Lecture 5
Fault Removal

ECE 422: Reliable and Secure Systems Design

Instructor: An Ran Chen
Term: 2024 Winter

Schedule for today

- Key concepts from last class
- An alternative solution, fault removal
  - Functional testing
    - Example of state-of-the-art
  - Structural testing
    - Effectiveness
- System dependability
- Dependability concept summary
- Questions

How to improve availability?

To increase availability, there are two solutions:

- Reduce the frequency of faults
  - By upgrading the hardware, expensive
  - By continuously improving the software
    - slow and expensive
    - after the fault occurs
  - Challenge: we can never reduce the probability of faults to zero
- Design systems that continue to work despite some faulty components; this solution is called Fault Tolerance

Fault tolerance techniques

Fault tolerance techniques provide mechanisms to the software system to prevent failures in the present of a fault.

It consists of:

- Error detection: identification of an error state
- Error diagnosis: assessment of the damage caused by the error
- Error confinement: prevention of further damages
- Error recovery: replacing the error state with an error-free state

Classification of fault tolerance techniques

- Single version techniques
  - Exception handling
- Multiple version techniques: use two or more versions of the same software module to achieve fault tolerance through software redundancy.
  - Recovery blocks
  - N-version programming
  - N self-checking programming
- Multiple data representation techniques (not covered in class)
  - Retry blocks
  - N-copy programming

What makes fault tolerance important?

It is practically impossible to build a perfect system, because each individual component in the system has its own reliability measure.

- Reliability is the likelihood that a system will perform its function without failure.
  - Calculated by MTBF (Mean Time Between Failure)
- System reliability is based on the reliability of all its components.
  - Calculated based on the following formula:

$$R = (1 - F1) * (1 - F2) * (1 - F3) * (1 - FX)....$$

where R refers to the overall system reliability, F1 refers to the failure rate of the first component and so on.

What makes fault tolerance important?

It is practically impossible to build a perfect system, because each individual component in the system has its own reliability measure.

- Reliability is the likelihood that a system will perform its function without failure.
  - Calculated by MTBF (Mean Time Between Failure)
- S… …e reliability of all its components.
  - Calc… …based on the following formula:

1 - failure rate = reliability

$$R = (1 - F1) * (1 - F2) * (1 - F3) * (1 - FX)....$$

where R refers to the overall system reliability, F1 refers to the failure rate of the first component and so on.

What makes fault tolerance important?

System reliability formula:

$$R = (1 - F1) * (1 - F2) * (1 - F3) * (1 - FX)....$$

Example:

- Suppose each component in a system has the reliability 99.99%
- A system consisting of 100 (non-redundant) components will have the reliability 99.01%
- A system consisting of 10,000 components will have the reliability 36.79%

As the complexity of a system grows, its reliability significantly decreases.

Schedule for today

- Key concepts from last class
- An alternative solution, fault removal
  - Functional testing
    - Example of state-of-the-art
  - Structural testing
    - Effectiveness
- System dependability
- Dependability concept summary
- Questions

How to improve availability?

To increase availability, there are two solutions:

- Reduce the frequency of faults
  - By upgrading the hardware, expensive
  - By continuously improving the software
    - slow and expensive
    - after the fault occurs
  - Challenge: we can never reduce the probability of faults to zero
- Design systems that continue to work despite some faulty components; this solution is called Fault Tolerance

How to improve availability?

To increase availability, there are two solutions:

- Reduce the frequency of faults; This solution is called Fault Removal
  - By upgrading the hardware, expensive
  - By continuously improving the software
    - slow and expensive
    - after the fault occurs
  - Challenge: we can never reduce the probability of faults to zero
- Design systems that continue to work despite some faulty components; this solution is called Fault Tolerance

Fault removal

- Why? Despite fault tolerance efforts, not all faults are tolerated, so we need fault removal.
  - To keep in mind: fault tolerance is a must-have property for safety-critical systems.
  - But for software that we use everyday, we want to remove faults to improve user experience.
- Improving system dependability by:
  - Detecting existing faults through software verification and validation
  - Eliminating the detected faults

Fault removal as two concepts:

1. as a solution to improve availability, and thus dependability
2. as a solution for faults that affect user's daily activities (not tolerated)

Software testing

Software testing is one of the most common fault removal techniques.

