Class3: Reliability Engineering Chapter 11

Adeeb Noor, PhD
IT Department
Faculty of Computing and Information Technology
King Abdulaziz University
Fall 2025

Today lecture topic

Lecture 4

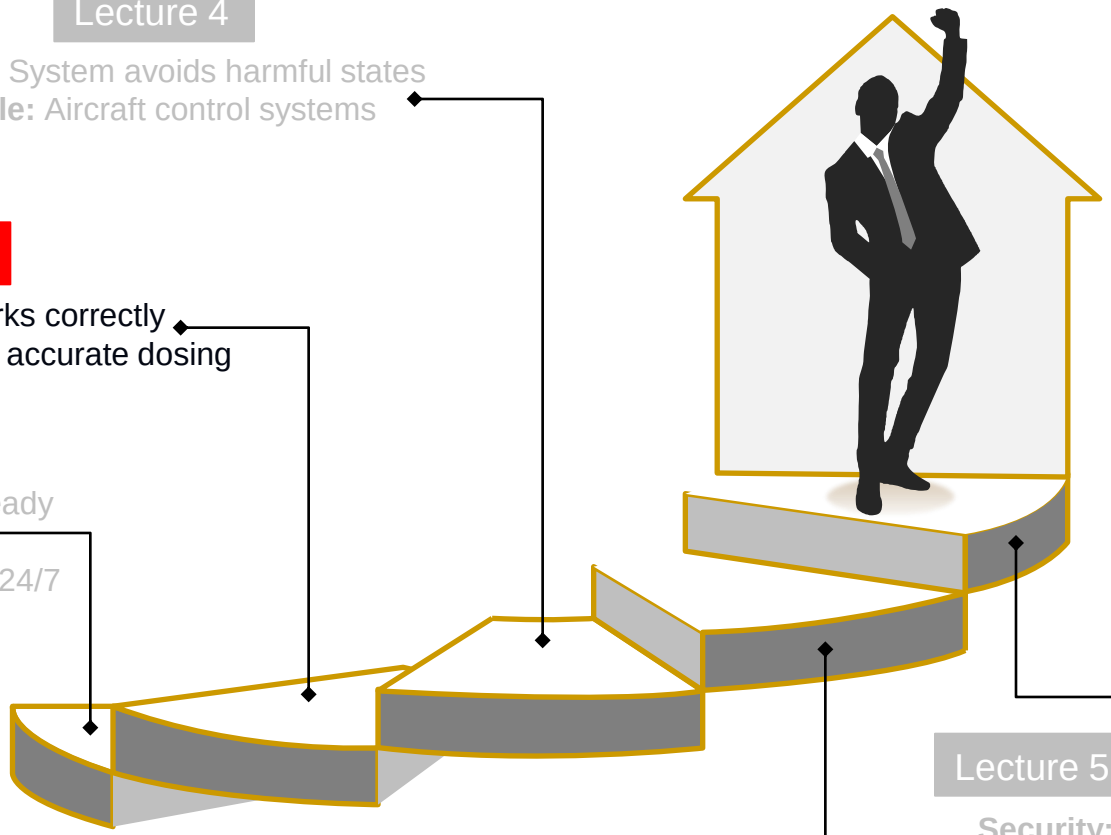
Safety: System avoids harmful states
Example: Aircraft control systems

