# Homework 1

ECE 590: Towards More Reliable Software

Jeff Fan || zf70@duke.edu

January 15, 2024

# Question 1:

**a. List all the attributes of software dependability and explain their meanings.**

**Availability:** This refers to the readiness of a system for correct service. It focuses on ensuring that the system is accessible and usable when needed.

**Reliability:** This attribute is concerned with the continuity of correct service. It measures how consistently a system performs its intended function without failure over time.

**Safety:** Safety is the attribute that emphasizes the absence of catastrophic consequences on the users and the environment. It's about ensuring that the system does not cause harm or adverse effects.

**Integrity:** This pertains to the absence of improper system alterations. Integrity ensures that the system's data and operation are not unlawfully or erroneously modified or tampered with.

**Maintainability:** This is the ability of the system to undergo modifications and repairs. Maintainability makes it easier to update, improve, and fix the system as needed.

**Confidentiality:** This is defined as the absence of unauthorized disclosure of information. It's crucial for protecting sensitive information from unauthorized access or exposure.

**b. Which do you think is the best means to achieve software dependability in the above paper and why?**

In my view, I consider fault tolerance as the best means to achieve software dependability based on the content of the paper. Fault tolerance is primarily aimed at failure avoidance, meaning it allows a system to continue operating correctly even in the presence of faults. This is crucial for systems where high availability and reliability are non-negotiable, such as in critical infrastructure systems or financial services. It also includes a range of techniques like error detection, system recov-

ery, rollback, rollforward, and compensation. These methods ensure that not only are errors detected, but the system can also recover from them, thus maintaining service continuity.

Fault tolerance techniques often include the concept of error confinement areas, which help in localizing the impact of a fault. Additionally, the effectiveness of a fault tolerance technique is measured by its coverage, or how well it can handle a range of fault scenarios. While fault tolerance is effective on its own, it also complements other dependability strategies like fault prevention, removal, and forecasting. This interplay enhances overall system dependability, making fault tolerance a key component in a comprehensive dependability strategy.

**c. Use your own words to describe the different definitions of fault, error and failure.**

Fault: Imagine a fault as a weak link in a chain. It's a defect or flaw within the system, but it hasn't yet caused a problem. The chain is still holding together, but the weak link represents a potential point of failure. In technical terms, a fault is a defect in a system's hardware, software, or design that has the potential to cause an error. It's like a dormant seed that, under the right conditions, can sprout into a bigger issue.

Error: Using the chain analogy, an error occurs when the weak link starts to give way under stress. The chain hasn't broken yet, but there's a clear sign that something is wrong – the link is deforming or starting to crack. In the context of computing systems, an error is the manifestation of a fault. It's a deviation from the correct or expected behavior of the system. The system hasn't failed yet, but there's an incorrect state within it, like corrupted data or a malfunctioning process.

Failure: Finally, a failure happens when the weak link in the chain breaks, causing the entire chain to fail in its function of holding something together. In a computing system, a failure is the point at which the system can no longer deliver the expected service to the user. It's the observable consequence of an error, where the system's operation deviates from what's expected, like a system crash or incorrect output.

To sum up, a fault is like a weak link in a chain – a potential problem; an error is like the weak link starting to give way – a problem in progress; a failure is like the chain breaking due to the weak link – a complete breakdown in operation.

# Question 2:

**Take any program that you have written and identify at least one fault (bug), the errors it will cause, the inputs that will trigger the fault and failure that results. How can you avoid them next time?**

```
1  import java.io.*;
2  import java.net.ServerSocket;
3  import java.net.Socket;
4  import java.lang.reflect.Method;
5
6  public class SocketServer {
7    public static void main(String[] args) throws Exception {
8      ServerSocket serverSocket = new ServerSocket(8080);
9      System.out.println("Server started. Listening on port
   8080...");
10
11       while (true) {
12         try (Socket clientSocket = serverSocket.accept();
13         BufferedReader in = new BufferedReader(new
   InputStreamReader(clientSocket.getInputStream()));
14         PrintWriter out = new PrintWriter(clientSocket.
   getOutputStream(), true);) {
15
16           String inputLine = in.readLine();
17           Method method = SocketServer.class.getMethod(inputLine
   );
18           String response = (String) method.invoke(null);
19           out.println(response);
20         } catch (Exception e) {
21           e.printStackTrace();
22         }
23       }
24    }
25
26    public static String getTime() {
27      return "The current time is " + System.currentTimeMillis()
   ;
28    }
29 }
```

**Fault:** The potential fault in this program is the lack of validation of the input line (line 17), which is used directly to reflectively call a method. If the input does not correspond to a method name, or if it is a method name not intended to be called, this could lead to errors.

**Error:** The primary error this fault can cause is a **NoSuchMethodException** if an input string is received that doesn't correspond to any method in the class. Additionally, if the input is malicious, it could potentially call unintended methods (if they exist and are public and static), leading to security vulnerabilities.

**Trigger Inputs:** Any input string that is not exactly "getTime" will trigger the fault. For example, inputs like "gettime", "Time", or "randomString" will cause the program to throw an exception.