There are two types of software testing:

- Functional testing (black-box testing)
- Structural testing (white-box testing)

We do not know anything, system as a black box

We know everything, system as a white box

Image from careerist

Schedule for today

- Key concepts from last class
- An alternative solution, fault removal
  - Functional testing
    - Example of state-of-the-art
  - Structural testing
    - Effectiveness
- System dependability
- Dependability concept summary

Functional testing

Functional testing checks the functionality of a program works as per the software requirements.

- In other words, functional testing checks for design errors.
- Also known as black-box testing as no knowledge of the internal code is required.

Example of functional testing:

- Suppose we test a program which adds two integers
- A functional test verifies the implemented operation is indeed addition, rather than multiplication.

Functional testing

- Functional testing: functionality + requirements

Why?

- Does the swing on the tree functions as a swing?
  - Yes
- Does the swing satisfies the user requirements?
  - Can the user use with swing?
  - No

Images from www.projectcartoon.com

Types of functional testing

Five types of functional testing (covered in class):

- Unit testing
- Integration testing
- Acceptance testing
- Regression testing
- … and more
- Recovery testing (not covered in class)
- Sanity testing (not covered in class)
- Smoke testing (not covered in class)

Types of functional tests

- Unit testing allows developers to verify the quality and reliability of individual units of the code are working as expected.
- Integration testing verifies that the different units of the code as a combined entity is reliable.
- Acceptance testing verifies whether a component satisfies the user requirements / reliable from the end user's perspective.
    - In modern software development, it is part of agile methodology.
- Regression testing ensures the software reliability by identifying unintended side effects (e.g., new bugs) after code changes.

Example of state-of-the-art

For web/Java applications, Selenium can be used for functional testing to detect and remove errors/faults.

Acceptance Test 1:

- Open browser
- Enter username <username>
- Enter password <password>
- Press submit button
- Assert text <Course Equivalency System>

Schedule for today

- Key concepts from last class
- An alternative solution, fault removal
  - Functional testing
    - Example of state-of-the-art
  - Structural testing
    - Effectiveness
- System dependability
- Dependability concept summary

Structural testing

Structural testing checks the internal structure or internal implementation of a program for errors.

- Structural testing focuses on how the system is doing rather than the functionality
- Also known as white-box testing as the access of the code is required

Example of structural testing:

- Suppose we test a program which adds two integers
- A structural test checks the “steps that lead to the sum being calculated”, rather than whether the returned output is correct.

Types of structural testing

Examples of structural testing:

- Mutation testing: assess the quality of the test cases which should be robust enough to fail mutant code.
- Data flow testing: examines the data flow with respect to the variables used in the code.
- Control flow testing: examines the execution paths in the code.

Structural testing effectiveness

The effectiveness of structural testing is expressed in terms of test coverage metrics which measure the portion of code exercised by tests.

- Statement coverage = (Number of Statements Exercised / Total Number of Statements) x 100 %
- Branch coverage = (Number of Decisions Outcomes tested / Total Number of Decision Outcomes) x 100 %
- Path coverage = (Number Paths Exercised / Total Number of Paths in the Program) x 100 %

Statement coverage

100% Statement coverage requires that each executable statement of a program is executed at least once during a test.

- Statement coverage only checks whether the loop body was executed or not
- It does not report whether loops reach their termination condition
- Statement coverage is insensitive to some control structures
    - E.g., && or || operators, switch statements

Example

```
x = 0;
if (condition)
   x = x + 1;
y = 10/x;
```

Test 1 where condition = true

- 100% statement coverage
- No error found in the code

Example

```
x = 0;
if (condition)
   x = x + 1;
y = 10/x;
```

Test 1 where condition = true

- 100% statement coverage
- No error found in the code

Test 2 where condition = false

- 75% statement coverage
- Error found in the code

Example

```
x = 0;
if (condition)
   x = x + 1;
y = 10/x;
```

Test 1 where condition = true

- 100% statement coverage
- No error found in the code

Test 2 where condition = false

- 75% statement coverage
- Error found in the code

Take-home 1: there is an insensitivity of statement coverage to control structures.

Take-home 2: 100% statement coverage does not mean there is no bug in the code.

Branch coverage

100% Branch coverage requires that each branch of a program is executed at least once during a test.

- Branch coverage checks boolean expressions as one predicate regardless of the presence of operators.
  - E.g., (condition A && condition B) is treated as one predicate
- Other statements such as switch statements and exception handlers are treated similarity.

