- Transcendental View: According to this view, quality is challenging to define, but it is recognizable if present. It is subjective and depends on the individual's perception and experience. For example, a person may consider a product to be of high quality if it meets their expectations and performs well.
- User View: This view considers quality as meeting the user's needs and expectations. It is essential to understand the user's requirements and deliver a product that satisfies those requirements. For example, a software application may be of high quality if it meets the user's needs and is easy to use.
- Manufacturing View: This view sees quality as conformance to process standards. It involves defining the processes and procedures that are necessary to produce a product with consistent quality. For example, a product may be considered of high quality if it meets specific manufacturing standards.
- Product View: According to this view, quality is the inherent characteristics of a product that results in an improved product. It involves identifying the essential features and characteristics that contribute to the product's quality. For example, a car may be considered of high quality if it has a reliable engine, comfortable seats, and good handling.
- Value-based View: This view considers quality as the willingness of the customer to pay for a product. It is based on the customer's perception of the product's value and their willingness to pay a premium price for it. For example, a luxury car may be considered of high quality if customers are willing to pay a high price for it.

In summary, the term "quality" is subjective and can mean different things to different people. The five different views presented in this slide provide different perspectives on what quality means. These views include the Transcendental View, User View, Manufacturing View, Product View, and Value-based View.

Slide 03

The slide describes the roles and responsibilities of consumers and producers in software development and maintenance.

Consumers are concerned with the external behavior of the software, meaning they are focused on the software's overall functionality and how it behaves in the user's environment. They are primarily concerned with the output or results of the software and whether it meets their needs. The consumers of software can be categorized into three groups: Customers, Users, and Non-human Users.

- Customers are the ones who pay for the software and have a vested interest in its performance and reliability.
- Users are the ones who use the software to accomplish specific tasks.
- Non-human users are software systems that use the software as an input or output to perform specific tasks.

On the other hand, producers are individuals or organizations involved in the development, management, maintenance, marketing, and services of software. They are concerned with the internal characteristics of the

software, meaning they are focused on the underlying code, architecture, and design of the software. Producers are primarily responsible for creating, testing, and maintaining the software, ensuring it meets the quality standards, and continuously improving it to meet the users' needs.

In summary, the roles and responsibilities of consumers and producers in software development and maintenance are different. Consumers are concerned with the external behavior of the software, while producers are concerned with the internal characteristics of the software. Consumers are primarily responsible for defining the requirements and expectations of the software, while producers are responsible for creating, testing, and maintaining the software to meet those requirements and expectations.

Slide 04

This slide describes the quality expectations of consumers, which includes users, non-human users, and customers.

The primary quality expectation of consumers is that the software system performs useful functions as specified. This means that the software should meet the user's needs and requirements, and it should perform the functions as specified in the requirements document.

Another expectation is that the software should perform the right functions as specified, which is also known as fit for use or validation. This means that the software should not only perform the specified functions but also perform them correctly.

Consumers also expect the software to perform the specified functions correctly over repeated use, long periods, and perform functions reliably, which is also known as verification. This means that the software should not only perform correctly the first time but also perform consistently over time.

Ease of use, or usability, is another important quality expectation of consumers. Consumers expect the software to be easy to use, with an intuitive user interface and clear instructions.

Non-human users also have quality expectations. They expect the software to ensure smooth operation and interaction between the software and non-human users in terms of interoperability and adaptability.

Finally, customers have quality expectations similar to users, including the cost, which is based on the value-based view. Customers expect the software to meet their needs and requirements, but they also consider the cost of the software and its value proposition.

In summary, consumers have various quality expectations from software, including the software system's functionality, fit for use, verification, usability, interoperability, and adaptability. Customers also consider the cost and value proposition when evaluating software quality.

Slide 05

This slide describes the quality expectations of producers, which includes individuals or organizations involved in the development, management, maintenance, marketing, and services of software.

Producers have a primary quality expectation to fulfill contractual obligations. This means that they are responsible for delivering the software system as per the agreed terms and conditions, including the functionality, performance, and quality of the software.