Lecture 3

Reliability: System works correctly
Example: Insulin pump accurate dosing

Lecture 2

Availability: System is ready when needed
Example: ATM available 24/7



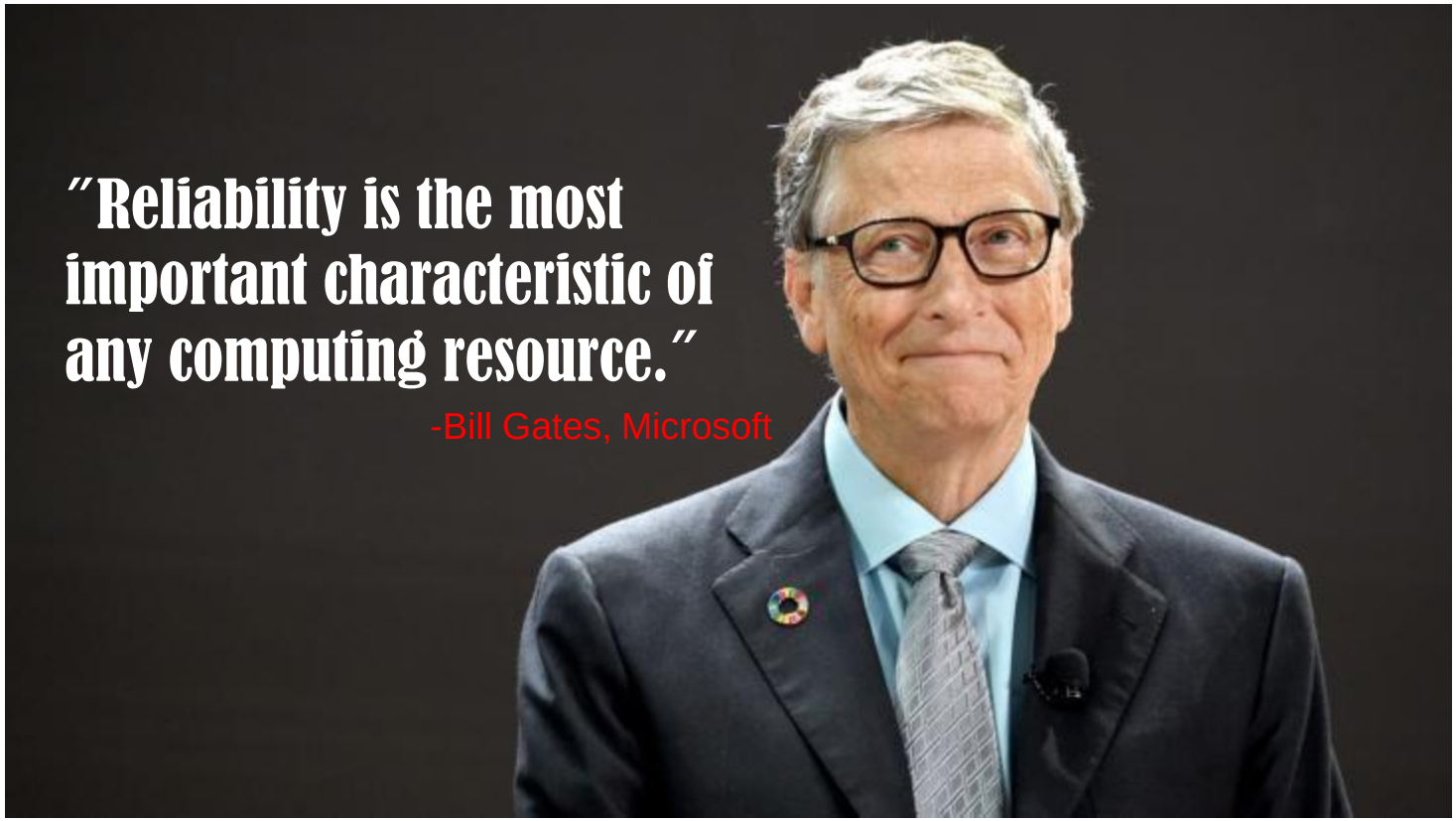
Lecture 6

Resilience: System recovers from failures
Example: Backup power systems

Lecture 5

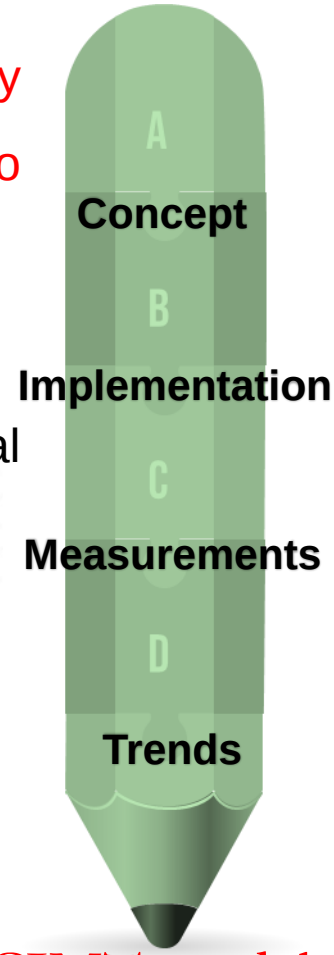
Security: System resists attacks
Example: Secure against fraud

The big picture



Take-home points

- The Key CLOs: write functional and non-functional dependability requirements, and apply metrics for reliability specification to specify measurable reliability requirements
- Reliability is essential but not sufficient for dependability
- Not all systems need the same level of reliability, i.e. non-critical applications may be willing to accept some system failures
- Prevention is cheaper than fixing
- Context determines reliability requirements
- Human error is the root of most system failures
- AI helps companies prevent failures and streamline operations



CIMT model

Software reliability: why matters

Toyota "Unintended Acceleration" Has Killed 89

Electronic control systems with critical software vulnerabilities

Inadequate fail-safe mechanisms, poor software design, insufficient testing

A 2005 Toyota Prius, which was in an accident, is seen at a police station in Harrison, New York, Wednesday, March 10, 2010. The driver of the Toyota Prius told police that the car accelerated on its own, then lurched down a driveway, across a road and into a stone wall. (AP Photo/Seth Wenig) / AP PHOTO/SETH WENIG

Unintended acceleration in Toyota vehicles may have been involved in the deaths of 89 people over the past decade, upgrading the number of deaths possibly linked to the massive recalls, the government said Tuesday.

