CITS5501 Software Testing and Quality Assurance
Quality assurance

Unit coordinator: Arran Stewart

April 10, 2018

Recap – testing

Testing – questions

- How do we decide what unit tests to write?
- Can we tell if we are missing any tests?

What tests to write

- Some may seem “obvious” from the specification

What tests to write

- Some may seem "obvious" from the specification
  - e.g. A method `stripSpaces` is supposed to return a copy of a string with all space characters removed – so test it on a string with some spaces

What tests to write

- Some may seem "obvious" from the specification
  - e.g. A method `stripSpaces` is supposed to return a copy of a string with all space characters removed – so test it on a string with some spaces
- To tackle them in a more methodical way, we can apply input space partitioning

Are we missing any tests?

- some useful techniques for working out if there are gaps in our tests:

Are we missing any tests?

- some useful techniques for working out if there are gaps in our tests:
  - *graph coverage* techniques (for instance) may show us that there are portions of code that have never been tested

Are we missing any tests?

- some useful techniques for working out if there are gaps in our tests:
  - *graph coverage* techniques (for instance) may show us that there are portions of code that have never been tested
  - *mutation testing* can be used to "test the tests" – if we mutate a program, and it still passes all our tests, something is wrong

Mutation example

- A sample method to test (in a Java-like language):

```
int mult(int a, int b) { return a * b; }
```

Mutation example

- A sample method to test (in a Java-like language):
  ```
  int mult(int a, int b) { return a * b; }
  ```
- A possible test:
  ```
  assertEquals( 1, mult(1,1) );
  ```

Mutation example

- A sample method to test (in a Java-like language):
  ```
  int mult(int a, int b) { return a * b; }
  ```
- A possible test:
  ```
  assertEquals( 1, mult(1,1) );
  ```
- Is this a useful test?

Mutation example (2)

- ans.: No, it's terrible.

Mutation example (2)

- ans.: No, it's terrible.
- Consider the following mutation:

```
int mult(int a, int b) { return a * b; }
```

⇒

```
int mult(int a, int b) { return a ** b; }
```

(where ** is a "power" operator)

Mutation example (2)

- ans.: No, it's terrible.
- Consider the following mutation:

```
int mult(int a, int b) { return a * b; }
```

⇒

```
int mult(int a, int b) { return a ** b; }
```

(where ** is a "power" operator)

- `1*1 == 1**1` so our test will still pass -

Mutation example (2)

- ans.: No, it's terrible.
- Consider the following mutation:

```
int mult(int a, int b) { return a * b; }
```

⇒

```
int mult(int a, int b) { return a ** b; }
```

(where ** is a "power" operator)

- 1*1 == 1**1 so our test will still pass -
  - so it's a pretty poor test

Mutation coverage

Mutation *operator* coverage (MOC):

- Have we used every mutation operator at least once?
- This is a pretty weak criterion.
- Suppose our mutation operators are something like . . .
  `CHANGE_CONSTANT` (mutate numeric or string constants)
  `MUTATE_ARITHMETIC_OP` (replace arithmetic operators with others)
- Then as long as our mutation testing package can find one constant to mutate, and one arithmetic operator to mutate, this is satisfied

Mutation coverage (2)

Mutation *production* coverage (MPC):

- Recall what a "production" is - effectively, a branch within a production rule

Mutation coverage (2)

Mutation *production* coverage (MPC):

- Recall what a "production" is - effectively, a branch within a production rule
- e.g. "integer" and "digit":

```
<integer> ::= <digit>|<integer><digit>
<digit> ::=  "0" | "1" | "2" | "3" | "4" | "5" | "6" |
             "7" | "8" | "9"
```

Mutation coverage (2)

Mutation *production* coverage (MPC):

- Recall what a "production" is - effectively, a branch within a production rule
- e.g. "integer" and "digit":

```
<integer> ::= <digit>|<integer><digit>
<digit> ::=  "0" | "1" | "2" | "3" | "4" | "5" | "6" |
             "7" | "8" | "9"
```

- *Production* coverage of this syntax would mean $10 + 2 = 12$ cases.