Product and service managers have the responsibility of selecting the right methodology, tools, languages, and other factors to ensure that the software is developed efficiently and effectively, meeting the quality expectations of the consumers. This means that they are responsible for ensuring that the software development process is aligned with the consumer's requirements and expectations.

Other producers, such as software service providers, maintenance personnel, and software packaging service providers, have specific quality expectations to fulfill. Usability and modifiability of software are dealt with by software service providers, ensuring that the software is easy to use and modify. Maintainability is the responsibility of maintenance personnel, ensuring that the software is easy to maintain and update. Portability is the responsibility of third-party or software packaging service providers, ensuring that the software is easy to install and run-on different platforms. Profitability and customer value are the responsibility of product marketing, ensuring that the software meets the customer's needs and expectations, and provides a value proposition that justifies the price.

In summary, producers have various quality expectations, including fulfilling contractual obligations, selecting the right methodology, tools, languages, and other factors, ensuring software usability, modifiability, maintainability, portability, profitability, and customer value. Producers must meet these quality expectations to deliver high-quality software that meets the consumers' needs and expectations.

Slide 06

What is ISO-9126, and why it was introduced?

ISO 9126 is a set of international standards for evaluating the quality of software. It was introduced by the International Organization for Standardization (ISO) in 1991 to provide a framework for assessing the quality of software products.

The ISO 9126 standard defines six quality characteristics that software should exhibit, including functionality, reliability, usability, efficiency, maintainability, and portability. These characteristics are further divided into sub-characteristics that provide a more detailed assessment of software quality.

ISO 9126 was introduced to provide a systematic approach to evaluating the quality of software, as the increasing complexity of software made it difficult to assess its quality. The standard provides a common language for developers, testers, and users to communicate about software quality, making it easier to identify and address quality issues.

ISO 9126 is widely used in the software industry as a basis for developing software quality models and metrics. It has been revised several times, with the latest version, ISO/IEC 25010, published in 2011. This version includes additional quality characteristics and sub-characteristics, reflecting the evolving needs of the software industry.

The ISO-9126 standard provides a hierarchical framework for assessing the quality of software products. It defines six top-level quality characteristics, each of which is associated with several sub-categories:

1. Functionality: This refers to the set of attributes that specify the existence of a set of functions and their specified properties. Functions are implied or stated, and this characteristic is further divided into four sub-categories:
    - Suitability: The degree to which the software meets specified requirements and needs.
    - Accuracy: The degree to which the software produces correct results or performs the intended functions.
    - Interoperability: The ability of the software to interact with other systems and software in a given environment.
    - Security: The ability of the software to protect data and resources from unauthorized access and other malicious attacks.
2. Reliability: This refers to the set of attributes that bear on the capability of software to maintain its level of performance under stated conditions for a specified period of time. This characteristic is further divided into three sub-categories:
    - Maturity: The degree to which the software is free from defects and errors.
    - Fault tolerance: The ability of the software to continue operating in the presence of faults or errors.
    - Recoverability: The ability of the software to recover from failures or errors without causing damage to the system or data.
3. Usability: This refers to the set of attributes that bear on the effort required for use, and on the individual assessment of such use, by a stated or implied set of users. This characteristic is further divided into three sub-categories:
    - Learnability: The degree to which the software is easy to learn and understand.
    - Operability: The degree to which the software is easy to operate and control.
    - Attractiveness: The degree to which the software is visually appealing and satisfying to use.
4. Efficiency: This refers to the set of attributes that bear on the relationship between the level of performance of the software and the number of resources used under stated conditions. This characteristic is further divided into two sub-categories:
    - Time behavior: The degree to which the software completes tasks within a specified time frame.
    - Resource utilization: The degree to which the software uses resources efficiently, such as memory and processing power.
5. Maintainability: This refers to the set of attributes that bear on the effort required to make modifications to the software after delivery. This characteristic is further divided into three sub-categories:
    - Analyzability: The degree to which the software can be analyzed to identify problems and make changes.
    - Changeability: The degree to which the software can be modified without introducing errors.
    - Stability: The degree to which the software remains stable and reliable after modifications are made.
6. Portability: This refers to the set of attributes that bear on the ability of the software to be transferred from one environment to another. This characteristic is further divided into three sub-categories:
    - Adaptability: The degree to which the software can be adapted to work in different environments.
    - Installability: The degree to which the software can be installed and run on different platforms.
    - Co-existence: The ability of the software to coexist with other software in a given environment.