The National Highway Traffic Safety Administration said that from 2000 to mid-May, it had received more than 6,200 complaints involving sudden acceleration in Toyota vehicles. The reports include 89 deaths and 57 injuries over the same period. Previously, 52 deaths had been suspected of being connected to the problem.

See: https://users.ece.cmu.edu/~koopman/toyota/index.html

It's All Your Fault: The DOT Renders Its Verdict on Toyota's Unintended-Acceleration Scare

The final word on the Toyota unintended-acceleration mess.

CSABA CSERE JUN 9, 2011

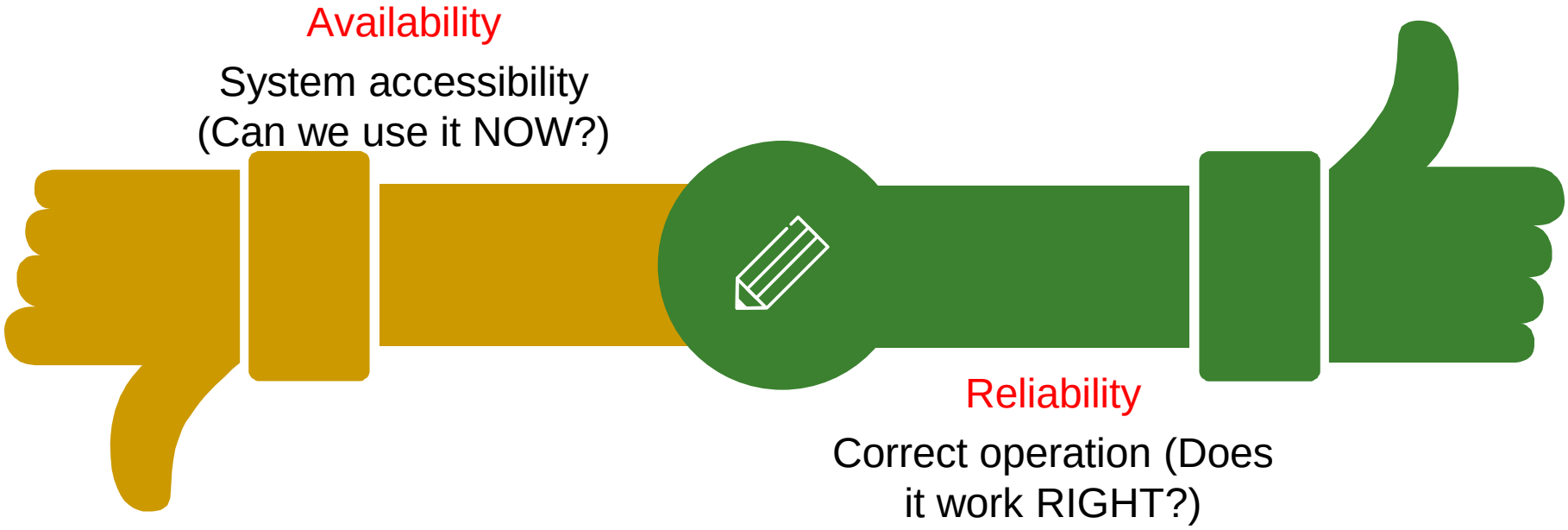
Over 8.1 million vehicles recalled, multiple fatalities

Root cause was lack of comprehensive safety engineering approach

From the June 2011 issue of *Car and Driver*

11.1 What is reliability engineering?

- Making sure software WORKS when you need it
- How likely your software will work correctly
- Measured over specific: time – environment – purpose

