CompSci 230
Software Design and Construction

Software Quality 2015S1
Myers' testing principles

Lecture plan

Week 1:	*No class - Anzac Day* What is software quality? Some key developer practices (version control, testing).
Week 2:	Black box testing. White-box testing. Myers' testing principles.
Week 3:	Traditional approach to testing (Waterfall). Agile approach to testing (XP). Famous failures.

Myers Ch. 2, pp. 12-18

Learning Goals for Today

- Have a working understanding of Myers' principled approach to software testing.
  - Given some information about a testing situation, can you apply Myers' principles? ("What would Myers do in this situation?")
  - Under what conditions are Myers' principles inapplicable or inappropriate?
    - I'm asking you to write some "test cases" for Myers' principles – considering its "unexpected inputs" as well as the inputs he explicitly considered!
    - Will we discover any situations where his set of principles does not do "what it is designed to do"?
    - Will we discover any situations where his principles will "do something unintended"?
- Start to develop your own "principled approach" to software testing.
  - Do you agree with all of Myers' principles? Do you have any additional ones?
  - Do you understand Myers' argument for each of his principles?

Principle 1 (review)

- "A necessary part of a test case is a definition of the expected output or result."

Rationale:

- "If the expected result of a test case has not been predefined,
  - "chances are that a plausible, but erroneous, result will be interpreted as a correct result
  - "because of the phenomenon of 'the eye seeing what it wants to see'."

Prescription:

- "A test case must consist of two components:

1. "A description of the input data to the program.
2. "A precise description of the correct output of the program for that set of input data."

Principle 2

- "A programmer should avoid attempting to test his or her own program."

Rationale:

1. By analogy: "Any writer knows – or should know – that it's a bad idea to attempt to edit or proofread his own work."
   - Don't good writers revise their work a few times **before** showing it to anyone else?
   - "As your studies progress it is important to become more independent with editing and proofreading your own work. The following points may be helpful in guiding you through this process." (http://www.monash.edu.au/lls/llonline/quickrefs/20-editing-proofreading.xml)
   - "Proofread more than once. If possible, work with someone else." (http://www.ucc.vt.edu/stdysk/proofing.html)
2. From Myers' psychological theory:
   - "… most programmers … cannot bring themselves to shift mental gears to attempt to expose errors."
3. Because the specification may be misinterpreted:
   - "The program may contain errors due to the progtammer's misunderstanding of the problem statement or specification… it is likely that the programmer will carry the same misunderstanding into tests of his or her own program."

Principle 2

- "A programmer should avoid attempting to test his or her own program."

Rationales:

1. By analogy with the (final) editing/proofreading stage of writing, where we distinguish the writer's task from the editor's task;
2. From Myers' psychological theory of "constructive" programmers and "destructive" testers; and
3. To adequately test errors arising from a misinterpretation of the spec.

Prescription:

- "[T]esting is more effective and successful if someone else does it."
- Caveat: "Debugging is more efficiently performed by the original programmer."

My question:

- Should the programmer develop the initial set of white-box tests for their code?

Principle 3

- "A programming organization should not test its own programs."

Rationale:

- By extension of the "psychology" rationale for Principle 2:
  - "A project or programming organization is, in many senses, a living organization with psychological problems similar to those of individual programmers."
  - "The testing process, if approached with the proper definition, may be viewed as decreasing the probability of meeting the schedule and the cost objectives."

Prescription:

  - "This does not mean that it is *impossible* for a programming organization to find some of its errors… organizations do accomplish this with some degree of success."
  - "Rather, it implies that it is more economical for testing to be performed by an objective, independent party."

My question:

  - Wouldn't a well-designed and properly-administered internal testing process be likely to reveal schedule-feasibility problems earlier, allowing these problems to be addressed at lower total cost, than if no testing were done?

Principle 3 (cont.)

- "A programming organization should not test its own programs."

Rationale:

- By Myers' psychological theory of "constructive" versus "destructive" motivation for programmers and testers, when it is extended to a theory of organisational behaviour.

Prescription:

  - "… it is more economical for testing to be performed by an objective, independent party."

My questions:

  - How likely is it that a third party (especially if they are "objective" and "independent") will have an accurate idea of what the stakeholder wants?
  - Is Myers' 3rd principle appropriate only for the testing of well-specified programs? How can we test our specifications?

Principle 4

- "Thoroughly inspect the results of each test."

Rationale:

- "… probably the most obvious principle, but … often overlooked."
- "We've seen numerous experiments that show many subjects failed to detect certain errors, even when symptoms of those errors were clearly observable on the output listings."
- "… errors that are found on later tests are often missed in the results from earlier tests."