In summary, the ISO-9126 standard provides a hierarchical framework for assessing the quality of software, with six top-level quality characteristics, each of which is associated with several sub-categories. By assessing software quality based on this framework, software developers and testers can identify and address quality issues to improve the overall quality of their software product.

Slide 09

An error is a human action or decision that results in an incorrect or unintended output or outcome. A fault is an incorrect step, process, or data definition in a computer program that represents an underlying condition causing a certain failure. A defect refers to a problem with the external or internal behavior of software. A failure is the inability of a system or component of a system to perform the required functions within specified performance requirements, which is a behavioral deviation from user requirements or system specifications.

To ensure software quality, it is necessary to deal with defects, which can be prevented, detected, removed, or contained. Defect prevention involves designing and implementing processes, standards, and techniques to eliminate or reduce the occurrence of defects. Defect detection and removal involve testing, debugging, and verification activities to identify and correct defects. Defect containment involves managing and mitigating defects that cannot be prevented or removed, such as through error handling, fault tolerance, or recovery mechanisms.

- Defect prevention involves designing and implementing processes, standards, and techniques to eliminate or reduce the occurrence of defects. This can include techniques such as code reviews, static analysis, and continuous integration, as well as following industry best practices and standards.
- Defect detection and removal involve testing, debugging, and verification activities to identify and correct defects. This can include different types of testing such as functional, non-functional, regression, and performance testing, as well as debugging techniques and quality control processes.
- Defect containment involves managing and mitigating defects that cannot be prevented or removed, such as through error handling, fault tolerance, or recovery mechanisms. This can include developing contingency plans, implementing backup systems, and ensuring that the system can recover from errors or failures in a timely and efficient manner.

By implementing these three strategies, defect prevention, detection and removal, and containment, software quality can be ensured, and the software can meet the requirements of the users, customers, and stakeholders.

Slide 10

This slide discusses the possibility of variations in users' experiences with software failures, as well as the possibility of a software package that has successfully served an organization for a long time suddenly becoming bugged.

- The answer to the first question is yes, it is possible for variations in users' experiences with failures to appear with the same software packages. This is due to the fact that software can be complex, and users

may interact with it in different ways or in different environments, which can lead to different experiences with failures.

- The answer to the second question is also yes, it is possible for a software package that has successfully served an organization for a long time to suddenly become bugged. This is due to the characteristics of software, including its complexity, the potential for errors by programmers during development, and the possibility of changes to the software environment.

Software failures are typically due to software errors made by programmers. These errors can be syntactical, meaning that they violate the syntax or grammar of the programming language, or logical, meaning that they result in incorrect program behavior. However, not all errors become faults. A fault is an incorrect step, process, or data definition in a computer program that represents an underlying condition causing a certain failure. And not all faults become failures. A failure is the inability of a system or component of a system to perform the required functions within specified performance requirements.

Therefore, it is important to have quality assurance processes in place throughout the software development lifecycle to minimize the occurrence of errors, faults, and failures. This includes techniques such as testing, code reviews, and quality control processes to ensure that the software meets the requirements of the users, customers, and stakeholders.

Slide 11

This slide discusses the various causes of software errors that lead to poor software quality. The causes of software errors can be attributed to any individual involved in the software development process, including analysts, programmers, testers, documentation experts, managers, and clients.

These causes can be identified and categorized based on the development phase in which they occurred. The following are the common causes of software errors:

- Faulty definition of requirements: Errors in requirements can lead to a software system that doesn't meet the user's needs.
- Client-developer communication failure: Misunderstandings between clients and developers about the project requirements and specifications can lead to errors.
- Deliberate deviations from software requirements: When programmers deliberately deviate from software requirements or best practices, it can lead to errors.
- Logical design errors: Errors in the logical design of the software system can result in a system that does not meet the requirements.
- Coding errors: Errors that occur while writing the code can cause the software system to malfunction.
- Non-compliance with coding and docs instructions: Non-compliance with coding and documentation instructions can lead to software errors.
- Shortcomings of testing process: A testing process that doesn't cover all possible scenarios can lead to errors in the software system.
- Procedure error: Errors that occur while executing the software process.
- Docs error: Errors that occur while documenting the software system can lead to confusion and errors in the software system.

