Software Engineering

Dr. Mritunjay Shall Peelam
Assistant Professor-SG, UPES

Course Introduction and Objectives

Course Code	Course name	L	T	P	C
CSEG2064	Software Engineering	3	0	0	3
Total Units to be Covered: 5		Total Contact Hours: 45			
Prerequisite(s):		Syllabus version: 1.0			

Course Objectives

1. To explore software development methodologies (waterfall, agile, DevOps) and their integration of testing, quality assurance, reliability, and risk management.
2. To comprehend software requirements engineering and develop skills in creating well-structured Software Requirements Specifications (SRS).
3. To acquire understanding of planning a software project, its cost estimation models and to understand the software quality models.

Course Introduction and Objectives

Course Outcomes

CO 1. Understand the fundamental concepts and importance of Software Engineering in modern software development.

CO 2. Learn various software development methodologies, including Agile, Waterfall, and iterative approaches.

CO 3. Explore software design principles and architectural patterns for creating robust and maintainable software systems.

CO 4. Apply project management principles to effectively plan, monitor, and control software projects.

Course Introduction and Objectives

Syllabus

Unit I: Introduction to Software Engineering **7 Lecture Hours**

Definition of Software Engineering, S/W characteristics, applications, Software development life cycle ; Life Cycle Models – Waterfall (classical and iterative), Spiral, Prototyping & RAD Models, Software processes, Process Models – overview Agile Model and Various Agile methodologies - Scrum, XP, Lean, and Kanban. Scope of each model and their comparison in real-world case studies.

Course Introduction and Objectives

Unit II: Requirements Modelling and Design **9 Lecture Hours**

System and software requirements; Requirements Engineering-Crucial steps; types of requirements, Functional and non-functional requirements; Domain requirements; User requirements; Elicitation and analysis of requirements; Requirements documentation – Nature of Software, Software requirements specification, Use case diagrams with guidelines, DFD, SRS Structure, SRS Case study, Design concepts and principles - Abstraction - Refinement - Modularity Cohesion coupling, Architectural design, Detailed Design Transaction Transformation, Refactoring of designs, Object-oriented Design User-Interface Design.

Course Introduction and Objectives

Unit III: Software Reliability **9 Lecture Hours**

Introduction to Software Reliability; Hardware reliability vs. Software reliability; Reliability metrics; Failure and Faults – Prevention, Removal, Tolerance, Forecast; Dependability Concept – Failure Behavior, Characteristics, Maintenance Policy; Reliability and Availability Modeling; Reliability Evaluation Testing methods, Limits, Starvation, Coverage, Filtering; Microscopic Model of Software Risk; Classes of software reliability Models; Statistical reliability models; Reliability growth models; Defining and interpreting reliability metrics; Fault Detection and Prevention; Techniques for detecting and mitigating software faults; Static analysis tools and techniques; Dynamic analysis methods; Software Fault Tolerance; Software Maintenance and Reliability; Reliability Assessment and Evaluation; Methods for assessing and quantifying software reliability; Case Studies and Real-world Applications.

Course Introduction and Objectives

Unit IV: Software Testing, metrics and Quality Assurance **10 Lecture Hours**

Testing types and techniques such as black box, white box, and gray box testing, functional and structural testing; Test-driven development, code coverage, and quality metrics; Testing process, design of Test cases, testing techniques - boundary value analysis - equivalence class testing - decision table testing, cause-effect graphing, path testing, data flow testing, and mutation testing. Unit, integration, system, alpha, and beta testing, debugging techniques; verification and validation techniques, levels of testing, regression testing, quality management activities, product and process quality standards (ISO9000, CMM), metrics understanding (process, product, project metrics), size metrics (LOC, Function Count, Albrecht FPA), product metrics, metrics for software maintenance, cost estimation techniques (static, single variable, multivariable models), cost-benefit evaluation techniques, Testing tools and standards such as Jira and Selenium, test automation frameworks and tools (Selenium, Appium, JUnit), performance testing and load testing, and defect management and root cause analysis.

Course Introduction and Objectives

Unit V: Software Quality and Risk Management **10 Lecture Hours**

McCall quality factors, ISO and CMM Model, Tools and Techniques for Quality Control, Pareto Analysis, Statistical Sampling, Quality Control Charts and the seven Run Rule. Modern Quality Management, Risk Management – importance, types, process and phases, qualitative and quantitative risk analysis, Risk Analysis and Assessment, Risk Strategies, Risk Monitoring and Control, Risk Response and Evaluation. Software Reliability: Reliability Metrics, Reliability Growth Modeling. Use Case: Defect Tracking and Management. Test Automation Tools: Jira, Selenium:, Appium; JUnit.

Text Books and References

Textbooks	1. Roger S. Pressman, "Software Engineering: A practitioner's approach", 7th Edition, McGraw Hill, 2009. 2. Pankaj Jalote, "An integrated approach to Software Engineering", 3rd Edition, Springer/Narosa, 2005.
Reference books	1. James F. Peters, and Witold Pedrycz, "Software Engineering: an Engineering approach", John Wiley, 2007. 2. Waman S Jawadekar, "Software Engineering principles and practice", McGraw Hill, 2004.
Web Resources **Journals**	
MOOCs, online courses	

Modes of Evaluation

Modes of Evaluation: Quiz/Assignment/ presentation/ extempore/ Written Examination etc.