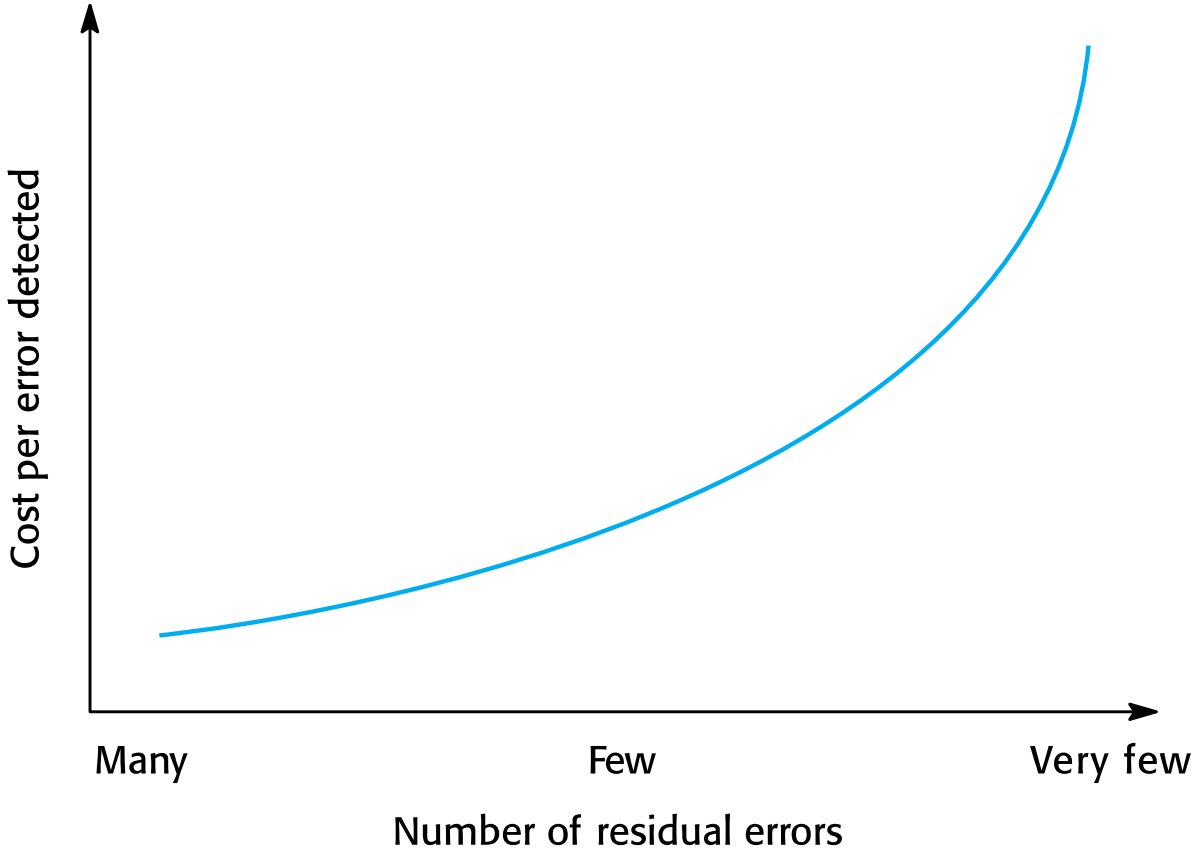


Human often cause failures

Human Error → System Fault → System Error → System Failure

- Human Error: Mistake in writing code (no input check)
- Fault: A problem in the code (accepting negative num)
- Error: Unexpected system behaviour (wrong outputs)
- Failure: The system completely breaks down (system stop to function)

These failures have real costs, so prevention matters



11.2 Reliability requirements

Checking Requirements	Input validation
	Error prevention
	Example: Reject temperatures outside human survival range
Recovery Requirements	System restoration
	Data preservation
	Example: Store transaction logs on multiple servers
Redundancy Requirements	Backup systems
	Failover capabilities
	Example: 5 independent computer systems, operate if 4 fail
Process Requirements	Development standards
	Quality assurance
	Example: Safety-Critical Software Development

