TECHNISCHE UNIVERSITEIT DELFT
Faculteit Elektrotechniek, Wiskunde en Informatica

TU Delft

Exam IN3205-I and TI2210
Software Quality and Testing

18 April 2012, 14:00-17:00, Rooms LR-{E,F,G,H,J}

- There are 5 questions worth of 19 points in total.
- Total number of pages: 3.
- Use of books and readers is not allowed
- You can answer in English or in Dutch
- Please list your answers in the right order
- Write clearly and avoid verbose explanations: Points may be deducted for unclear or sloppy answers
- The tentative grading scheme is:

Question:	1	2	3	4	5	Total
Points:	4	2	4	5	4	19
Score:						

- If p is the number of points you score, the exam grade E will most likely be determined by

$$E = 1 + 9 * p/19$$

- Your final grade F is determined based on your results for the labwork L and exam E:

$$F = (L + 3 * E)/4$$

Note that you can only pass if both $E \geq 6.0$ and $L \geq 6.0$.

GOOD LUCK!

```
public class PictureHandler {
        private PictureReader myReader;
        private HandlerListener myListener;
        ...
        /**
         * Try to read a given picture, retrying it at most n times.
         * Inform the listener if the picture was read successfully.
         * @param name Name of the picture to be read.
         * @param n Number of read attempts to be made.
         */
        public void readRepeatedly(String name, int n) {
                int i = 0;
                Picture pict = null;

                while (i <= n && pict == null) {
                        i++;
                        pict = myReader.readPicture(name);
                }

                if (pict != null) {
                        myListener.pictureSuccessfullyRead();
                }
        }
        ...
}
```

Figure 1: A Picture Manipulation Method

1. You are responsible for testing the proper behavior of the method `readRepeatedly` as shown in Figure 1.
   (a) (1 point) Draw a control flow diagram for the method under test.
   (b) (1 point) What is the minimum number of test cases needed to achieve *statement coverage*? Explain your answer.
   (c) (1 point) What is the minimum number of test cases needed to achieve *branch coverage*? Explain your answer.
   (d) (1 point) What is the minimum number of test cases to achieve *loop boundary adequacy*?
2. To test the code from Figure 1, one can use *mock objects*.
   (a) (1 point) Are there classes you would like to mock in order to increase the *controllability* of the method under test? Explain why, and, if so, indicate which ones, and provide the mock code in Mockito-style.
   (b) (1 point) Are there classes you would like to mock in order to increase the *observability* of the method under test? Explain why, and if so, indicate which ones, and provide the mock code in Mockito-style.