Examination Scheme

Components	IA	MID SEM	End Sem	Total
Weightage (%)	50	20	30	100

Unit-3

Unit III: Software Reliability **9 Lecture Hours**

Introduction to Software Reliability; Hardware reliability vs. Software reliability; Reliability metrics; Failure and Faults – Prevention, Removal, Tolerance, Forecast; Dependability Concept – Failure Behavior, Characteristics, Maintenance Policy; Reliability and Availability Modeling; Reliability Evaluation Testing methods, Limits, Starvation, Coverage, Filtering; Microscopic Model of Software Risk; Classes of software reliability Models; Statistical reliability models; Reliability growth models; Defining and interpreting reliability metrics; Fault Detection and Prevention; Techniques for detecting and mitigating software faults; Static analysis tools and techniques; Dynamic analysis methods; Software Fault Tolerance; Software Maintenance and Reliability; Reliability Assessment and Evaluation; Methods for assessing and quantifying software reliability; Case Studies and Real-world Applications.

Software Reliability

Basic Concepts

There are three phases in the life of any hardware component i.e., burn-in, useful life & wear-out.

In **burn-in phase,** failure rate is quite high initially, and it starts decreasing gradually as the time progresses.

During **useful life period**, failure rate is approximately constant.

Failure rate increase in **wear-out phase** due to wearing out/aging of components. The best period is useful life period. The shape of this curve is like a “bath tub” and that is why it is known as bath tub curve. The “bath tub curve” is given in Fig.7.1.

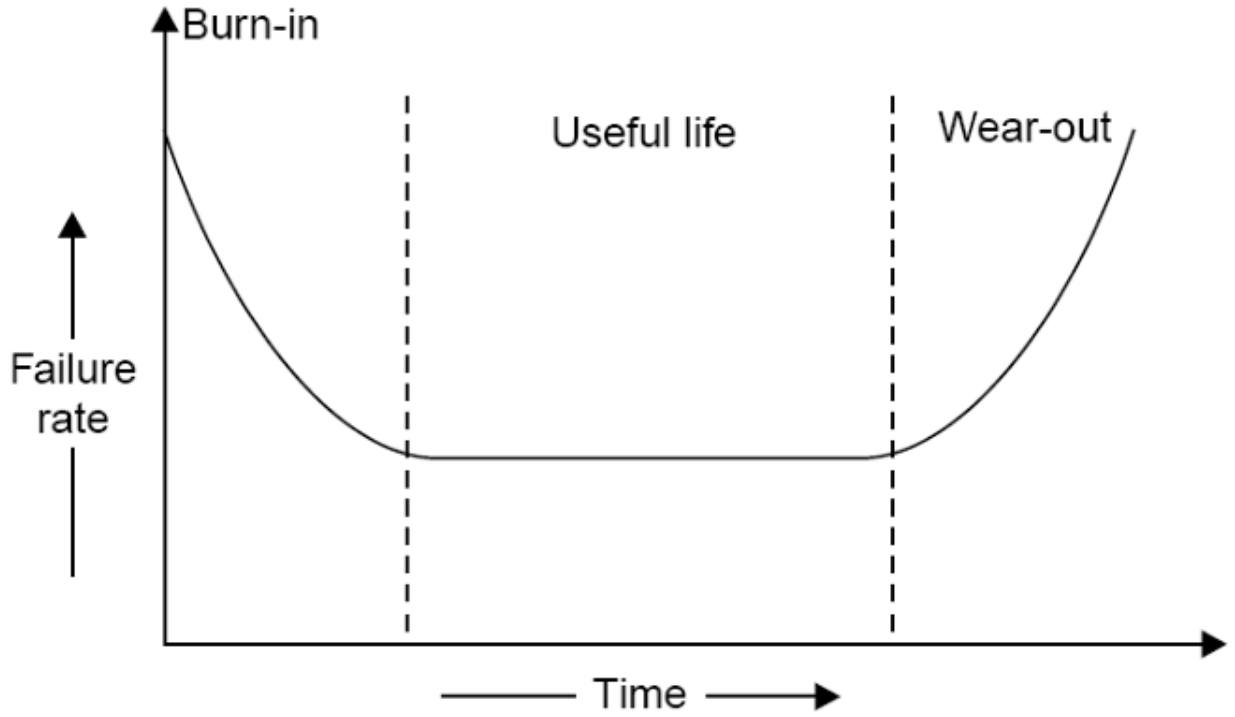


Fig. 7.1: Bath tub curve of hardware reliability.

We do not have wear out phase in software. The expected curve for software is given in fig. 7.2.