Prescription:

  - The tester should pay attention to all observables. They should **not** restrict their focus to the specified outputs.

My questions:

  - Can software be adequately tested by a fully-automated process?
  - What information should be included on the testing report?
  - Should the tester inspect the system that produced the testing report?

Principle 5

- "Test cases must be written for input conditions that are invalid and unexpected, as well as for those that are valid and expected."

Rationale:

- "Few people, for instance, feed the [triangle-classification] program the numbers 1, 2, 5 to make sure that the program does not erroneously interpret this as a scalene triangle instead of an invalid triangle."
- "… many errors that are suddenly discovered in production programs turn up when the program is used in some new or unexpected way."

Prescription:

- "… test cases representing unexpected and invalid input conditions" should be included, because they "… have a higher error-detection yield than do test cases for valid input conditions."

My questions:

  - If the programmer sees our "unexpected" test cases, then these cases are no longer unexpected… should some of our test cases be secret?
  - How can we generate "unexpected" inputs?

Principle 6

- "Examining a program to see if it does not do what it is supposed to do is only half the battle; the other half is seeing whether the program does what it is not supposed to do."

Rationale:

- "… a payroll program that produces the correct paychecks is still an erroneous program if it also produces extra checks for nonexistent employees or if it overwrites the first record of the personnel file."

Prescription:

- "Programs must be examined for unwanted side effects."

My questions:

  - For black-box testing, should the specification include some invariants, to help the tester know what things they should monitor for unwanted change?
  - For white-box testing, should the program be accompanied by a specification of the system it will be running on, so that the tester can analyse the program to discover the system resources which could, conceivably, be affected by a bug?

Principle 7

- “Avoid throwaway test cases unless the program is truly a throwaway program.”

Rationale:

- “A common practice is to sit at a terminal and invent test cases on the fly, and then send these test cases through the program.”
- “The major problem is that test cases represent a valuable investment… [before retesting] the test cases must be reinvented… people tend to avoid [this work]… the retest is rarely as rigorous as the original test”

Prescription:

- Your test cases are valuable, don’t throw them away!

My question:

  - If you’re actually writing throwaway programs (i.e. rapid versioning) should you save your cases, or should you throw them away too?

Principle 8

- "Do not plan a testing effort under the tacit assumption that no errors will be found."

Rationale:

- "This is a mistake project managers often make." (!)
- Any such effort would not be a "testing process" (under Myers' definition).

Prescription:

- Assume that errors will be found.
  - If no errors are found, this should be a wake-up call to the test manager – the testing process is probably not rigorous enough!!

My questions:

- What about acceptance testing? I think a product should not be presented for a final test unless there's strong confidence that it will pass, i.e. that no errors will be found and the product can be released!
- If an acceptance test is repeatedly failed, then I think there's a good chance the final release will be unacceptable to its end-users (even after it has been bug-fixed to pass the acceptance test.)

Principle 9

- "The probability of the existence of more errors in a section of a program is proportional to the number of errors already found in that section."

Rationale:

- If more errors have been found, to date, in module A than in module B, and if module A has not been purposely subjected to a more rigorous test than module B, then future tests will probably reveal more errors in module A than in module B.
- "… errors tend to come in clusters… in the typical program, some sections seem to be much more prone to errors than other sections"
  - "nobody has supplied a good explanation of why this occurs"
  - This is now an active area of SE research, see e.g. http://dx.doi.org/10.1007/978-3-540-73101-6_18

Prescription:

  - If a particular section of a program seems to be much more prone to errors than other sections, then this phenomenon tells us that, in terms of yield on our testing investment, additional testing efforts are best focused against this error-prone section

My question:

  - What about black-box testing, do errors tend to "cluster" in a portion of the spec?

Principle 10

- "Testing is an extremely creative and intellectually challenging task."

Rationale:

- "It is probably true that the creativity required in testing a large program exceeds the creativity required in designing that program."

Prescription:

- None (!) Perhaps: managers should hire creative and intelligent testers?

My questions:

- Could too much creativity, or too much intellectual curiosity, be undesirable in a tester?
- Should the most creative people be security testers?
  - You might imagine a plausible motivation for an attacker who has plausible skills and access rights, then imagine "what could go wrong" when the program is installed and operated, and then determine whether the attacker would succeed or whether the system running this program would have an adequate defence.

Learning Goals for Today

- Have a working understanding of Myers' principled approach to software testing.
  - Given some information about a testing situation, can you apply Myers' principles? ("What would Myers do in this situation?")
  - Under what conditions are Myers' principles inapplicable or inappropriate?
  - Note that I'm asking you to write some "test cases" for Myers' principles – considering its "unexpected inputs" as well as the inputs he explicitly considered!