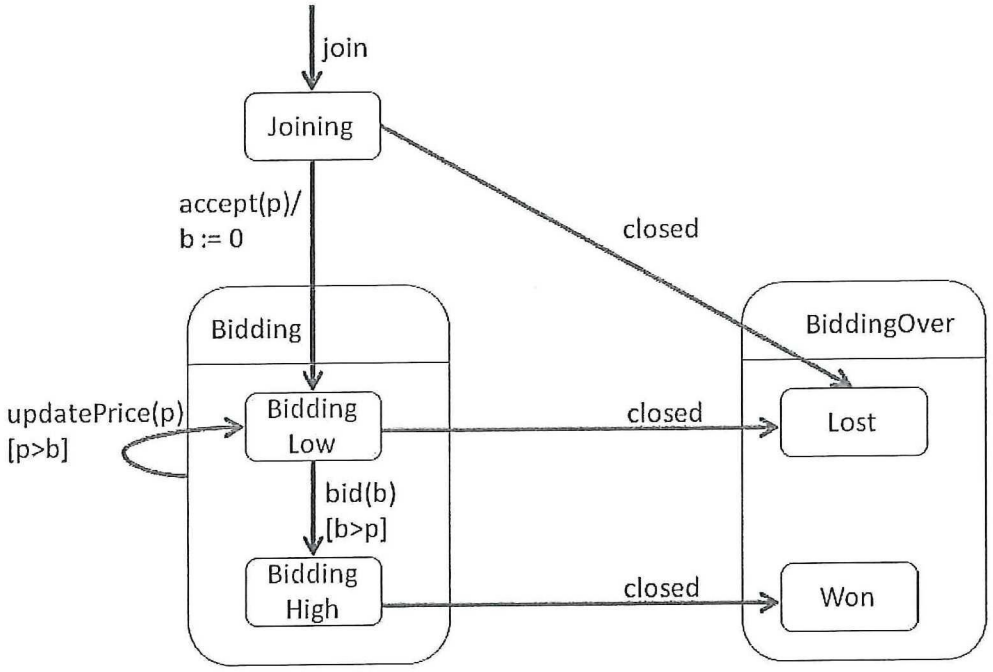

Figure 2: State machine for bidder in an auction

3. You are involved in a software system for an auction house. The design documents include the figure shown in Figure 2, displaying a UML state diagram for a bidder involved in an auction[1]

   A bidder can join an auction, after which it can start bidding. After receiving a price p, it can issue a bid b, provided $b > p$. During bidding, the bidder can be notified of price updates (caused by other bidders), leading to a new price p which is larger than the bidder's last bid b. When the auction closes the moment the bidder holds the highest bid, this bidder wins the auction of this item.

   Your job is to test the implementation of the bidder.

   (a) (1 point) Draw a flattened diagram of the state machine.

   (b) (1 point) Turn the state machine into a transition tree, and derive a test suite achieving *all-roundtrip path* coverage. How many test cases does it contain?

   (c) (1 point) Turn the state machine into a state transition table, in order to derive a "sneak path" test suite. How many test cases does it contain? What is a sensible default action for (event, state) pairs for which no action is defined?

   (d) (1 point) You next consider adopting the *boundary interior loop coverage* criterion for your test suite. To how many additional test cases does this lead? Explain your answer.

[1] Based on a state machine from Freeman & Ryce, *Growing Object-Oriented Software, Guided by Tests*, Addison-Wesley, 2010.

4. You're involved in developing a web application for a law firm specializing in family law. The firm wants to provide a web application giving men advice whether they can become *legal father* of a child of a particular mother. The firm provides you with three situations in which a candidate father can indeed become a legal father:

   I. If the candidate father was married to the mother at the time of birth.

   II. If the mother agrees that the candidate father becomes legal father, provided there is no other legal father already.

   III. If the candidate father is the biological father, provided he is not merely a sperm donor, and provided there is no other legal father already. In this case legal fatherhood is not given automatically, but only by a judge who will verify that recognition by the candidate father does not negatively affect the child's development.

   It is your responisbility to implement this description into the web site, and to test it thoroughly.

   (a) (1 point) Turn the checklist from the law firm into a decision table. Ensure that there is a row for each condition you can identify, and a column for each of the three situations described by the firm.

   (b) (1 point) How many test cases are needed at least to achieve *basic condition adequacy* for a test suite derived from this decision table? Explain your answer.

   (c) (1 point) How many test cases are needed at least to achieve *compound condition adequacy*? Explain your answer.

   (d) (2 points) How many test cases are needed at least to achieve *modified condition/decision coverage (MC/DC)*? Explain your answer, and provide the required test cases.

5. The e4 platform builds upon OSGi and provides a plugin mechanism also found in Eclipse. Each plugin is an OSGi bundle, and also provides so-called *extension points*. This often entails defining a regular Java *interface* that others can implement in order to extend the behavior of the plugin.

   Suppose you are developing a plugin *InteractionAnalysis* for collecting file manipulation statistics. You define one extension point, *Persistence*, consisting of a regular Java interface including a read and a write method for the data collected.

   (a) (1 point) What can you as developer of the *InteractionAnalysis* plugin do to test the *Persistence* extension point?

   (b) (1 point) You want to encourage external developers who may implement the *Persistence* extension point to test their implementation well. What can you do to make it as easy as possible for them to test their extension?

   (c) (1 point) Your colleague provides an implementation of the *Persistence* extension point which requires that a particular database already exists. Is this consistent with the principles of design by contract? Explain why.

   (d) (1 point) Is there a way to design the *Persistence* interface differently, in such a way that the needs of your colleague can be catered for?

End of exam