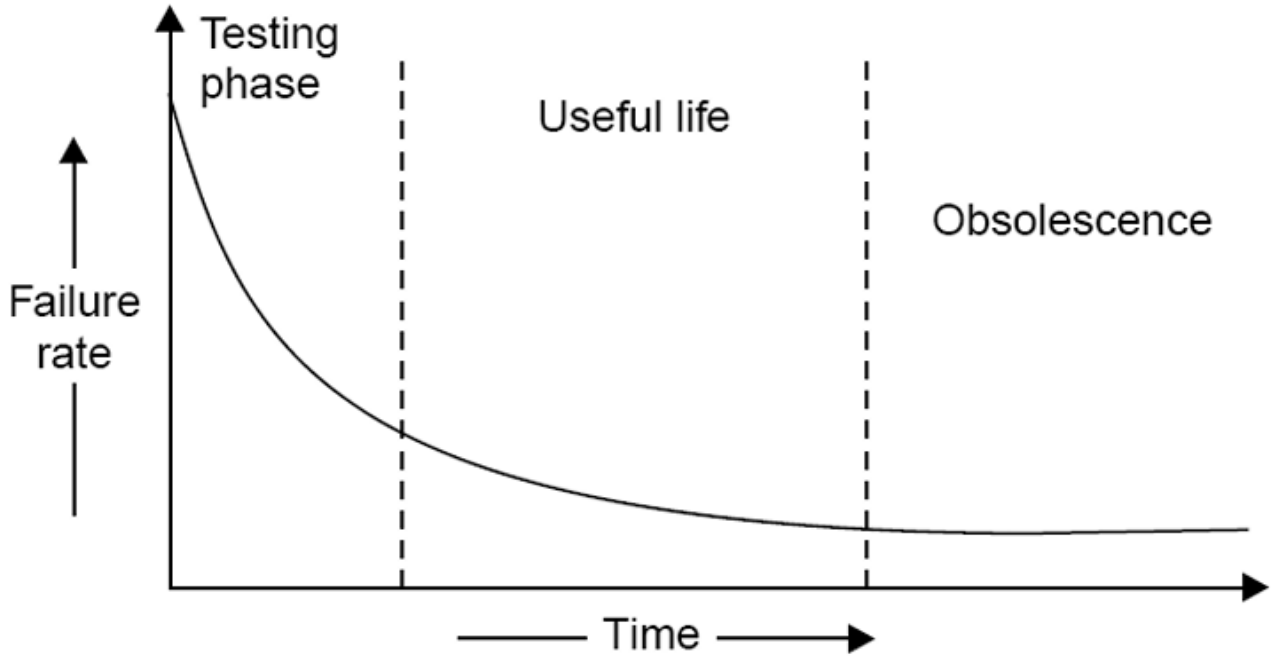


Fig. 7.2: Software reliability curve (failure rate versus time)

Software may be retired only if it becomes obsolete. Some of contributing factors are given below:

- ✓ change in environment
- ✓ change in infrastructure/technology
- ✓ major change in requirements
- ✓ increase in complexity
- ✓ extremely difficult to maintain
- ✓ deterioration in structure of the code
- ✓ slow execution speed
- ✓ poor graphical user interfaces

What is Software Reliability?

"Software reliability means operational reliability. Who cares how many bugs are in the program?

As per IEEE standard: "Software reliability is defined as the ability of a system or component to perform its required functions under stated conditions for a specified period of time".

Software reliability is also defined as the probability that a software system fulfills its assigned task in a given environment for a predefined number of input cases, assuming that the hardware and the inputs are free of error.

"It is the probability of a failure free operation of a program for a specified time in a specified environment".

- Failures and Faults

A fault is the defect in the program that, when executed under particular conditions, causes a failure.

The execution time for a program is the time that is actually spent by a processor in executing the instructions of that program. The second kind of time is calendar time. It is the familiar time that we normally experience.

There are four general ways of characterising failure occurrences in time:

1. time of failure,
2. time interval between failures,
3. cumulative failure experienced up to a given time,
4. failures experienced in a time interval.

- Failure
  - A failure is said to occur if the **observable** outcome of a **program execution** is different from the expected outcome.
- Fault
  - The adjudged cause of failure is called a fault.
  - Example: A failure may be cause by a defective block of code.
- Time
  - Time is a key concept in the formulation of reliability. If the time gap between two successive failures is short, we say that the system is less reliable.
  - Two forms of time are considered.
    - Execution time (τ)
    - Calendar time (t)

Failure Number	Failure Time (sec)	Failure interval (sec)
1	8	8
2	18	10
3	25	7
4	36	11
5	45	9
6	57	12
7	71	14
8	86	15
9	104	18
10	124	20
11	143	19
12	169	26
13	197	28
14	222	25
15	250	28

Table 7.1: Time based failure specification

1st failure occurred at 8 seconds
2nd failure at 18 seconds
3rd failure at 25 seconds
and so on…

Failure Interval=Current Failure Time–Previous Failure Time

Time (sec)	Cumulative Failures	Failure in interval (30 sec)
30	3	3
60	6	3
90	8	2
120	9	1
150	11	2
180	12	1
210	13	1
240	14	1

Table 7.2: Failure based failure specification

At 30 sec → 3 total failures
At 60 sec → 6 total failures
At 90 sec → 8 total failures

What This Table Shows

Early intervals have more failures (3, 3, 2).
Later intervals show fewer failures (1, 1, 1).This indicates failure rate is decreasing.
Hence, system reliability is improving over time.

Difference from Previous Table

Previous table (Time-based) → Records exact time of each failure.

This table (Failure-based / Interval-based) → Groups failures into fixed time intervals.

Failures in interval=Current cumulative−Previous cumulative

Value of random variable (failures in time period)	Probability	
	Elapsed time t_A = 1 hr	Elapsed time t_B = 5 hr
0	0.10	0.01
1	0.18	0.02
2	0.22	0.03
3	0.16	0.04
4	0.11	0.05
5	0.08	0.07
6	0.05	0.09
7	0.04	0.12
8	0.03	0.16
9	0.02	0.13

Table 7.3: Probability distribution at times t_A and t_B

At 1 Hour (t_a)

- Higher probabilities for **small number of failures (0–3)**.
- Example:
  - P(0 failures) = 0.10
  - P(2 failures) = 0.22 (highest)
- Indicates system is relatively reliable in short duration.

At 5 Hours (t_β)

- Higher probabilities shift toward **larger number of failures (6–9)**.
- Example:
  - P(8 failures) = 0.16
  - P(9 failures) = 0.13
- Indicates failures accumulate over longer time.

As time increases, probability mass shifts toward higher failure counts.
This shows: Failure occurrence increases with time.
Reliability decreases as operating time increases.

