Midterm Review

CSCE 247 - Lecture 13 - 03/04/2019

General Questions

- Today: Go over practice midterm questions.
- First - any general questions on course content or homework?

Topics

- 75-minute exam, about 9-10 questions.
- Covering all topics to date:
  - Overview/Principles
  - Requirements
  - Use Cases
  - Version Control and Issue Tracking
  - Testing Fundamentals
  - Requirements-Based Testing
  - Test Automation/JUnit
  - Build Systems
  - Design Fundamentals

Question 1

Briefly explain why a software system must change or become progressively less useful?

Question 1 - Solution

- The world is constantly changing, and if software does not change too, it will be out of date and useless.
- Changes might be
  - organizational (user's needs have changed).
  - infrastructure (hardware and OS are changing).
  - changed computational model (standalone systems are now networked)
  - … etc...

Question 2

The following requirements are unclear and ambiguous. Explain why, and then rewrite the statements so that they can be objectively evaluated.

a. The response time should be minimized.

b. The alarm should be raised quickly after a high fuel level has been detected.

Question 2 - Solution

a. The response time should be minimized.

“should” != shall

What does minimized mean?

Response time to what?

“The system shall respond to a user request within ten seconds.”

Question 2 - Solution

b. The alarm should be raised quickly after a high fuel level has been detected.

Quickly?

Is “high fuel level” a boolean condition or a specific quantity?

“The alarm shall be raised within 5 seconds of the fuel level reaching 10 cm.”

Question 3

What are the properties of a “good” individual requirement (as opposed to properties of a requirements document)?

Mention four (4) properties of a “good” requirement and explain what they mean.

For each, give an example of a requirement that violates the property.

Question 3 Solution

- Unambiguous
  - There is only one way of interpreting the requirement.
- Correct
  - It captures the stakeholders true needs.
- Testable
  - It must be written so that it can be objectively used as an oracle during testing process.
- Traceable
  - We must be able to trace the requirement to its definition, environmental assumptions, business case, use-case, etc.