Non-Functional vs. functional

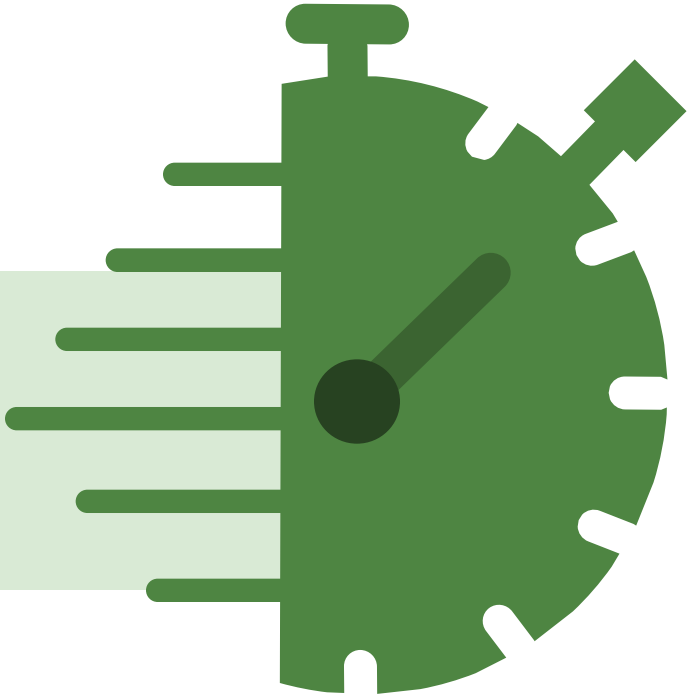
Non-Functional

"how well" the system performs

Functional

"what" the system must do

Reliability metrics explained



01 POFOD (Probability of Failure on Demand)

- Chance of failure when system is used
- Example: 1 in 1000 chances of failure

02 ROCOF (Rate of Occurrence of Failures)

- Number of failures in a time period
- Example: 2 failures per day

03 AVAIL (Availability)

- Percentage of time system is operational
- Example: 99.99% uptime = Super reliable!

Writing Non-Functional Requirements

- TEMPLATE: The [SYSTEM] shall achieve [METRIC] of [VALUE] under [CONDITIONS]

- EXAMPLE 1: ATM System
- System availability shall be 0.999 between 6AM - 10PM
- System shall have POFOD less than 0.001 per transaction

- EXAMPLE 2: Medical System
- System shall achieve ROCOF less than 2 failures year
- Recovery time shall be less than 30 seconds per incident

Writing Functional Requirements

- Checking Requirements:
  - All user inputs must be validated within [TIME] seconds
  - System must verify data integrity before processing
- Recovery Requirements
  - System must auto-save user data every [TIME] minutes
  - System must maintain backup copy of all transactions
- Redundancy Requirements
  - Critical operations must have backup processing path
  - System must replicate data across multiple servers

Examples of functional reliability requirements

RR1: A pre-defined range for all operator inputs shall be defined and the system shall check that all operator inputs fall within this pre-defined range. (Checking)

RR2: Copies of the patient database shall be maintained on two separate servers that are not housed in the same building. (Recovery, redundancy)

RR3: N-version programming shall be used to implement the braking control system. (Redundancy)

RR4: The system must be implemented in a safe subset and checked using static analysis. (Process)

11.3 Fault-tolerant architectures

01 Protection Systems

- Backup system that activates if main system fails
- Ensures passenger safety via AUTOMATICALLY applies brakes

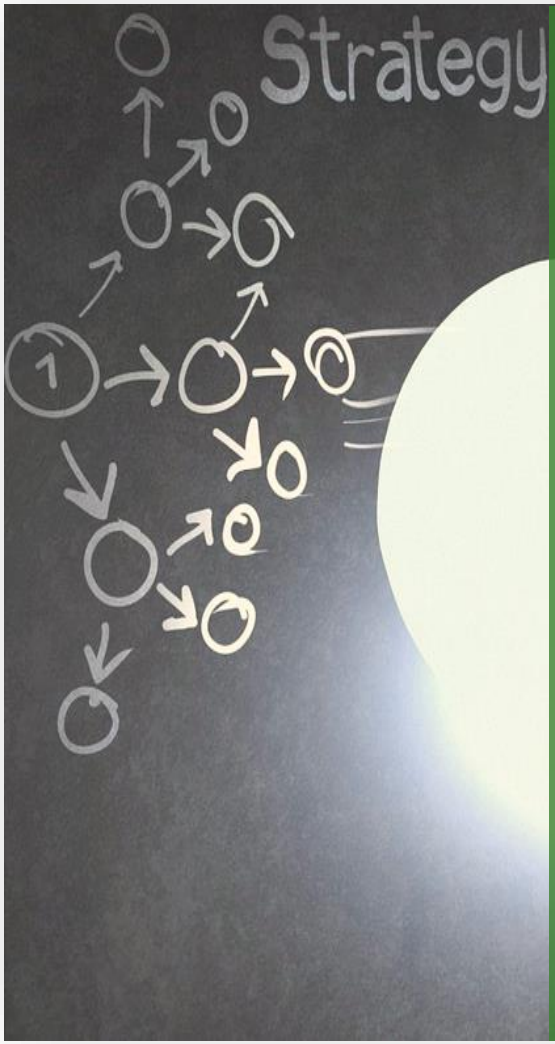
02 Self-Monitoring Architectures

- Multiple versions running simultaneously
- Constantly check each other

03 N-Version Programming

- Multiple independent software versions
- Extremely safety-critical systems

11.4 Programming for reliability



1. Limit the visibility of information in a program, get () and put ()
2. Check all inputs for validity, study all kinds of inputs
3. Provide a handler for all exceptions, apply fault tolerance
4. Minimize the use of error-prone constructs, organize and test
5. Provide restart capabilities, keep copies of work
6. Check array bounds, bounded buffer
7. Include timeouts when calling external components, distributed systems services
8. Name all constants that represent real-world values, make variables easy to understand