Value of random variable (failures in time period)	Probability	
	Elapsed time t_A = 1 hr	Elapsed time t_B = 5 hr
10	0.01	0.10
11	0	0.07
12	0	0.05
13	0	0.03
14	0	0.02
15	0	0.01
Mean failures	3.04	7.77

Table 7.3: Probability distribution at times t_A and t_B

As time increases → expected failures increase.
Probability shifts toward higher failure counts.

Reliability decreases with longer operating time.

System follows a probabilistic failure model (Poisson-type behavior).

This table represents the **probability distribution of number of failures** occurring within a given time period at two different elapsed times:

- $t_A = 1$hour
- $t_B = 5$hours

A random process whose probability distribution varies with time to time is called non-homogeneous. Most failure processes during test fit this situation. Fig. 7.3 illustrates the mean value and the related failure intensity functions at time t_A and t_B. Note that the mean failures experienced increases from 3.04 to 7.77 between these two points, while the failure intensity decreases.

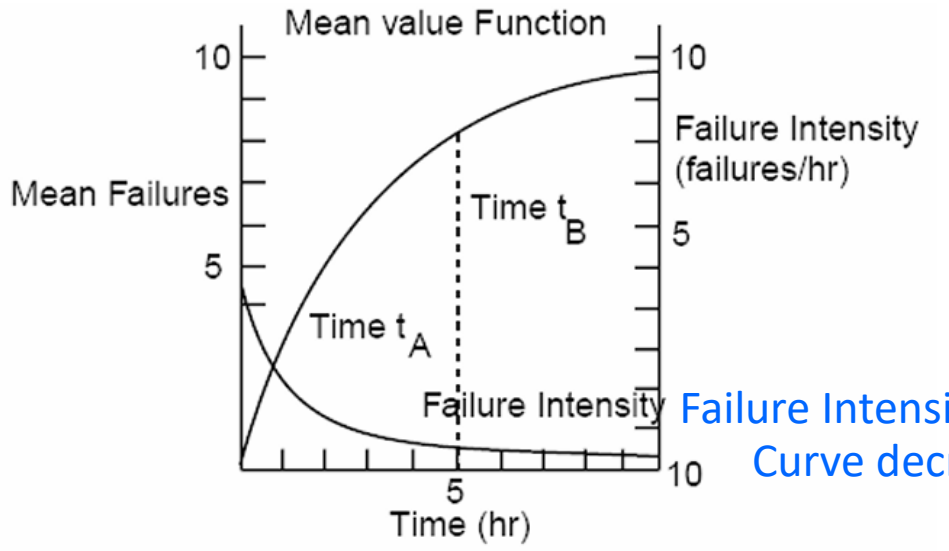
Failure behavior is affected by two principal factors:

- ✓ the number of faults in the software being executed.
- ✓ the execution environment or the operational profile of execution.

Summary Table

Type of Data	What It Represents	Nature of Information	Purpose
Time-Based Failure Data	Exact failure occurrence times	Continuous event data	Used for reliability growth modeling
Interval-Based Failure Data	Failures grouped every 30 sec	Aggregated data	Used to study failure rate trend
Probability Distribution	Probability of k failures at given time	Statistical/Probabilistic data	Used for reliability prediction

Mean Value Function – *m(t)* (Upper Curve)



Failure Intensity – *λ(t)* (Lower Curve)
Curve decreases over time.

Fig. 7.3: Mean Value & failure intensity functions.

- **It shows the expected cumulative number of failures detected up to time *t*.**
- **Initially, many faults are found quickly.**
- **Later, fewer faults remain → curve flattens.**
- **Eventually it approaches a maximum (total inherent defects in system).**

- **This figure represents Software Reliability Growth Model (SRGM) behavior, typically based on an NHPP (Non-Homogeneous Poisson Process) model.**

- **It shows the rate at which failures are occurring at time *t*.**
- **Early testing → high failure rate.**
- **As bugs are fixed → failure rate reduces.**
- **Eventually → system becomes stable.**

Uses of Reliability Studies

There are at least four other ways in which software reliability measures can be of great value to the software engineer, manager or user.

1. you can use software reliability measures to evaluate software engineering technology quantitatively.

2. software reliability measures offer you the possibility of evaluating development status during the test phases of a project.

3. one can use software reliability measures to monitor the operational performance of software and to control new features added and design changes made to the software.

4. a quantitative understanding of software quality and the various factors influencing it and affected by it enriches into the software product and the software development process.

Software Reliability Attributes and Specification

- Reliability attributes are the measure of software reliability. There may be many reliability requirements for different software but all the software have some common reliability measures.
- Reliability attributes for different categories of software products may be different. Therefore, it is necessary to specify the that the level of reliability required for a software product in the software requirements specification document.
- There are many reliability attributes which can be used to express the reliability of a software product.

Software Reliability Attributes and Specification

- **Rate of occurrence of failure (ROCOF) :**

- ROCOF measures the frequency of occurrence of unexpected behavior of the software.
- It basically measures how many times the software product fails.
- ROCOF measure of a software product can be obtained by having the record of the behavior of a software product.
- ROCOF is basically the total number of failures occurring during the specified time interval.

$$\mathbf{ROCOF} = \frac{Total\ Number\ of\ Failures}{Total\ Time\ Interval}$$

Software Reliability Attributes and Specification

Question: a software system is observed for 200 hours, and during this time 10 failures occur.

Solution:

$$ROCOF = \frac{10}{200}$$

$$ROCOF = 0.05 \ failures\ per\ hour$$

Software Reliability Attributes and Specification

- **Mean Time Between Failures (MTBF):** Measurement of reliability testing is done in terms of mean time between failures (MTBF).