Example

```
if (condition1)
   x = 0;
else
   x = 2;
if (condition2)
   y = 10*x;
else
   y = 10/x;
```

Test 1 where condition1 = true, and condition2 = true,

Test 2 where condition1 = false, condition2 = false,

- 100% branch coverage
- No error found in the code

Test 3 where condition1 = true,

and condition2 = false,

- 50% branch coverage
- Error found in the code

Take-home 1: 100% branch coverage does not mean there is not bug in the code.

Branch coverage = (Number of Decisions Outcomes tested / Total Number of Decision Outcomes) x 100 %

Path coverage

Path coverage requires that each of the possible paths through the program is followed during a test.

- The most reliable metric, however, not applicable to large programs.
  - Number of paths is exponential to the number of branches
- Suppose a test that takes 1 microsecond (10^{-6} sec) to execute
- Testing all paths of a program that contain 30 if-statement will take 18 minutes
  - ($2^{30} * 10^{-6}$ sec) / 60 = 18 min
- Testing all paths of a program that contain 60 if-statement will take 366 centuries
  - ($2^{60} * 10^{-6}$ sec) / 3,600= 3202,55974 hours = 36,559 years = 366 centuries

Example of state-of-the-art

For web/Java applications, Behat/JBehave can be used for structural testing to verify user stories.

1. Write story

Plain text

```
Scenario: A trader is alerted of status

Given a stock and a threshold of 15.0
When stock is traded at 5.0
Then the alert status should be OFF
When stock is traded at 16.0
Then the alert status should be ON
```

2. Map steps to Java

POJO

```
public class TraderSteps {
    private TradingService service;   // Injected
    private Stock stock; // Created

    @Given("a stock and a threshold of $threshold")
    public void aStock(double threshold) {
        stock = service.newStock("STK", threshold);
    }
    @When("the stock is traded at price $price")
    public void theStockIsTraded(double price) {
        stock.tradeAt(price);
    }
    @Then("the alert status is $status")
    public void theAlertStatusIs(String status) {
        assertThat(stock.getStatus().name(), equalTo(status));
    }
}
```

JBehave

- Is a behavior-driven development tool
- Lets you write tests in form of user stories.

Structural testing vs functional testing

Implementation vs specifications

- Structure testing evaluates code structure or internal implementation of the code
- Functional testing verifies whether the software functions in accordance with functional specifications

Code vs requirements

- Structure testing requires a understanding of the code
- Functional testing requires a understanding of the software’s requirements

Logic error vs system error

- Structure testing find errors in the internal code logic
- Functional testing ensures that the system is error-free

Schedule for today

- Key concepts from last class
- An alternative solution, fault removal
  - Functional testing
    - Example of state-of-the-art
  - Structural testing
    - Effectiveness
- System dependability
- Dependability concept summary

System dependability

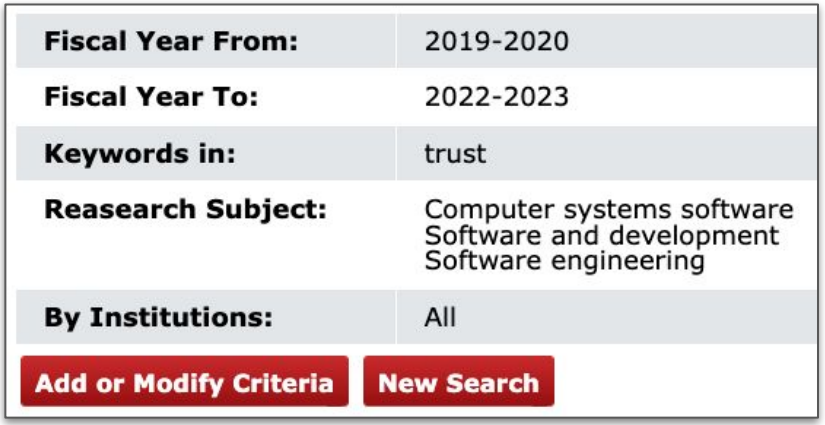
Dependability is the ability of a system to deliver its intended level of service to its users.

- Dependability reflects the user's degree of trust in the system
- In modern systems, dependability becomes important not only for traditional safety-, mission-, and business-critical applications, but also for everyday systems.

Why is it important?