Reliability programming example

```java
public class ReliableSystem {
    // Concept 1: "Limit visibility of information, get() and put()"
    // Concept 2: "Check all inputs for validity, study all kinds of inputs"
    public static double validateTemperature(double temp) {
        if (temp < -50.0 || temp > 100.0) {
            throw new IllegalArgumentException("Temperature out of range");
        }
        return temp;
    }

    // Concept 3: "Provide a handler for all exceptions, apply fault tolerance"
    // Concept 5: "Provide restart capabilities, keep copies of work"
    public static void saveData(String data) {
        try {
            writeToPrimary(data);
        } catch (Exception e) {
            writeToBackup(data);  // Backup system for fault tolerance
        }
    }

    // Concept 4: "Minimize use of error-prone constructs"
    // Concept 6: "Check array bounds, bounded buffer"
    public static int getReliableReading(int s1, int s2, int s3) {
        return (s1 == s2 || s1 == s3) ? s1 :
               (s2 == s3) ? s2 : (s1 + s2 + s3) / 3;
    }

    // Concept 7: "Include timeouts when calling external components"
    // Concept 8: "Name all constants that represent real-world values"
    public static String getData(String url) {
        Future<String> future = executor.submit(() -> fetchData(url));
        try {
            return future.get(5000, TimeUnit.MILLISECONDS); // Named timeout
        } catch (TimeoutException e) {
            throw new RuntimeException("Operation timed out");
        }
    }
}
```

11.5 Reliability measurement

1- Identify operational profile:
- Understand real-world usage patterns
- Create operational profiles

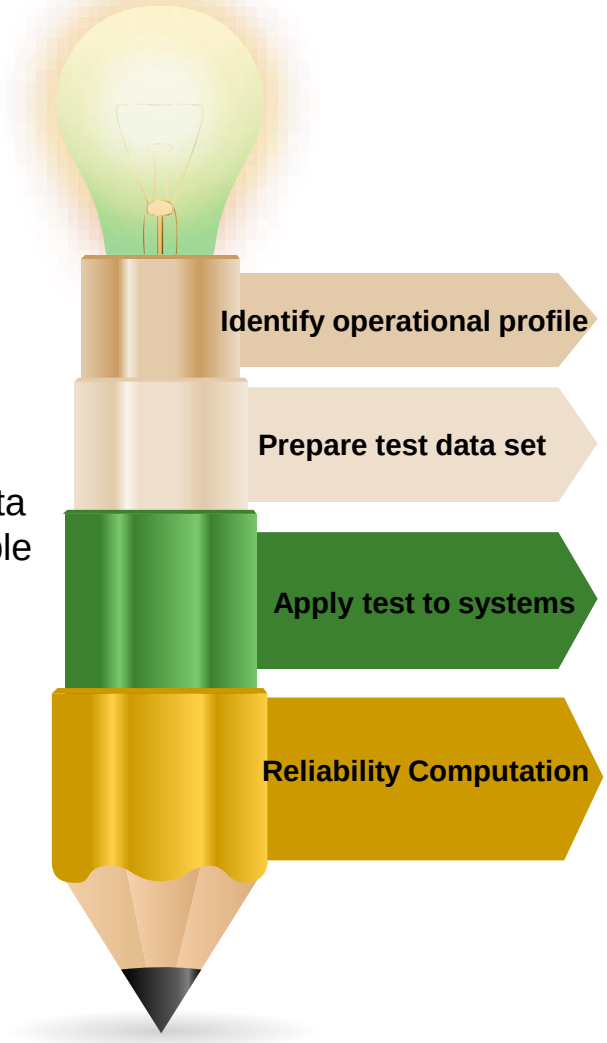
2- Prepare test data set:
- Generate inputs matching real-world distribution
- Use automated test data generators

3- Apply test to systems:
- Count and log system failures
- Measure time between failures
- Track failure types

