SENG438

Software Testing, Reliability, and Quality

objectives

- Understanding fundamental of software testing reliability, and quality.
  - Understand software testing & reliability engineering process
  - Understand software design with quality mind set
  - Have an overview of the careers outlooks of software testing and quality assessment.
  - Gain experience working in a software testing team

- Designing test cases and improving test suites
  - Understand test planning process, test selection and prioritization strategies
  - Assess test adequacy using test coverage metrics
- Assessing reliability and quality of software.
  - Use test tools and frameworks and track quality during development and post-release

Java is used → develop test cases in Java

Terminology

Time:

- Meeting the project deadline.
- Reaching the market at the right time.

Cost :

- Meeting the anticipated project costs.

Quality:

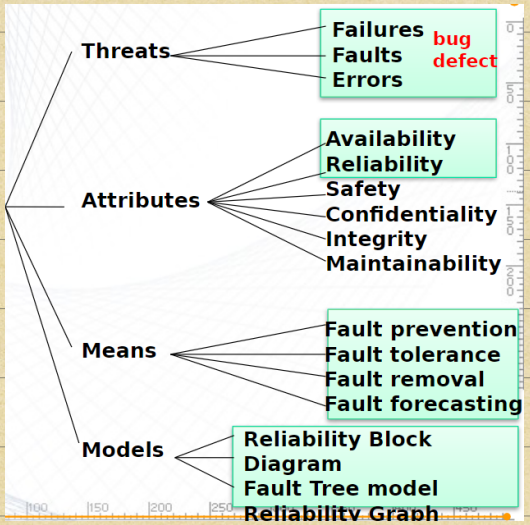
- Working fine for the designated period on the designated system.

Dependability

The ability of a system to avoid failures that are more frequent or more severe, and outage durations that are longer, than is acceptable to the users.

The ability of a system to deliver service that can justifiably be trusted.

Dependability



Software Testing

- Concept: Exploratory, Black box, White box, Regression, Unit testing, Integration, etc
- Model: Coverage Model, Adequacy Model or Statistical Model
- Process: Test Process

Software Reliability

- Concept: Reliability, Availability, Failure Rate, MTTF, Failure Density
- Model: Single Failure Model, Multiple failure Model → Reliability Growth Model
- Process: SRE Process

Software Quality

- Size & Effort: Predict size, Predict effort, Predict time
- Standards & Models: ISO 9126, ISO 25K
- Assessment techniques: Pre-release, Post-release

Service:

- The service delivered by a system is its behaviour as it is perceived by its users; a user is another system (physical, human) that interacts with the former at the service interface.
- The function of a system is what the system is intended to do, and is described by the functional specification
- Correct service is delivered when the service implements the system function.
- The delivery of incorrect service is a system outage.
- A transition from incorrect service to correct service is service restoration.

Dependability: Threats

- Error: A human action that results in software containing a fault.
- Fault: A cause for either a failure of the program or an internal error (incorrect timing/state). Must be detected and removed
- Failure: Usually what can be observed among the 3 threats
- Failure: A system failure is an event that occurs when the delivered service deviates from correct service to incorrect service, i.e., to not implementing the system function. Any departure of

system behavior in execution from user needs. A failure is caused by a fault and the cause of a fault is usually a human error.

- Failure Mode: The manner in which a fault occurs, i.e., the way in which the element faults.
- Failure Effect: The consequences of a failure mode on an operation, function, status of a system/process/activity/environment. The undesirable outcome of a fault of a system element in a particular mode.

		Inspection Effort	
		Higher	Lower
Failure Density	Higher	Good/ Not bad	Worst Case
	Lower	Best Case	Unsure

Failure Intensity & Density

- Failure Intensity (failure rate): The rate failurer are happening, i.e., number of failures per natural or time unit. Failure intensity is a way of expressing system reliability. e.g. 5 failures per hour, 2 failures / 1000 transactions. For system end users.
- Failure Density: failure per KLOC (or per FP) of developed code, e.g., 1 failure per KLOC, 0.2 failure per FP, etc. (system developers)

Failure Density vs Inspection Effort

- Is failure density a good metric for software quality?
- The more bugs found & fixed doesn't necessarily imply better software quality because the fault injection rate and the effort to fix them may be different.

Testing vs. Inspection

Inspections are strict and close examinations conducted on specifications, design code, test and other artifacts.

1) Inspections allow for defect detection, prevention and isolation.
2) Testing allows for defect detention.
1) Start early in life cycle.
2) Start later

Cool stats:

- Inspections are up to 20 times more efficient than testing.
- Code reading detects twice as many defects/hour as testing.
- 80% of development errors are usually found by inspections.
- Inspections resulted in a 10x reduction in cost of finding errors.

Inspections or Testing

- Can inspection replace testing?
- Basically no. Inspections could replace testing if and only if all information gleaned through testing could be obtained through inspection.
  - Complex interactions in large systems.
  - Software reliability indicator
  - Nonfunctional requirements: performance, usability, etc.