- **Mean Time To Failure (MTTF):** The time between two consecutive failures is called as mean time to failure (MTTF).

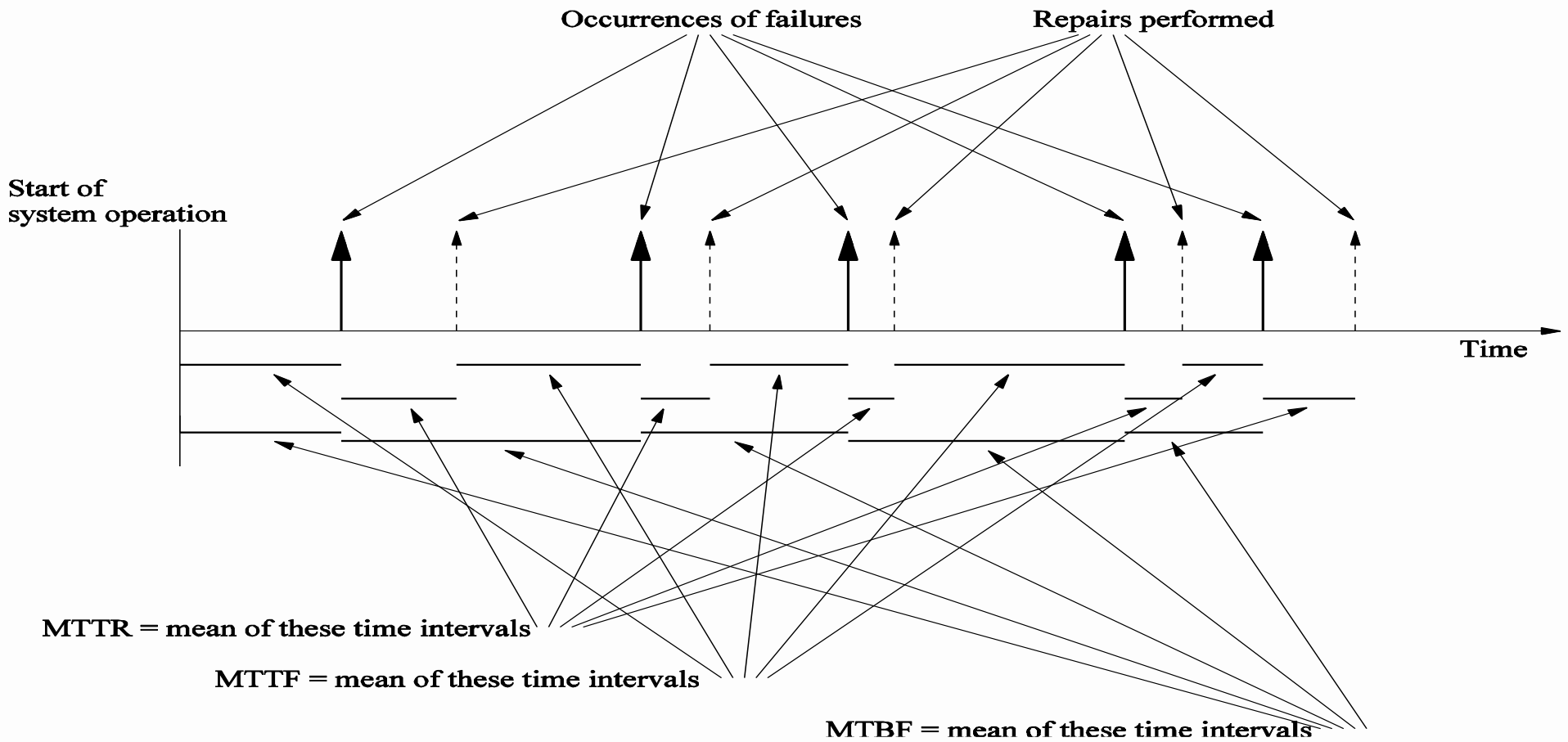
$$\text{MTTF} = \frac{\text{Total Uptime}}{\text{Number of Failures}}$$

- **Mean Time To Repair (MTTR):** The time taken to fix the failures is known as mean time to repair (MTTR).

$$\text{MTTR} = \frac{\text{Total Repair Time}}{\text{Number of Failures}}$$

- **MTBF = MTTF + MTTR** MTBF indicates that once a failure occurs, the next failure is expected after same time.

Software Reliability Attributes and Specification



What is Reliability?

- Two ways to measure reliability
  - Counting failures in periodic intervals
    - Observer the trend of **cumulative failure count - $\mu(\tau)$.**
  - Failure intensity
    - Observe the trend of **number of failures per unit time – $\lambda(\tau)$.**
- $\mu(\tau)$
  - This denotes the **total number of failures** observed until execution time **τ** from the beginning of system execution.
- $\lambda(\tau)$
  - This denotes the **number of failures observed per unit time** after **τ** time units of executing the system from the beginning. This is also called the failure intensity at time **τ.**
- Relationship between $\lambda(\tau)$ and $\mu(\tau)$
  - $\lambda(\tau) = d\mu(\tau)/d\tau$

Definitions of Software Reliability

- First definition
  - Software reliability is defined as the probability of failure-free operation of a software system for a specified time in a specified environment.
    - Key elements of the above definition
      - Probability of failure-free operation
      - Length of time of failure-free operation
      - A given execution environment
    - Example
      - The probability that a PC in a store is up and running for eight hours without crash is 0.99.
- Second definition
  - Failure intensity is a measure of the reliability of a software system operating in a given environment.
    - Example: An air traffic control system fails once in two years.
- Comparing the two
  - The first puts emphasis on MTTF, whereas the second on count.