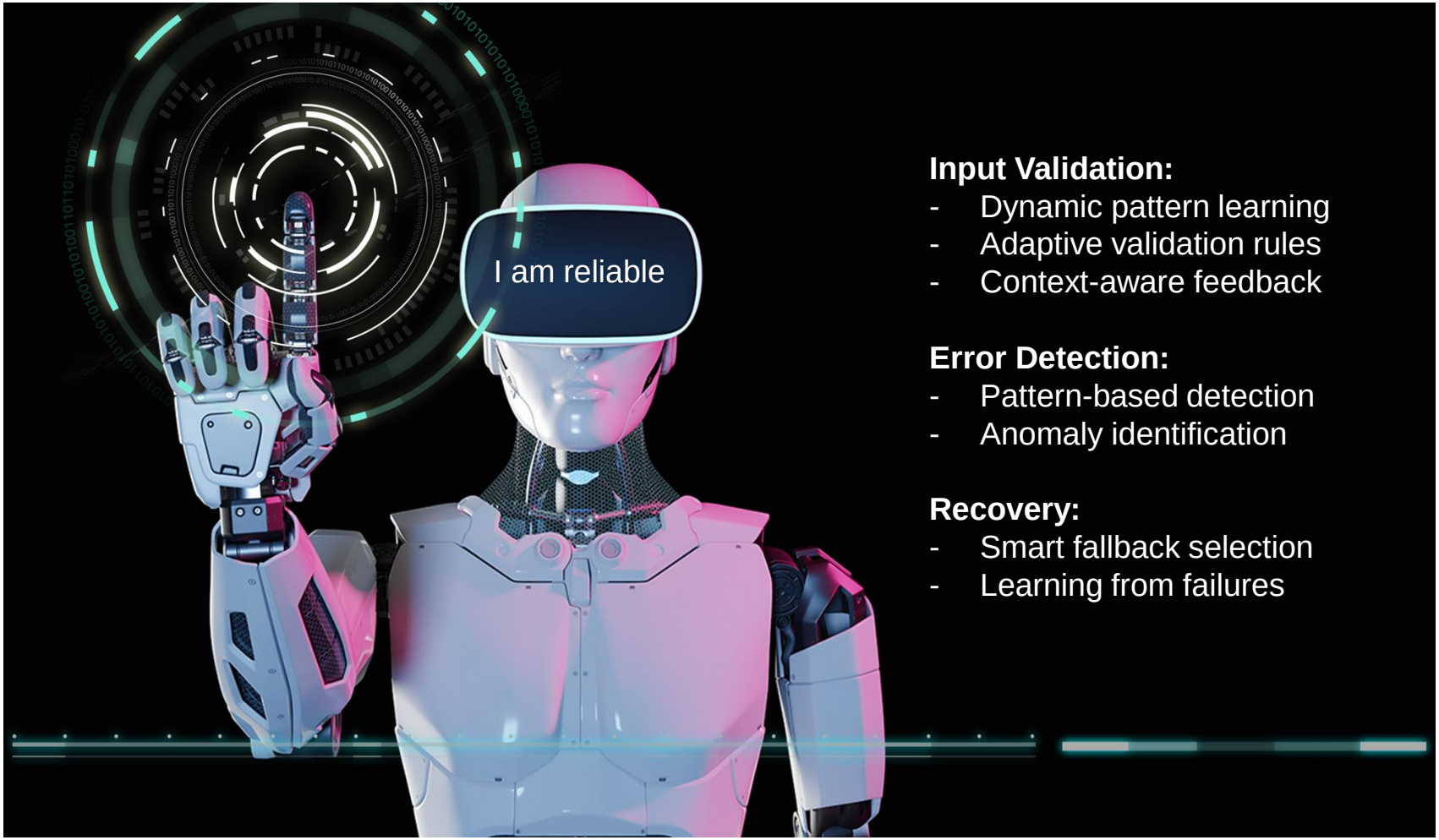
4- Reliability Computation:
- Calculate failure probabilities
- Assess system reliability metrics

Challenges

- Generating comprehensive test data
- Measuring highly reliable systems
- Recognizing what constitutes a failure



Reliability engineering in AI era



Input Validation:
- Dynamic pattern learning
- Adaptive validation rules
- Context-aware feedback