- **Fault**: A cause for either failure of the program or an internal error.
  - A fault must be detected and removed.
  - Fault can be removed without execution (code inspection, design review)
  - Fault removal due to execution depends on the occurence of associated "failure".
  - Occurrence depends on length of execution time and operational profile.
- **Defect**: Refers to either fault (cause) or failure (effect)

Error

- Error has two meanings:
  - A discrepancy between a computed, observed or measured value or condition and the true, specified or theoretically correct value or condition.
  - A human action that results in software containing a fault.
- Human errors are the hardest to detect.

Dependability Attributes

- **Availability**: readiness for correct service
- **Reliability**: continuity of correct service
- **Safety**: absence of catastrophic consequences on the users and the environment.
- **Confidentiality**: Absence of unauthorized disclosure of information.

- **Integrity**: absence of improper system state
- **Maintainability**: ability to undergo repairs and modifications

- Dependability attributes may be emphasized to a greater or lesser extent depending on the application: availability is always required, whereas reliability, confidentiality, safety may or may not be required.
- Other dependability attributes can be defined as combinations or specializations of the six basic attributes.

- **Example**: Security is the concurrent existence of
  - Availability for authorized users only.
  - Confidentiality; and
  - Integrity with improper taken as meaning unauthorized.

- **Availability**: a measure of the delivery of correct service with respect to the alternation of correct and incorrect service.

$$\text{Availability} = \frac{\text{Uptime}}{\text{Uptime} + \text{Downtime}}$$

$$\text{Availability} = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}} = \frac{\text{MTTF}}{\text{MTBF}}$$

- Reliability: A measure of the continuous delivery of correct service.
- Reliability is the probability that a system or a capability of a system functions without failure for a specified time or number of natural units in a specified environment. Given that the system was functioning at the begining of the time period.
- Probability of failure-free operation for a specified time in a specified environment for a given purpose.
- A recent survey of software consumers revealed that reliability was the most important quality attribute of the app software.

Notes:

- Reliability depends on how the software is used. A model of usage is required.
- Reliability can be improved over time if certain bugs are fixed. (reliability growth). Therefore a trend model (aggregation or regression) is needed.
- Failures may happen at random time. Therefore a probabilistic model of failure is needed.

Safety:

- Safety: Absence of catastrophic consequences on the users and the environment.

- Safety is an extension of reliability: safety is reliability with respect to catastrophic failures.

- When the state of correct service and the states of incorrect service due to non-catastrophic failure grouped into a safe state (in the sense of being free from catastrophic damage, not from danger), safety is a measure of continuous safeness, or equivalently, of the time to catastrophic failure.

- Confidentiality: Absence of unauthorized disclosure of information.
- Integrity: Absence of improper system state alterations.
- Maintainability: Ability to undergo repairs and modifications. It is a measure of the time to service restoration since the last failure occurence, or equivalently, measure of the continuous delivery of incorrect service.

Means:

- Fault prevention: How to prevent the occurrence or introduction of faults.
- Fault tolerance: How to deliver correct service in the presence of faults.
- Fault removal: How to reduce the number or severity of faults.

- Fault forecasting: How to estimate the present number, the future incidence, and the likely consequences of faults.

Fault Prevention

- To avoid fault occurrences by construction.
- Fault prevention is attained by quality control techniques employed during the design and manufacturing of software.
- Fault prevention intends to prevent operational physical faults.
- Technique Examples:

Design review, modularization, consistency checking, structured prog etc

Fault Prevention Activities

- Requirement Review
- Design Review
- Clear Code
- Establishing standards (ISO 9000-3, etc)
- Using CASE tools with built-in check mechanisms.

Fault Tolerance

- A fault-tolerant computing system is capable of providing specified services in the presence of a bounded number of failures.

- Use of techniques to enable continued delivery of service during system operation.
- It is generally implemented by error detection and subsequent system recovery.
- Based on the principle of:
  - Act during operation
  - Defined during specification and design.

Process

1) Detection: Identify faults and their causes (errors)
2) Assessment: Assess the extent to which the system has been damaged or corrupted.
3) Recovery: Remain operational or regain operational status.
4) Fault treatment and continued service: Locate and repair the fault to prevent another occurrence.

Fault Removal

- Fault removal is performed both during the development phase, and during the operational life of a system.
- Fault removal during the development phase of a system life-cycle consists of three steps:
  verification → diagnosis → correction
- Verification: The process of checking whether the system adheres to given properties, called verification conditions.

If it does not, the other two steps follow:
Diagnosing the faults that prevented the verification conditions from being fullfilled, and then performing the necessary corrections.

- After correction, the verification process should be repeated in order to check that fault removal had no undesired consequences; the verification performed at this stage is usually called non-regression verification.
- Checking the specification is usually referred to as validation.
- Uncovering specification faults can happen at any stage of the development, either during the specification phase itself, or during subsequent phases when evidence is found that the system will not implement its function, or that the implementation cannot be achieved in a cost effective way.