Question 3 Solution

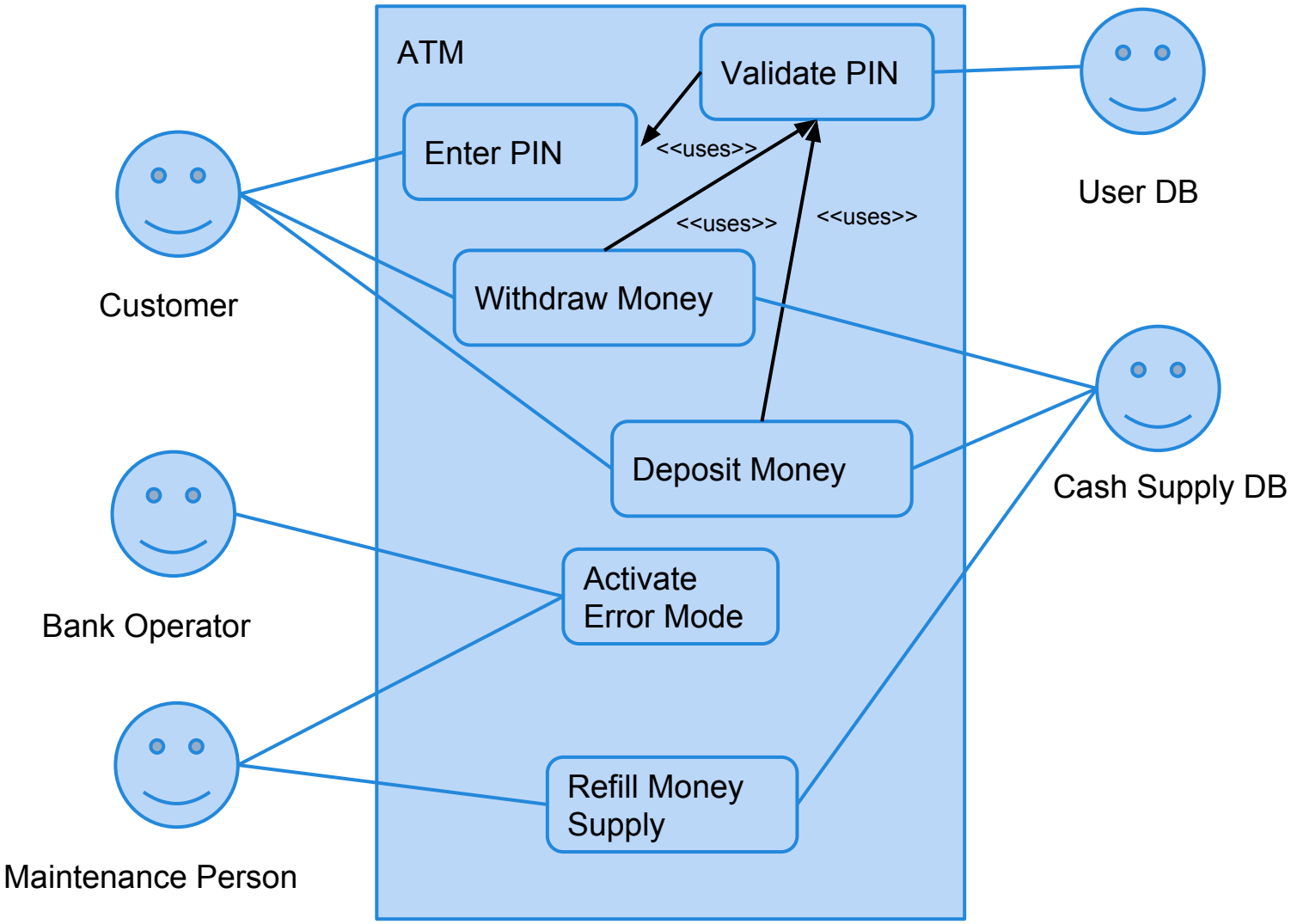
- Feasible
  - We must be able to actually implement this requirement.
- Prioritized
  - We must prioritize the requirements so that we can decide wisely which ones to do first when we (inevitably) run out of time.
- Complete
  - Make sure we have covered all aspects of this particular requirement.
- Consistent
  - A requirement cannot be self-contradictory.
- Precise, unambiguous, and clear

Questions 4 & 5

ATM

- What are the actors and use-cases involved in an ATM system?
- Draw a use-case diagram
- Pick one use case and write a scenario.

Question 4 Solution



Question 5 Solution

Validate PIN

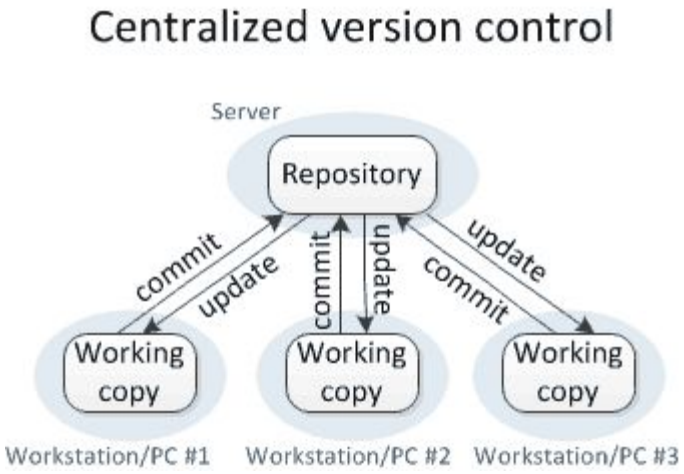
- **Basic Course of Events:**
  - The customer inserts the card in the ATM.
  - The ATM reads the card.
  - The ATM asks the customer for the PIN.
  - The ATM validates the PIN for the card.
- **Alternative Paths:** In step 2, if the card cannot be read, the card is returned to the customer and the ATM is ready to accept another card.
- **Exception Paths:** In step 4, if the PIN is incorrect, the ATM will ask the customer again. The customer will get three (3) attempts after which the ATM will provide an error message and return the card.
- **Extension Points:** None
- **Assumptions:** The customer has an account and ATM card.
- **Precondition:** The card central repository is available for card validation.
- **Postcondition:** The card has been accepted for further transactions and the customer can proceed.