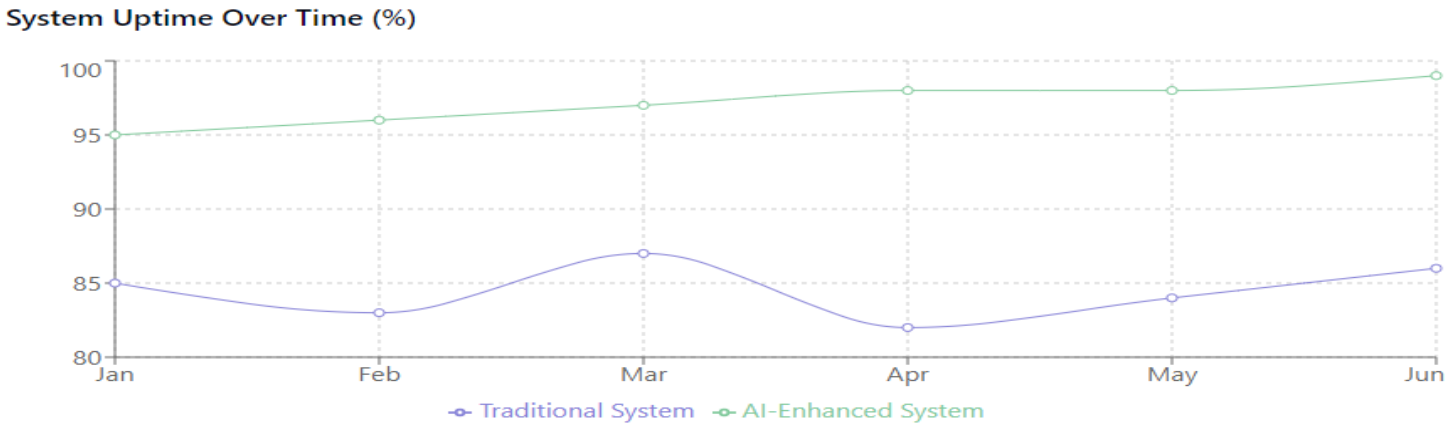
Error Detection:
- Pattern-based detection
- Anomaly identification

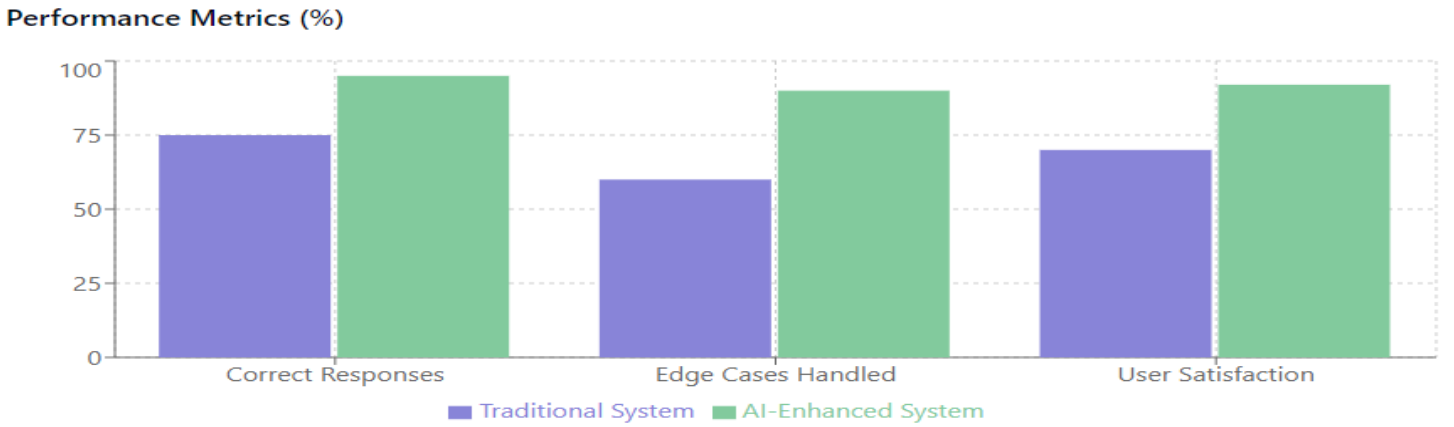
Recovery:
- Smart fallback selection
- Learning from failures

Reliability requirements & AI

Checking Requirements	Input validation - **Smart Input Validation**
	Error prevention - **Use LLM to check input context**
	Example: Reject temperatures outside human survival range
Recovery Requirements	System restoration - **Intelligent Error Handling**
	Data preservation - **LLM suggests fixes for common errors**
	Example: Store transaction logs on multiple servers
Redundancy Requirements	Backup systems - **Multiple LLM providers as backup**
	Failover capabilities - **Automatic fallback on failure**
	Example: 5 independent computer systems, operate if 4 fail
Process Requirements	Development standards
	Quality assurance
	Example: Safety-Critical Software Development

Ex; AI enhancement customer service bot





To master today's lesson

- Assignment 2 is designed to assess your mastery of second and third (CLOs): Write functional and non-functional dependability and security requirements, and apply metrics for reliability specification to specify measurable reliability requirements
- Total Marks: 5
- Submission Deadline: 8:00 PM every Saturday. No late submissions accepted
- You can use any LLM (ChatGPT, Claude, etc.), but you must follow the IT department's LLM ethics guidelines
- Collaboration allowed but submit individual work