CSCE 740 - Practice Midterm
Name:

This is a 75-minute exam. On all essay type questions, you will receive points based on the quality of the answer - not the quantity.

Make an effort to write legibly. Illegible answers will not be graded and awarded 0 points.

There are a total of 11 questions and 100 points available on the test.

Question 1—7 Points.

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

Suggested Solution:

The world is constantly changing, if the software does not change with the world, it will eventually be completely out of date and useless. The changes in the world might be organizational (the users' needs have changed and the software must adapt), infrastructure (the hardware and operating systems are changing and the software must be adapted), the computational model changes (formerly stand alone systems must now be networked), etc. etc.

Question 2—7 Points.

The properties of the environment of a system are generally of critical importance for the system under development to be able to satisfy its stated system requirements. Therefore, it is essential to capture environmental assumptions in a requirements document. Briefly discuss how the environment may influence a system's ability to satisfy its requirements.

Suggested Solution:

A system can typically not meet the requirements without some help (or at least collaboration) from the environment. For example, consider the example of a patient monitoring system discussed in the world-machine model lecture. In this system, the device cannot meet its requirements unless the nurses are close enough to hear the alarm. Similar situations occur in most systems, the environment must cooperate to some degree with the system to get the job done. The reason to capture environmental assumptions is to document the degree of which we expect the environment to cooperate with us—our system assumes certain cooperation from the environment. If this cooperation is not present, our system may not satisfy the stated requirements.

In other words:

1. *A piece of software is always intended to work correctly in a given environment. Without a specification of what that environment is, it is impossible to determine if the software works correctly, it has to be evaluated in its intended environment.*
2. *The software cannot be designed to handle all conceivable situations in all environments, we must limit the scope of the software. We do this by making assumptions about the environment, for example, assumptions about situations that cannot occur or how the environment will respond to commands. If the assumptions do not hold, the software may not work as intended.*
3. *The software may not be able to achieve all system goals by itself. We may have to constrain the environment to meet the goals. For example, we may require that the software is used on aircraft above 1500 ft only. This must be captured in the requirements.*

Question 3—8 Points.

Pick two of the following key tenets of extreme programming and briefly explain why each is important:

1. Collective Ownership
2. Sustainable Pace
3. Open Workspaces
4. Customer as a Team Member
5. Test First Design
6. Short Iterative Cycles
7. Stories as Requirements

Suggested Solution:

1. *All developers own all the code. If a change to someone else's code needs to be done, all developers are empowered to do it. We do not need any approval or signatures to slow us down, simply redesign, make sure you do not break old functionality, and go for it.*
2. *Developers produce worse results if forced to work too much overtime. Keep iteration scope in check.*
3. *Since we are operating with very little documentation and there are constant changes, free, open, and quick communication is essential. An open workspace helps here.*
4. *Since there are no well-defined requirements, rapid feedback on iterations and quick response to questions is absolutely essential. Having a customer as part of the team provides this rapid response.*
5. *Developing tests is a way of capturing some required scenarios. This helps provide light weight requirements and forces us to clarify thinking before we commence the tricky task of coding things up.*
6. *We are able to quickly get feedback on the current status of a project.*
7. *We can use the expected usage of the system as the basis for development. Lightweight requirements document.*

Question 4—12 (7+5) Points.

You are involved in a development of a new software product. The product is an insurance application intended to determine what insurance products a potential customer is eligible for. The eligibility requirements are captured in various laws and regulations. An external contractor is doing most of the work. Your organization has hired this contractor to assist with all aspects of planning, management, and development since your organization is lacking the expertise to complete the project on its own. The plan was to do a traditional waterfall process. The product will be long lived and good documentation is a must. In addition, the laws and regulations were thought to constitute a good start for the requirements of the project—thus, a waterfall process was deemed the best choice.

During the requirements capture process, the team discovers that the laws and regulations are incomplete, ambiguous, and generally obtuse—the capture is taking way longer than planned and required a lot of interaction with case workers that possess the knowledge of how these laws and regulations are applied in practice. Towards the end of the requirements process, the requirements document is incomplete and there is much more requirements work to be done.

At this point, the contractor decides that since there is so much requirements risk and we are far behind schedule, the project will switch on an agile method—eXtreme Programming—in an attempt to recover. In addition, to save money, the contractor is offshoring the coding efforts to a development center in a low-cost country.

Part 1:
Is this approach likely to succeed? Briefly justify your answer.

Part 2:
What would your recommendation have been? Briefly explain why you provide that recommendation.

Suggested Solution:
Part 1: I do not think this is likely to succeed. Agile techniques are good at hedging against requirements risk, but they do require easy access to the customer (or domain expert/the one that knows). Moving the development off-shore seems like it would make this problem quite severe. In addition, the cultural differences and unfamiliarity with the U.S. insurance business may be a hindrance.

Part 2: I would have recommended re-assessing the development effort to determine if this project is viable at all. I would assess the existing requirements to see if we can start an iteration building the system with the known pieces in place while the remaining requirements are elicited and captured. If done well, the new rules (as they are discovered or modified) would be easily integrated into the system. Any sensible answer will be accepted.

Question 5—10 (6+4) Points.

Part 1:
Explain the difference between *validation* and *verification*.

Part 2:
Validation is generally considered harder. Why?

Suggested Solution:

Part 1:
Validation is concerned with building the product the customer wants. It attempts to answer the question "Are we building the right product?"

Verification is concerned with if we are building a product with the properties we said it was going to have (does the product match the requirements specification?). It is trying to answer, "Are we building the product right?"