By identifying and addressing these causes during the development process, software errors can be prevented, detected, or contained, leading to improved software quality.

Slide 12

This slide discusses the history of quality assurance and how it has evolved over time, specifically in relation to software engineering. Traditionally, quality assurance was associated with physical objects and systems such as cars, radios, and televisions. In this setting, quality control was primarily concerned with ensuring that the products conformed to their specifications, with reducing variance being a key goal.

However, with the rise of the services industry, businesses had to adjust to shifting customer needs and expectations. Customer loyalty became more important than simply adhering to prescribed standards or specifications. The software industry incorporated both the conformance to specifications and the service view of quality.

High quality software is now defined as having three key characteristics: conformance, adaptability, and innovation. Conformance means that the software conforms to its specifications, while adaptability means that the software can be easily adapted to changing customer needs. Innovation means that the software is continually improving and incorporating new ideas and technologies.

Slide 13

The slide highlights the importance of quality in software engineering and how it is an integral aspect of the success or failure of a software product. The four major stages of software engineering are described as follows:

1. Functional Stage: This stage focuses on providing useful functions that replace manual handling. At this stage, the software should be able to perform the functions as specified and meet the user's requirements.
2. Schedule Stage: At this stage, the focus is on introducing important features in a timely and orderly manner to satisfy the urgent needs of the user. The software should be able to meet the deadlines and be delivered on time.
3. Cost Stage: The focus is on reducing the cost of the software to stay competitive in the market. The software should be able to provide value for money to the user.
4. Reliability Stage: The focus is on managing users' quality expectations. The software should be able to perform reliably and consistently over time, without any failures or defects. This stage is important as it determines the overall quality of the software and its ability to meet the user's expectations.

Therefore, the software engineering process should focus on achieving quality at every stage to ensure the success of the software product.

Lecture 02

Slide 02

This slide refers to the concept of Software Quality Assurance (SQA). SQA refers to the planned and systematic actions that are necessary to provide adequate confidence that software conforms to established technical requirements and standards.

The objectives of SQA include:

- Assuring an acceptable level of confidence that software will conform to functional requirements: This means ensuring that the software meets the specified functional requirements, such as providing the desired functionality, usability, reliability, and performance.
- Assuring an acceptable level of confidence that software will conform to managerial scheduling and budgeting requirements: This means ensuring that the software development process meets the required timelines, budget constraints, and other managerial requirements.
- Initiating and managing activities for improvement and greater efficiency of software development and SQA activities: This means continuously improving the software development process and SQA activities by identifying and addressing process improvement opportunities, adopting best practices, and promoting efficiency and effectiveness.

By achieving these objectives, SQA helps to ensure that the software is of high quality, meets user needs, and is delivered on time and within budget.

Slide 03

Defect handling is an important aspect of software quality assurance. The purpose of defect handling is to ensure that the system is delivered to customers or released in the market without any defects or with minimal defects that cause minimal damage or disruption. There are three stages of defect handling: defect prevention, defect detection, and defect correction.

Defect prevention is the first stage of defect handling. The objective of this stage is to prevent defects from being injected into the software in the first place. This is achieved through various quality assurance (QA) activities that prevent certain types of faults from being injected into the software. Defect prevention can be done in two ways:

1. Eliminating certain error sources: This involves eliminating the root causes of errors such as ambiguities or correcting human misconception that leads to errors.
2. Fault prevention or blocking: This stage breaks the relationship between error sources and faults through using certain tools and technologies such as formal methods, automated testing, code inspections, and reviews.

By implementing these defect prevention techniques, software developers can reduce the likelihood of defects in the software, resulting in a more reliable and high-quality product.

Slide 04

Defect reduction is the second stage of defect handling, where the goal is to identify and remove defects from the system. The techniques used for defect reduction include inspection, testing, static analysis, and dynamic analysis. These techniques help in detecting faults that were not prevented during the defect prevention stage.