Applications of Software Reliability

- Comparison of software engineering technologies
  - What is the cost of adopting a technology?
  - What is the return from the technology -- in terms of cost and quality?
- Measuring the progress of system testing
  - Key question: How of testing has been done?
  - The failure intensity measure tells us about the present quality of the system: high intensity means more tests are to be performed.
- Controlling the system in operation
  - The amount of change to a software for maintenance affects its reliability. Thus the amount of change to be effected in one go is determined by how much reliability we are ready to potentially lose.
- Better insight into software development processes
  - Quantification of quality gives us a better insight into the development processes.

Factors Influencing Software Reliability

- A user's perception of the reliability of a software depends upon two categories of information.
  - The number of faults present in the software.
  - The ways users operate the system.
    - This is known as the *operational profile*.
- The fault count in a system is influenced by the following.
  - Size and complexity of code
  - Characteristics of the development process used
  - Education, experience, and training of development personnel
  - Operational environment

Software Reliability Attributes and Specification

Q. A software system runs for 1000 hours and experiences 8 failures during this period. The total repair time recorded is 80 hours.

Calculate:
a) Mean Time To Failure (MTTF)
b) Mean Time To Repair (MTTR)
c) Mean Time Between Failures (MTBF)

Solution:

Software Reliability Attributes and Specification

Solution:

Calculate Total Uptime

$$\text{Total } \textit{Uptime} = \textit{Total Time} - \textit{Total Repair Time}$$

$$\text{Total } \textit{Uptime} = 1000 - 80 = 920 \textit{ hours}$$

$$\text{MTTF} = \frac{\text{Total Uptime}}{\text{Number of Failures}}$$

$$\text{MTTF} = \frac{920}{8} = 115 \text{ hours}$$

$$\text{MTTR} = \frac{\text{Total Repair Time}}{\text{Number of Failures}}$$

$$\text{MTTR} = \frac{80}{8} = 10 \text{ hours}$$

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

$$\text{MTBF} = 115 + 10 = 125 \text{ hours}$$

Software Reliability Attributes and Specification

- **Probability of Failure on Demand (POFOD):**
- **This attribute does not involve time measurements like the other attributes.**
- **POFOD measures the behavior of the system failing when a service request is made.**
- **For example, if POFOD measures 0.001 then it means that out of every 1000 one time the service requests would result in a failure.**

$$\mathbf{POFOD} = \frac{\textit{Number of Failures}}{\textit{Number of Service Requests (Demands)}}$$

Software Reliability Attributes and Specification

Question: An online payment system processes **5000 transactions**, and **10 failures** occur. Find the POFOD.

Solution:

$$POFOD = \frac{10}{5000}$$

$$POFOD = 0.002$$

Failure probability is 0.2% per request.

Software Reliability Attributes and Specification

- **Availability:** Availability of a system is a measure of possibility of availability of the system for use over a given period of time. It considers the number of failures occurring during a time interval. It also takes into measure the downtime of a system when a failure occurs. This attribute is important for systems which are supposed to be never down. This plays a vital role in telecommunication systems and operating systems. These systems need to run even during the repair time.

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

The availability of the system is 0.9524 or 95.24%, meaning the system is operational about 95% of the time.

- **Fault Tolerance:** The ability of a system to function correctly in the event of unexpected events or component breakdowns is known as fault tolerance. The ability of these systems to carefully recover from faults keeps just one point of breakdown from creating significant interruptions.

Software Reliability Attributes and Specification

- **Resilience:** The system's resilience is its capacity to adjust to shifting circumstances and carry on operating even in the face of unexpected challenges. These systems are resilient enough to withstand disturbances like hardware failures, network problems or spikes in traffic without significantly affecting performance.
- **Recoverability:** Recoverability refers to a system's ability to bounce back from a setback or unforeseen incident quickly and effectively. Recoverability of a system guarantees less downtime and loss of data, which improves overall dependability and user happiness.

Software Reliability Attributes and Specification

- **Redundancy:** Redundancy is the process of duplicating essential system parts or operations to guarantee uninterrupted use in the event of a failure. It increases the overall reliability of the system, helps distribute the load and avoids a single point of failure.
- **Scalability:** Scalability is the capacity of a system to add nodes or resources to accommodate growing workloads or user loads. The performance and dependability of these systems can grow along with the user base or workload, due to their capacity to adapt.
- **Consistency:** Maintaining consistency makes a system operate in a predictable and dependable manner, giving consumers a consistent experience. Errors and confusion can result from inconsistent behavior.