- Undependable systems can cause information loss
  - Consequence: a high recovery cost.
- Systems that are not dependable may be rejected by their users / not trustworthy

Why trustworthiness matters



Fiscal Year From:	2019-2020
Fiscal Year To:	2022-2023
Keywords in:	trust
Reasearch Subject:	Computer systems software Software and development Software engineering
By Institutions:	All

Add or Modify Criteria | New Search

[NSERC's Fundings Database]

- Research subject: software
- Keywords in (project title/summary): trust
- Year 2019 to 2023
- 96 records of successful application

Showing 1 to 10 of 96 records

Show 10 records

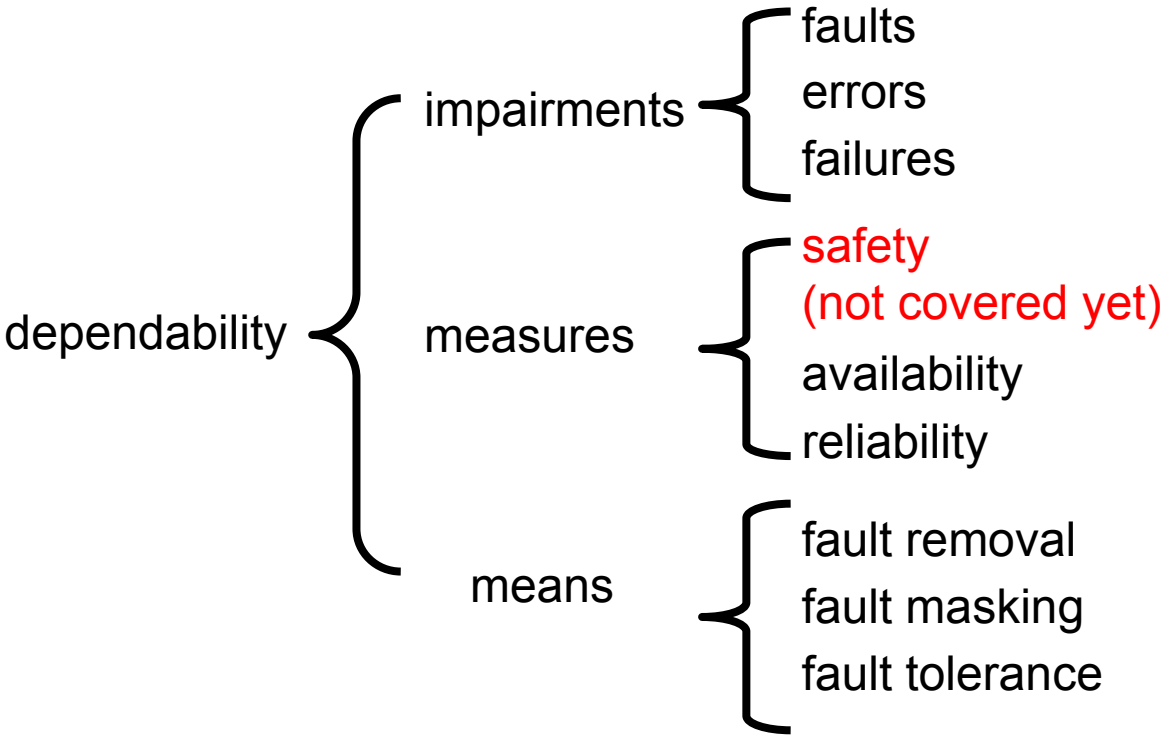
Name	Project Title	Amount($)	Fiscal Year	Program
		294,454	2021-2022	Collaborative Research and Training Experience
		294,454	2022-2023	Collaborative Research and Training Experience
		291,117	2020-2021	Collaborative Research and Training Experience

Dependability

Dependability consist of three characteristics:

- Attributes: properties which are required of a system.
  - Depending on the application design and behavior
    - E.g., for ATMs, availability is important
    - E.g., for heart pacemaker, reliability is important
- Impairments: reasons for a system to cease to perform its function
  - Threats to dependability
- Means: methods and techniques enabling the development of a dependable system
  - E.g., fault tolerance

Dependability concept summary



Dependability means:

- Fault tolerance is one of the methods for achieving dependability
- Fault tolerance can be used in combination with other methods (e.g., fault removal)

Questions

- Give an example of a code in which a bug will not be detected in spite of 100% statement coverage.
- Does the example above implies you have 100% branch coverage?
- Is the opposite always true? In other words, does 100% branch coverage implies 100% statement coverage?
- Give an example of a code in which a bug will not be detected in spite of 100% branch coverage.