Defect containment is the final stage of defect handling, which aims to prevent failures and contain the effects of failures that cannot be prevented. The goal is to minimize the damage caused by failures and keep them localized, so they don't propagate and cause more significant problems. There are two types of defect containment techniques:

1. Fault tolerance techniques: These techniques are designed to break the relationship between faults and failures, so that local faults do not result in global failures. Fault tolerance techniques include redundancy, error-correcting codes, and checkpointing.
2. Failure avoidance techniques: These techniques are designed to avoid catastrophic consequences such as death, injury, property or environmental damage. Examples of failure avoidance techniques include fail-safe design, backup systems, and emergency shutdown procedures.

Slide 05

This slide discusses the two phases of dealing with defects in software development: pre-release and post-release defect handling. The goal of pre-release defect handling is to prevent and remove defects during the software development process to ensure that the software meets the quality standards before it is released to the market. The two techniques used in pre-release defect handling are defect prevention and defect reduction.

However, sometimes defects may still remain in the software after release, these defects are called “Dormant” defects. The post-release defect handling phase deals with the defects that are detected after the software product has been released. Fixing defects in post-release is costlier than pre-release, so it is important to minimize them as much as possible.

Beta testing is a type of post-release defect handling technique in which the software is released to a selected group of users to gather feedback and identify any issues. Defect containment is also an important post-release defect handling technique that aims to minimize the impact of dormant defects during operational use after the software has been released to the market.

Slide 07

This slide explains that defect prevention is an important aspect of software quality assurance and that it is based on identifying and addressing the sources of errors that can occur during software development. The sources of errors can be human actions, imprecise designs and implementations, non-conformance to processes and standards, or other factors.

The slide suggests that if human misconceptions are the cause of errors, then education and training can be used to prevent them. If imprecise designs and implementations deviating from product specifications are the root causes, then formal methods can be used. If non-conformance to selected processes and standards is the source of error, then process conformance and standard enforcement can be used. Finally, if certain tools and technologies can reduce fault injection, then they should be adopted to prevent defects.

The goal of defect prevention is to eliminate or minimize the occurrence of defects during software development, which can ultimately save time, resources, and costs associated with detecting and fixing defects later in the development process or after the software has been released.

Slide 08

This slide highlights the importance of education and training in preventing defects in software. It suggests that many sources of errors in software development are due to human actions and can be addressed through education and training. By improving the way people work, it is possible to prevent certain types of faults from being injected into software.

The slide lists several areas where education and training should be focused. These include:

1. Product and Domain specific Knowledge: Developers should have a deep understanding of the product they are developing and the domain it operates in. This can help prevent errors caused by misunderstanding requirements or the intended use of the software.
2. Software development Knowledge and Expertise: Developers should be knowledgeable about software development best practices and have expertise in the specific programming languages and tools used in the development process.
3. Knowledge about Development Methodology, Technology and Tools: Developers should have a good understanding of the development methodology being used, as well as the technology and tools used in the development process.
4. Development Process Knowledge: Developers should be knowledgeable about the entire software development process, including requirements gathering, design, development, testing, and deployment.

By providing education and training in these areas, organizations can help their developers to work more effectively and prevent many types of errors from occurring.

Slide 09

Defect Prevention Formal Method is a technique used in software engineering to eliminate certain sources of errors and to verify the absence of related faults. It involves two main steps: formal specification and formal verification. Formal specification refers to producing a clear and unambiguous set of product specifications, which can help eliminate errors caused by imprecise designs or implementation deviations from product specifications. Formal verification, on the other hand, involves checking the conformance of software design/code against these formal specifications, which can help ensure that the software is free of faults.

Other defect prevention techniques include using tools and technologies that result from "Fat to Lean" software development practices, which aim to reduce the amount of unnecessary code and features in software, making it more efficient and less prone to errors. Properly defined and managed processes, such as those outlined by industry standards like ISO and CMMI, can also help reduce the likelihood of errors. Finally, some tools and technologies can help reduce the chances of fault injections by providing automated checks and validations throughout the software development process.