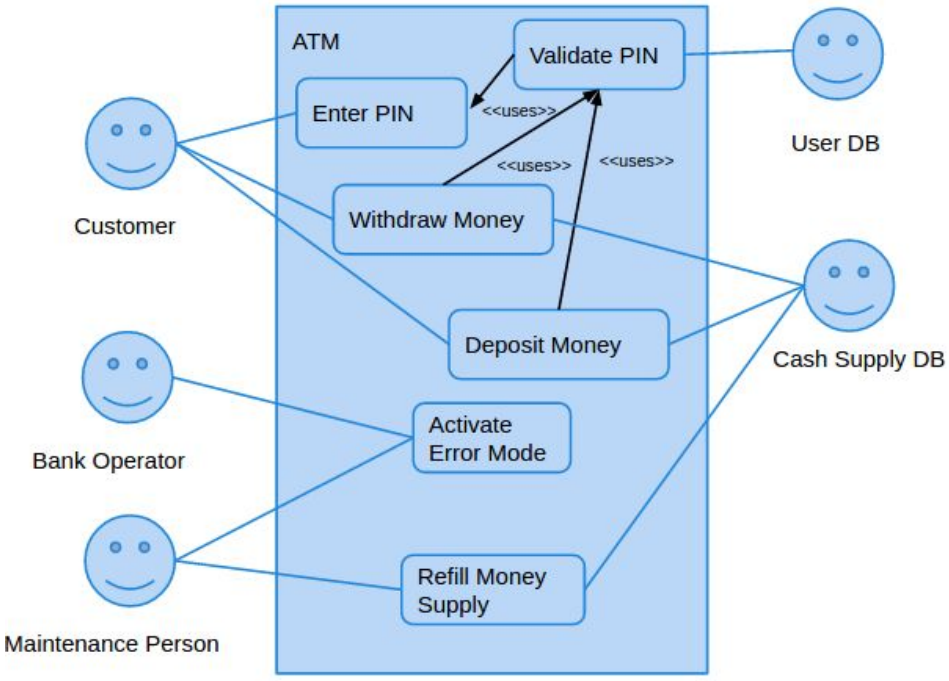
Part 2:

Validation is harder because it is subjective - up to the whims of the customers. You can "measure" verification, build evidence that the product works under the conditions that you have set. In verification, you have the specification to compare the product to. In validation, you have to make the customers happy. Customers change their minds, they leave out details, and - if there are multiple stakeholders - they may have conflicting desires. This makes validation harder.

Question 6—9 Points.

You are all familiar with an Automatic Teller Machine (cash machine). Please draw a use-case diagram with at least three (3) use-cases and the actors involved in these use-cases.

Suggested Solution:
Something along the lines of the following will be accepted:



Question 7—8 Points.

Pick one of the use-cases from the previous question and write down the typical scenario for this use-case.

Suggested Solution:
Any reasonable scenario will be accepted (see examples from the use-case lecture on the ATM)

Question 8—15 Points.

The airport connection check is part of a travel reservation system. It is intended to check 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).

ValidityCode is an integer with value 0 for OK, 1 for invalid airport code, 2 for a connection that is too short, 3 for flights that do not connect (arrivingFlight does not land in the same location as departingFlight), or 4 for any other errors (malformed input or any other unexpected errors).

In order to design requirements-based test cases, perform input partitioning and derive representative values of the parameters from this specification for the validConnection function.

Suggested Solution:
The following are the essential input partitions. Others are possible.

Recall the lectures on requirements-based testing. The approximate process of taking a requirement and writing tests is to:

1. *Refine the requirement so that it is testable.*
2. *As a requirement is just a property of the software, you usually can't directly "test the requirement." Instead, you need to use a function of the software and show that the requirement holds during that execution. So, the next step is to identify the independently testable features of the software.*
3. *Now that you have those, you need to look at the parameters and figure out which inputs to pass in. You cannot exhaustively test a function, there are too many possible parameters.* ***So, instead, you partition the input domain into representative regions.*** *If you try inputs from each of these areas, you are more likely to trigger a fault than through random testing alone. We discussed some methods of doing so during Lecture 8.*
4. *Once you have the inputs partitioned, you can form abstract test cases for which you can transform into actual test cases by coming up with concrete input values from the identified partitions.*

In this question, you have been given a testable feature, so you have been asked to perform the activity from Step #3 above - given the parameters, partition the inputs into representative regions. This function has two explicit inputs - an arriving flight and a departing flight - and an implicit input - an airport connection database.

A flight is a complex data structure containing several fields, so for each field, you need to partition the inputs. Remember that the function's parameters may influence each other (testing this function requires considering both the arriving and departing flight's field values as well as what is in the database), so that may influence the partitions.

<u>Parameter: Arriving flight</u>

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 - where connection takes place):
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
invalid
valid

Minimum connection time:
not found in database
invalid
valid

Question 9—8 (4+4) Points.

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.

Suggested Solution:

a: "should" != shall. What does minimized mean? Response time to what?

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

b. 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 10—8 Points.

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?

Suggested Solution:

1. *A test case will greatly help us in the integration testing phase. Now our testing groups can start defining test cases and procedures early and be ready when the system is coming on-line.*

2. *Test cases force us to write testable (thus, pretty good) requirements. If a requirement is not testable, we simply cannot write a test case.*

Question 11—8 Points.

You are setting out to develop a new GUI for an old application. The system has a diverse set of users and the system has to be acceptable to all of the user types.

What development process would you use? Justify your answer.

Suggested Solution:

Any process that makes use of evolutionary prototyping would work. Build something rapidly and get use feedback, then build something new that incorporates that feedback. Look for a reasonable argument.