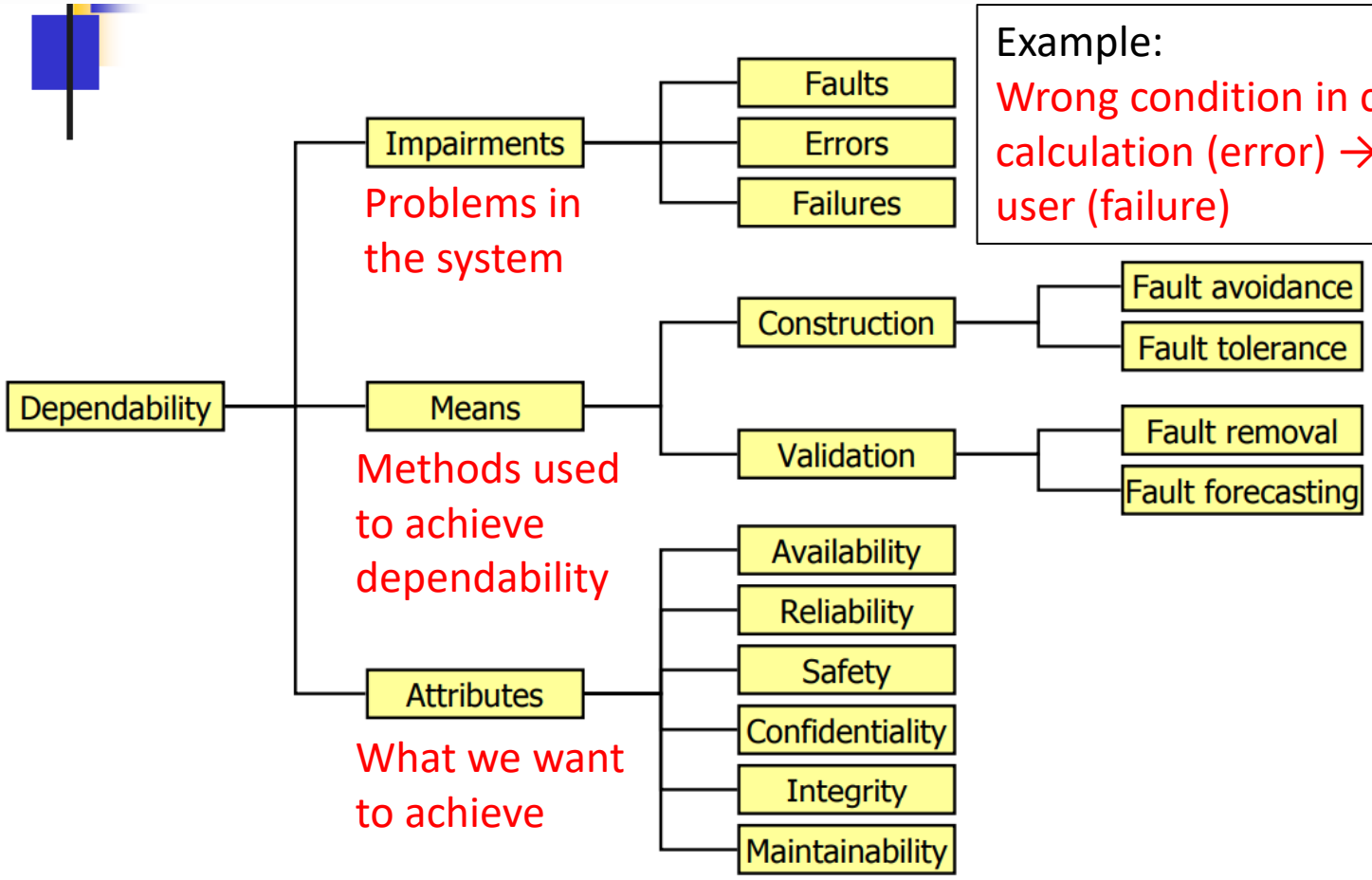
Advantages of focusing on reliability attributes in software development

- **Increased customer satisfaction:** Reliable software systems that are available, fault-tolerant, and recoverable can enhance customer satisfaction by reducing downtime, minimizing errors, and improving overall system performance.
- **Enhanced brand reputation:** A reliable software system can enhance the reputation of the brand and increase customer loyalty by demonstrating a commitment to quality and reliability.
- **Improved system performance:** By focusing on reliability attributes, software developers can identify potential performance issues and address them before they become major problems.
- **Reduced maintenance costs:** Maintainable software systems that are designed with reliability in mind can reduce the cost and effort associated with system maintenance and updates.

Disadvantages of focusing on reliability attributes in software development

- **Increased development time and cost:** Focusing on reliability attributes requires additional time and resources during the software development lifecycle, which can increase development costs and delay time-to-market.
- **Difficulty in balancing reliability and functionality:** Focusing too much on reliability attributes can sometimes result in sacrificing functionality or usability, which can negatively impact the end-user experience.
- **Difficulty in predicting and simulating real-world conditions:** It can be difficult to accurately predict and simulate real-world conditions and usage patterns during the development and testing of software systems, which can limit the effectiveness of reliability testing.

Reliability metrics; Failure and Faults – Prevention, Removal, Tolerance, Forecast Dependability Concept – Failure Behavior, Characteristics, Maintenance Policy



Example:
Wrong condition in code (fault) → incorrect calculation (error) → wrong output shown to user (failure)

Reliability metrics; Failure and Faults – Prevention, Removal, Tolerance, Forecast Dependability Concept – Failure Behavior, Characteristics, Maintenance Policy

- **Dependability is the system property that integrates such attributes as reliability, availability, safety, security, survivability, maintainability.**
- **A systematic exposition of the concepts of dependability consists of three parts: the threats to, the attributes of, and the means by which dependability is attained.**
- **Computing systems are characterized by five fundamental properties: functionality, usability, performance, cost, and dependability.**
- **Dependability of a computing system is the ability to deliver service that can justifiably be trusted.**

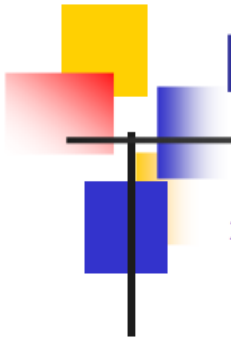
- Impairments: those things that stand in the way of dependability
- Means: the various technical means to achieve dependable software
- The attributes of dependability enable the properties of dependability and provide a way to assess achievement of those properties

Fault

- A fault is the identified or hypothesized cause of an error
- Sometimes called a "bug"
- It can be viewed as simply the "consequence of a failure of some other system that has delivered or is now delivering a service to the given system"
- An active fault is one that produces an error

Error

- An error is part of the system state that is liable to lead to a failure
- It can be unrecognized as an error (latent) or detected
- An error may propagate, i.e., produce other errors
- Faults are known to be present when errors are detected
- An error is the manifestation of a fault



Failure

- A failure occurs when the service delivered by the system deviates from the specified service, otherwise termed as an incorrect result
- The expected service is described, typically by a specification or set of requirements
- The cycle: ...failure → fault → error → failure → fault...