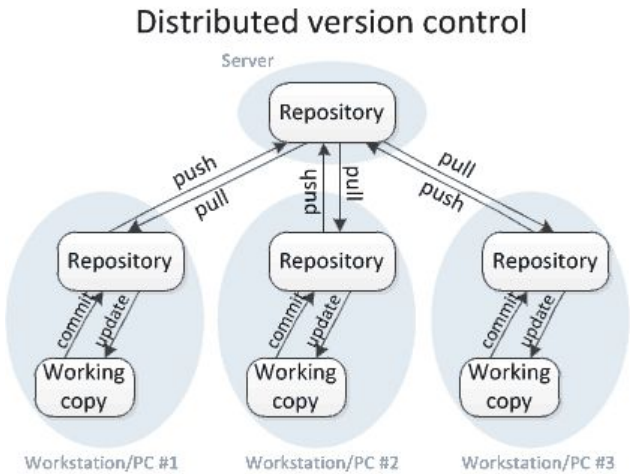
Question 6

Briefly explain the key differences between centralized and distributed version control.

What are the strengths and weaknesses of each approach?

Question 6 Solution





- In centralized VCS, the repository is stored on a central server.
- In decentralized VCS, all working copies are paired with a copy of the repository.
  - A central repository is used to coordinate and synchronize the local repositories.

Question 6 Solution

- Decentralized VCS has become standard.
- Centralized CVS has a single point of failure.
  - If the central server is lost, we could lose data.
  - If it is unavailable, developers cannot push changes, merge updates, or roll back code.
- Decentralized avoids this by keeping a copy of the repository in each working copy.
  - If the central server is lost, any copy could become the “central” source of truth.
- Decentralized has more complex workflow.
  - Additional pull step before update, additional push after commit.

Question 7

- Explain the difference between *verification* and *validation*.
- Which of these is considered harder? Why?

Question 7 - Solution

- Explain the difference between *verification* and *validation*.
  - Validation: Does the system meet the customer's needs? "Are we building the right product?"
  - Verification: Does the system meet the specifications we laid out? "Are we building the product right?"
- Which of these is considered harder? Why?
  - Validation is harder.
  - It requires that we understand the customer's actual desires. They might not have told us those, or changed their minds.

Question 8

The airport connection check is part of a travel reservation system. It checks the validity of a single connection between two flights in an itinerary.

```
validConnection(Flight arrivingFlight, Flight departingFlight) returns ValidityCode.
```

A Flight is a data structure consisting of:

- A unique identifying flight code (string, three characters followed by four numbers).
- The originating airport code (three character string).
- The scheduled departure time (in universal time).
- The destination airport code (three character string).
- The scheduled arrival time (in universal time).

There is also a flight database, where each record contains:

- Three-letter airport code (three character string).
- Airport country (two character string).
- Minimum connection time (integer, minimum number of minutes that must be allowed for flight connections).

Question 8

A Flight is a data structure consisting of:

- A unique identifying flight code (string, three characters followed by four numbers).
- The originating airport code (three character string).
- The scheduled departure time (in universal time).
- The destination airport code (three character string).
- The scheduled arrival time (in universal time).

There is also a flight database, where each record contains:

- Three-letter airport code (three character string).
- Airport country (two character string).
- Minimum connection time (integer, minimum number of minutes that must be allowed for flight connections).

Derive representative values of the parameters from this specification for the validConnection function.

Question 8 - Solution

Parameter: Arriving flight

Flight code:
- *malformed*
- *not in database*
- *valid*

Originating airport code:
- *malformed*
- *not in database*
- *valid city*

Scheduled departure time:
- *syntactically malformed*
- *out of legal range*
- *legal*

Destination airport (transfer airport):
- *malformed*
- *not in database*
- *valid city*

Scheduled arrival time (tA):
- *syntactically malformed*
- *out of legal range*
- *legal*

Parameter: Departing flight

Flight code:
- *malformed*
- *not in database*
- *valid*

Originating airport code:
- *malformed*
- *not in database*
- *differs from transfer airport*
- *same as transfer airport*

Scheduled departure time:
- *syntactically malformed*
- *out of legal range*
- *before arriving flight time (tA)*
- *between tA and tA + minimum connection time (CT)*
- *equal to tA + CT*
- *greater than tA + CT*

Destination airport code:
- *malformed*
- *not in database*
- *valid city*

Scheduled arrival time:
- *syntactically malformed*
- *out of legal range*
- *legal*

Parameter: Database record

This parameter refers to the database time record corresponding to the transfer airport.

