TECHNISCHE UNIVERSITEIT DELFT
Faculteit Elektrotechniek, Wiskunde en Informatica

TUDelft

Exam TI2210
Software Quality and Testing

26 June 2013, 14:00-17:00, Room CT-1.98

- There are 5 questions worth of 23 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 with question numbers clearly indicated.
- Write clearly and avoid verbose explanations: Points may be deducted for unclear or sloppy answers.
- If you want your graded to be filed under code IN3205 please indicate so clearly on your exam.
- The tentative grading scheme is:

Question:	1	2	3	4	5	Total
Points:	7	3	4	5	4	23
Score:						

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

$$E = 1 + 9 * p/23$$

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

$$F = (L + E)/2$$

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

GOOD LUCK!

1. Current versions of the Linux kernel provide around 10,000 configuration options (specified in `kconfig` files), which are technically implemented and enforced in the code by means of preprocessor macros. In this exercise we will explore testing Linux kernels with different configurations.

   To simplify our reasoning, in this exercise, we will make the following assumptions:

   - Each configuration option can have 3 distinct values,
   - It is possible to combine any option with any other.

   Before any such configuration can be tested, it must be compiled first. In this exercise we study how many configurations should be created.

   Answer the following questions:

   (a) (1 point) How many different kernels can be configured?

   (b) (1 point) Provide a concise definition of the *pairwise* testing strategy for a system depending on N different parameters.

   (c) (1 point) Explain how *pairwise* testing can be applied to the problem of selecting which Linux kernel configurations to compile and test, and how pairwise *coverage* can be used to build up a test suite.

   (d) (2 points) Even with pairwise testing, there are too many kernel configurations possible to achieve full pairwise coverage. A more likely scenario is to fix the number of test cases to, say, 10, and then use t-way testing to maximize variation among those kernels for a value of $t \geq 2$.

   Assume we are willing to compile and test 10 different kernel configurations, and that you use t-way testing to derive those.

   1. Assume $t = 2$. How many pairs of configuration options can you test? (Hint: Remember from your Probability courses that there are $\binom{n}{t} = \frac{n!}{t!(n-t)!}$ ways to choose a combination of t variables from a set of n variables)
   2. Now assume $t = 3$. How many 3-way combinations of configurations can you test?
   3. Finally assume $t = 10,000$. How many full configurations can you test?

   (e) (1 point) Now assume that you have information about actual installations in the field, i.e., you know what the most common configurations are. How would this affect the 10 different kernel configurations you would test?

   (f) (1 point) Your colleague proposes to use the *category-partition strategy* instead of *pairwise testing*

   1. Briefly describe the main steps of the *category-partition strategy*.
   2. Your colleague decides to give every configuration option the `[single]` attribute. What is the smallest number of kernels to be compiled given this assessment?

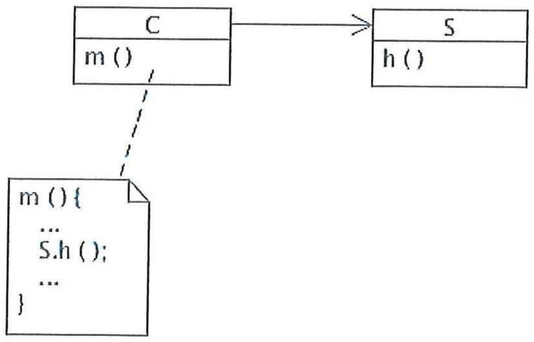

Figure 1: Class diagram for question 2

2. You are writing a test for a method *m* in a client class *C*, which uses library method *h* from a class *S* provided by an external supplier. A class diagram illustrating this is shown in Figure 1.

   You run the test suite with assertion checking enabled. Your test case fails since *h* raises one assertion failure.

   You seek to identify the cause of the problem. We assume that the assertions themselves are correct, but that the actual implementation (of either the client *C* or the library *S*) may be incorrect.

   (a) (1 point) Assume it is the *precondition* of *h* that is failing. Which of the following statements is true? Explain your answer.

   1. This most likely requires a fix in your client *C*;
   2. This most likely requires a fix in the library class *S*;
   3. This most likely requires a fix in *both* your client *C* and the external library class *S*;
   4. None of the above.

   (b) (1 point) Now assume the precondition of *h* passes, but that its *postcondition* is failing. Which of the following statements is true? Explain your answer.

   1. This most likely requires a fix in your client *C*;
   2. This most likely requires a fix in the library class *S*;
   3. This most likely requires a fix in *both* your client *C* and the external library class *S*;
   4. None of the above.

   (c) (1 point) Finally, assume that the only reported assertion failure indicates that only the *invariant* of library class *S* is failing when your test invokes *h*. Which of the following statements is true? Explain your answer.

   1. This most likely requires a fix in your client *C*;
   2. This most likely requires a fix in the library class *S*;
   3. This most likely requires a fix in *both* your client *C* and the external library class *S*;
   4. None of the above.

3. Next, you explore the possibility of using a specialized subclass S' of library class S' to remedy your problem of the previous question.
   (a) (1 point) Provide a concise definition of the Liskov Substitution Principle.
   (b) (2 points) Assume method h has precondition P and postcondition Q, and that class S has invariant I. The invariant, pre- and postconditions of S and h' can be composed from I, P and Q combined with additional conditions I', P' and Q'.
   Which of the following leads to a subclass that conforms to the Liskov Substitution Principle? Explain your answer.
      1. The precondition of h' is: `assert` $P \wedge P'$
      2. The precondition of h' is: `assert` $P \vee P'$
      3. The postcondition of h' is: `assert` $Q \wedge Q'$
      4. The postcondition of h' is: `assert` $Q \vee Q'$
      5. The invariant of S' is: `assert` $I \wedge I'$
      6. The invariant of S' is: `assert` $I \vee I'$
   (c) (1 point) In light of your previous answer, which assertion failures for your test case of $C.m()$ can be resolved by switching to S'? Explain your answer.

4. You are 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 responsibility 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. For a pension fund you need to test an application in which the requirements state that premium has to be paid as long as
   - $18 < age \leq 65$, and
   - $years\text{-}worked < 40$.

   (a) (2 points) Create a domain matrix that can be used for testing when premium has to be paid.
   (b) (2 points) Your colleague argues that instead of domain testing with the one-by-one strategy, it is easier and better to adopt a strategy based on branch coverage of the system-under-test. Do you agree? Why (not)? Provide a concise and to-the-point answer highlighting exactly why (not).

End of exam