Means to achieve dependable software

- Two major groups
  - Construction: those that are employed during the software construction process
  - Validation: those that contribute to validation of the software after it is developed

Fault avoidance or prevention

- Fault avoidance or prevention techniques are dependability enhancing techniques employed during software development to reduce the number of faults introduced during construction
- These techniques may address:
  - System requirements specification
  - Structured design and programming methods
  - Formal methods
  - Software reuse

System requirement specification

- A system failure may occur due to logic errors incorporated into the requirements
- This results in software that is written to match the requirements, but the behavior specified in the requirements is not the expected or desired behavior
- The specification of system requirements is currently an imperfect process at best

Fault avoidance/prevention summary

- System requirements specification – generally accepted and widely used
- Structured design and programming methods – popular in developing software
- Formal methods – thorough, large overhead, acceptable only in small components – an active research area
- Software reuse – popular and helpful, but simply reusing code does not ensure dependability
- Despite fault prevention efforts, faults are created, so fault removal is needed!

Fault removal

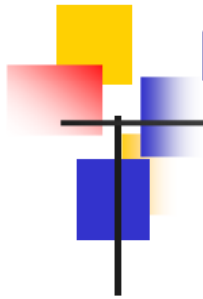
- Fault removal techniques are dependability-enhancing techniques employed during software verification and validation
- Improving software dependability by
  - Detecting existing faults, using verification and validation (V&V) methods
  - Eliminating the detected faults
- Techniques
  - Software testing
  - Formal inspection
  - Formal design proofs

Fault/failure forecasting

- Fault/failure forecasting includes dependability enhancing techniques that are used during the validation of software to estimate the presence of faults and the occurrence and consequence of failures
- Usually focuses on the reliability measure of dependability
- Also known as software reliability measurement

Fault/failure forecasting (cont.)

- It involves
  - The formulation of a fault/error/failure relationship
  - An understanding of the operational environment
  - The establishment of reliability models
  - The collection of failure data
  - The application of reliability models by tools
  - The selection of appropriate models
  - The analysis and interpretation of results
  - Guidance for management decisions

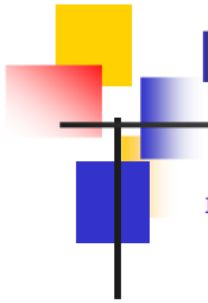


Fault tolerance

- Fault tolerance techniques provide mechanisms to the software system to prevent system failure from occurring when a fault occurs
  - Reduces the risks of software design faults
  - Enables a system to tolerate software faults remaining in the system after its development
- Techniques
  - Single version software techniques
  - Multiple version software techniques
  - Multiple data representation techniques

Single version software environment

- Monitoring techniques
- Atomicity of actions
- Decision verification
- Exception handling



Multiple version software environment

- **Design diverse techniques** are used in this environment
- These techniques utilize functionally equivalent yet independently developed software versions to provide tolerance to software design faults
- Example techniques
  - Recovery blocks (RcB)
  - N-version programming (NVP)
  - N self-checking programming (NCSP)

Software fault tolerance process

- The fault tolerance process is the set of activities whose goal is to remove errors and their effects from the computational state before a failure occurs
- It consists of:
  - Error detection: an erroneous state is identified
  - Error diagnosis: the damage caused by the error is assessed and the cause of the error is determined
  - Error containment/isolation: further damages are prevented
  - Error recovery: the erroneous state is substituted with an error-free state

Reliability metrics; Failure and Faults – Prevention, Removal, Tolerance, Forecast Dependability Concept – Failure Behavior, Characteristics, Maintenance Policy

- **availability**: readiness for correct service,
- **reliability**: continuity of correct service,
- **safety**: absence of catastrophic consequences on the user(s) and the environment,
- **confidentiality**: absence of unauthorized disclosure of information,
- **integrity**: absence of improper system state alterations;
- **maintainability**: ability to undergo repairs and modifications.

Reliability Models

- Main idea
  - We develop mathematical models for **λ(τ)** and **μ(τ).**

- Basic assumptions in developing a reliability model
  - Faults in the program are independent.
  - Execution time between failures is large w.r.t. instruction execution time.
  - Potential test space covers its use space.
  - The set of inputs per test run is randomly chosen.
  - The fault causing a failure is immediately fixed or else its re-occurrence is not counted again.

Reliability Models

- Intuitive idea
  - As we observe another system failure and the corresponding fault is fixed, there will be fewer number of faults remaining in the system and the failure intensity will be smaller with each fault fixed.
  - In other words, as the cumulative failure count increases, **the failure intensity decreases.**
- Two decrement processes
  - Decrement process 1
    - The decrease in failure intensity after observing a failure and fixing the corresponding fault is **constant**.
      - This gives us the Basic model.
  - Decrement process 2
    - The decrease in failure intensity after observing a failure and fixing the corresponding fault is **smaller** than the previous decrease.
      - This gives us the Logarithmic model.

Software Reliability Models (SRGM)


thank
you