Airport code:
- *malformed*
- *not found in database*
- *valid*

Airport country:
- *malformed*
- *not a real country*
- *valid*

Minimum connection time:
- *not found in database*
- *Negative*
- *0*
- *valid*

Question 9

You are writing test cases for the following method:

```
public double max(double a, double b);
```

This method returns the largest of two integers.

Using the JUnit notation, write three test cases for this method.

Question 9 Solution

```
@Test
  public void aLarger() {
    double a = 16.0;
    double b = 10.0;
    double expected = 16.0;
    double actual = max(a,b);
    assertTrue("should be larger", actual>b);
    assertEquals(expected, actual);
  }
@Test
  public void bLarger() {
    double a = 10.0;
    double b = 16.0;
    double expected = 16.0;
    double actual = max(a,b);
    assertTrue("b should be larger", b>a);
    assertEquals(expected, actual);
  }
```

```
@Test
  public void bothEqual() {
    double a = 16.0;
    double b = 16.0;
    double expected = 16.0;
    double actual = max(a,b);
    assertEquals(a,b);
    assertEquals(expected, actual);
  }
@Test
  public void bothNegative() {
    double a = -2.0;
    double b = -1.0;
    double expected = -1.0;
    double actual = max(a,b);
    assertTrue("should be negative",actual<0);
    assertEquals(expected, actual);
  }
```

Question 10

When we discuss software testing, we refer to Faults and Failures.

Please briefly describe what a Fault is and what a Failure is.

Make sure to point out the difference between a Fault and a Failure.

Question 10 Solution

- A Fault is a problem with the implementation.
  - It is something that is missing, extra, or erroneous.
- A Failure is an incorrect execution of the software.
  - We get an output we did not expect.
- A Failure is the manifestation of a Fault.
  - If the execution executes the Fault and the corrupted state propagates to the output, we can observe it as a Failure.

Question 11

In class, we discussed the importance of defining a test case for each requirement. What are the two primary benefits of defining this test case?

Question 11 - Solution

1. A test case will greatly help us in the integration testing phase. Groups can start defining tests early and be ready when the system comes online.
2. Test cases force us to write testable (thus, good) requirements. If a requirement is not testable, we cannot write a test case.

Question 12

Briefly discuss the concept of incrementality (from now on, this is a real word) as it applies to software development.

Question 12 - Solution

Incrementality is the principle of breaking the software project (or anything else) into smaller manageable pieces that can be used by the customer while other pieces are still in development.

As new pieces are completed, they are integrated until we have a complete system.

Question 13

The main function of a vending machine is to allow the customer to buy products from the machine (soda, candy, etc). When the customer wants to buy some of the products, they insert money, select one or more products, and the machine dispenses the product to the customer. Should the product cost less than the amount of money inserted, the machine will dispense change. The machine must be restocked when it runs out of products. A collector comes and collects money from the vending machine.

1: Identify the actors and use cases.

2: Define the basic course of events for one use case.

3: What are the exception or alternate paths for that scenario?

Question 13 - Solution

Actor: **Customer**

Use Cases:

- Buy Product

What about these:

- Select Product
- Dispense Product
- Dispense Change

Question 13 - Solution

Actor: **Stocker**

Use Cases:

- Restock Machine

Actor: **Collector**

Use Cases:

- Collect Money

Can these two actors be the same person?

Question 14

In the class, we discussed non-functional requirements.

Explain the concept of non-functional requirements and give two examples.

Question 14 - Solution

Explain the concept of non-functional requirements and give two examples.

Requirements that do not impact the correctness of the functional behavior (the services the system performs).

Usually related to security, performance, reliability, maintainability, etc.

Any other questions?

Next Class:

- The Midterm

After Spring Break

- Software Architecture
  - (high level design)