**Failure:** The failure resulting from this error would be the server crashing or behaving unexpectedly, potentially exposing the system to security risks. The server might also respond with an error message or stack trace to the client, which is undesirable in a production environment.

**How to Avoid:**

a. Always validate user inputs, especially when using them in sensitive operations like reflection. In this case, a whitelist of allowed method names could be used to validate the input before attempting to invoke a method.

b. We can enhance the error handling by specifically catching NoSuchMethodException and returning a user-friendly message. For example

```
...
    Set<String> allowedMethods = Set.of("getTime");
    while (true) {
      try (Socket clientSocket = serverSocket.accept();
      BufferedReader in = new BufferedReader(new
 InputStreamReader(clientSocket.getInputStream()));
      PrintWriter out = new PrintWriter(clientSocket.
 getOutputStream(), true);) {

        String inputLine = in.readLine();
        try {
          if (allowedMethods.contains(inputLine)) {
            Method method = SocketServer.class.getMethod(
 inputLine);
            String response = (String) method.invoke(null);
            out.println(response);
          } else {
            out.println("Invalid method request.");
          }
        } catch (NoSuchMethodException e) {
          out.println("Requested method not found.");
        } catch (Exception e) {
          out.println("An error occurred.");
```

```
            e.printStackTrace(); // For debugging purposes
        }
...
```

c. We can also restrict the use of reflection as it can be a security risk. If reflection is necessary, ensure it's tightly controlled and secure. Employ principles like least privilege and secure coding practices.

d. We can implement proper logging to record unusual activities or inputs. Monitoring these logs can help in quickly identifying and rectifying issues.

# Question 3:

**Find a well-publicized incident, read about it and determine, the fault(s), errors and failure of this accident. And find out how the engineers react after the incident happens?**

The Target data[1] breach of 2013 is a famous example of a cybersecurity incident.

**Faults:** The initial fault lay in the security systems of both Target and its third-party vendor, Fazio Mechanical. The vendor's lack of malware detection software and the failure of both companies to properly segment their networks created vulnerabilities.

**Errors:** The errors occurred when cybercriminals successfully executed a phishing scam to gain network access, installed malware on Target's point-of-sale systems, and began collecting customer data. The malware's presence was detected, but Target initially failed to act on these alerts.

**Failure:** The failure was the large-scale compromise of approximately 40 million customers' credit/debit card information and 70 million customers' personal details (e.g., names, addresses, phone numbers), resulting in significant financial and reputational damage to Target.

**Engineers' Response:** Post-breach, Target undertook several recovery steps including working with a forensics firm, offering credit monitoring to customers, upgrading their point-of-sale systems, and implementing better network segmentation and access controls. They faced over 140 lawsuits, an $18.5 million settlement, and changes in executive leadership due to the breach.

[1] target investigating data breach. https://krebsonsecurity.com/2013/12/sources-target-investigating-data-breach/

# Question 4:

**Develop and describe a classification of errors.**

**1. System-Level Errors**

**Hardware Failures:** Malfunctions in physical components of the system, like hard drive crashes or memory corruption.

**Operating System Errors:** Issues within the operating system that affect software execution, such as file system errors or driver conflicts.

**2. Application-Level Errors**

**Runtime Exceptions:** Errors that occur during the execution of an application, like null pointer exceptions or out-of-bounds errors.

**Logic Errors:** Flaws in the application's logic leading to incorrect behavior or results, even though the program runs without crashing.

**Data Processing Errors:** Issues in handling and processing data, such as incorrect data parsing or arithmetic errors.

**Concurrency Issues:** Problems arising in multi-threaded applications, like race conditions or deadlocks.

**Resource Leaks:** Inadequate management of resources leading to issues like memory leaks or file descriptor exhaustion.

**3. Interface-Level Errors**

**API Misuse:** Incorrect use of APIs or failure to handle API errors gracefully.

**Integration Problems:** Issues arising when integrating with other software components, services, or databases.

**Network Communication Errors:** Problems in the network layer, such as failure to handle lost connectivity or delayed responses.

**4. User Interface-Level Errors**

**Input Validation Errors:** Failure to properly validate user inputs leading to incorrect or unexpected behavior.

**Navigation Flaws:** Issues in the UI flow that confuse users or lead them to dead ends.

**Display Issues:** Problems with how information is presented on the screen, like overlapping text or incorrect data representation.

**5. Configuration-Level Errors**

**Setup and Configuration Mistakes:** Errors due to incorrect setup or configuration parameters of the software or its environment.

**Dependency Issues:** Problems arising from incompatible or missing software dependencies.

**6. Security-Level Errors**

**Vulnerabilities:** Flaws that compromise the security of the application, like buffer overflows or SQL injection vulnerabilities.

**Access Control Issues:** Problems related to authentication and authorization, like privilege escalation or unauthorized access.

**7. Development and Build-Level Errors**

**Compilation Errors:** Issues that prevent the source code from successfully compiling, like syntax errors or unresolved references.

**Build and Deployment Mistakes:** Errors in the build pipeline or deployment process, leading to incorrect or failed deployments.

Here I draw the figure to clearly show you the classification: