Module 3: Software Bugs

CYBR 515: Software Security
Summer 2024

Mohammad Jamil Ahmad, Ph.D.

WestVirginiaUniversity.
JOHN CHAMBERS COLLEGE OF BUSINESS AND ECONOMICS

Table of Contents

- Why do software fail?
- Attributes of good software
- Software failures
- Evolution of software issues
- Version Control Systems
- Static code analysis demo

Software Dysfunction: Why Do Software Fail?

- **Context**: Software is pervasive but often unnoticed until problems arise.
- **Importance**: It is labor-intensive, complex, and *error-prone.*
- **Focus**: The studies on eCampus explore reasons for software failures.
  - Also, provide guidelines on how to discover those

Attributes of Good Software

1. **Functionality**: Delivers required functionality and performance.
2. **Maintainability**: Easy to maintain and evolve.
3. **Dependability**: Reliable, secure, and safe.
4. **Efficiency**: Optimizes system resource usage.
5. **Acceptability**: Understandable, usable, and compatible.
6. **Survivability**: Adapts to various environments.

Are all software equally important?

1. **Safety-critical**: Protects human lives (e.g., medical devices, aircraft controls).
2. **Mission-critical**: Essential tasks (e.g., financial transactions, telecommunications).
3. **Business-critical**: Protects confidential information (e.g., banking, military applications).

What Is Software Failure?

- **Definition**: Inability of a system or component to perform its required functions.
- **Concepts**:
  - **Fault/Defect**: Incorrect step causing incorrect results.
  - **Failure**: System's inability to deliver expected service.
  - **Error**: Difference between computed and correct values.

Classification of Failures

1. **Unplanned Events**: Crashes, hangs, incorrect or no output.
2. **Planned Events**: Scheduled system shutdowns for maintenance.
3. **Configuration Failures**: Errors due to configuration settings.

Causes of Software Failure

1. **Lack of Design**: Poor or no design before coding.
2. **Inadequate Testing**: Insufficient testing before release.
3. **Attitudinal Changes**: Shift from meticulous planning to “code and fix” approach.
4. **Incompatibilities from Software Changes**: Introduction of new errors.
5. **Hostile Agent Attacks**: Intentional changes to cause failure.
6. **Unanticipated Applications**: Use cases not foreseen by developers.

Causes of Software Failure

7. **Complexity**: Increased complexity leads to more potential for errors.
8. **Human Error**: Mistakes made by developers and users.
9. **External Factors**: Environmental conditions, cyber attacks.

Other Human and External Factors

- **Human Error**: Inappropriate use, incorrect data input.
- **Management Laxity**: Ignoring warnings due to cost concerns.
- **Support Systems**: Dependency on reliable hardware and power.
- **Cyber Security**: Vulnerabilities exploited by threats.
- **Environmental Factors**: Natural disasters affecting systems.

Examples of Critical Software Failures

- **Ariane-5 Rocket**: Software error leading to rocket destruction.
- **Therac-25**: Radiation overdose causing deaths.
- **US Navy Ship Yorktown**: Improper input leading to system failure.
- **Mars Climate Orbiter**: Unit conversion error causing loss.
- **US Telephone Networks**: Software issues causing network failures.

Evolution of Software Issues

- **Software Errors:** Mistakes in code or logic
  - Syntax errors, logical errors.
- **Software Faults:** Incorrect steps or data causing incorrect results.
  - Miscalculation in algorithms.
- **Software Failures:** Inability to perform required functions.
  - System crashes, incorrect outputs.
- **Software Bugs:** Flaws in the software causing unintended behavior.
  - Infinite loops, memory leaks.
- **Software Vulnerabilities:** Weaknesses that can be exploited.
  - SQL injection, buffer overflow.

Comparison Table: Errors vs. Faults vs. Failures vs. Bugs vs. Vulnerabilities

Term	Definition	Cause	Impact	Example
Error	A human mistake made during software development.	Human actions (e.g., coding, design errors)	Leads to faults if not corrected.	Incorrectly implemented algorithm due to misunderstood requirements.
Fault	A flaw in the software resulting from an error.	Error in code or logic	Can cause software to behave incorrectly.	Misplaced decimal point causing incorrect calculations.
Failure	The inability of software to perform its required functions.	Activation of a fault	Results in system malfunction.	System crash due to division by zero.
Bug	An issue in the software causing incorrect or unexpected results.	Fault in the code	Affects software functionality.	Infinite loop in a program due to incorrect loop condition.
Vulnerability	A weakness in the software that can be exploited to cause harm.	Security flaw or bug	Can be exploited by attackers.	SQL injection vulnerability due to improper input validation.

How all of those are related to eachother?

- **Error to Fault:** An error leads to a fault if it results in incorrect code.
- **Fault to Failure**: A fault leads to a failure when it affects the system's functionality.
- **Failure to Bug**: A failure is recognized as a bug when it is reported and documented.
- **Bug to Vulnerability:** A bug becomes a vulnerability if it can be exploited.

How to detect each?

Detecting	Detecting Errors: Linting tools, code reviews. •Using ESLint for JavaScript.
Detecting	Detecting Faults: Static analysis tools, formal methods. •Using static analysis tools like FindBugs.
Detecting	Detecting Failures: Automated testing frameworks, continuous integration. •Using JUnit for automated testing in Java.
Detecting	Detecting Vulnerabilities: Security scanners, penetration testing. •Using OWASP ZAP for web application security scanning.

Python example

```python
# Error: Developer mistakenly assumes denominator will never be zero
def divide(a, b):
    return a / b  # Fault: No check for division by zero

# Fault in code: Incorrect handling of zero division
try:
    result = divide(10, 0)
except ZeroDivisionError as e:
    print("Caught an exception:", e)

# Failure: When zero is passed as the denominator, it causes a failure
# Output: Caught an exception: division by zero

# Bug: This scenario would be reported as a bug in the issue tracker

# Vulnerability: Improper input validation can be exploited
def secure_divide(a, b):
    if b == 0:
        raise ValueError("Denominator cannot be zero")
    return a / b

# Fix: Properly handle the zero division case
try:
    result = secure_divide(10, 0)
except ValueError as e:
    print("Caught an exception:", e)

# Output: Caught an exception: Denominator cannot be zero
```

```
Caught an exception: division by zero
Caught an exception: Denominator cannot be zero
```