Mutation coverage (2)

- Mutation production coverage is easiest to understand in the case of mutating, say, some sort of input (e.g. an image file format)

Mutation coverage (2)

- Mutation production coverage is easiest to understand in the case of mutating, say, some sort of input (e.g. an image file format)
- We have some BNF grammar describing the input

Mutation coverage (2)

- Mutation production coverage is easiest to understand in the case of mutating, say, some sort of input (e.g. an image file format)
- We have some BNF grammar describing the input
- We have some mutation operator (say, “mutate digits”)

Mutation coverage (2)

- Mutation production coverage is easiest to understand in the case of mutating, say, some sort of input (e.g. an image file format)
- We have some BNF grammar describing the input
- We have some mutation operator (say, “mutate digits”)
- MPC: For each mutation operator, and for each production (not rule) which the operator can apply to, we have applied the operator

Mutation coverage (2)

- Mutation production coverage is easiest to understand in the case of mutating, say, some sort of input (e.g. an image file format)
- We have some BNF grammar describing the input
- We have some mutation operator (say, “mutate digits”)
- MPC: For each mutation operator, and for each production (not rule) which the operator can apply to, we have applied the operator
- e.g.

```
<height> ::= 'infinite' | <digit>
```

Mutation coverage (2)

- Mutation production coverage is easiest to understand in the case of mutating, say, some sort of input (e.g. an image file format)
- We have some BNF grammar describing the input
- We have some mutation operator (say, "mutate digits")
- MPC: For each mutation operator, and for each production (not rule) which the operator can apply to, we have applied the operator
- e.g.

```
<height> ::= 'infinite' | <digit>
```

  - one of the productions in this rule is a digit, and could be mutated using some digit mutation operator

Mutation of programs

- In the case of `mutpy`, what is being mutated is a program

Mutation of programs

- In the case of `mutpy`, what is being mutated is a program
- Trying to cover every possible mutation is far too many cases

Mutation of programs

- In the case of `mutpy`, what is being mutated is a program
- Trying to cover every possible mutation is far too many cases
- Trying to get operator coverage (or even production coverage) is too few

Mutation of programs

- In the case of `mutpy`, what is being mutated is a program
- Trying to cover every possible mutation is far too many cases
- Trying to get operator coverage (or even production coverage) is too few
- `mutpy` instead produces random mutations of a program – given enough random mutations, we should get reasonable coverage

Quality assurance

Overview

- Source: Pressman, R. Software Engineering: A Practitioner's Approach (McGraw-Hill, 2005)
- Aspects of quality
  - Definition, types of quality costs
- Organizational responsibility
  - Who is responsible for software quality?
- Software reliability, availability, safety
- Quality assurance plans
- Techniques
  - Software reviews

What is Quality Management

- Also called software quality assurance (SQA)
- Serves as an umbrella activity that is applied throughout the software process
- Involves doing the software development correctly versus doing it over again
- Reduces the amount of rework, which results in lower costs and improved time to market

What is Quality Management (cont'd)

SQA encompasses:

- A software quality assurance *process*
  - (i.e. What is our process for ensuring we maintain quality?)
- Specific quality assurance and quality control *tasks* (including formal technical reviews and a multi-tiered testing strategy)
  - (i.e. What tasks are involved?)
- Effective software engineering practices (methods and tools)
  - (e.g. Do we use revision control? Unit testing frameworks? OO analysis and design?)
- Control of all software work products and the changes made to them
  - (Do we know what and where our software artifcats are, and who can change them, and when and why?)
- A procedure to ensure compliance with software development standards
  - (e.g. What checks do we have in place?)
- Measurement and reporting mechanisms
  - (e.g. How do team leaders/management know how we're doing?)

Quality Defined

- "*a* quality" is defined as a characteristic or attribute of something
- Refers to measurable characteristics that we can compare to known standards
- "quality" (in general) is "the degree of excellence of something"
- Software is more difficult in some ways to measure than other things
- But there are still many attributes which we can measure

Software quality – a definition

Definition: "Conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software"

Software quality – a definition (cont'd)

- This definition emphasizes three points
  - Software requirements are the foundation from which quality is measured; lack of conformance to requirements is lack of quality
  - Specified standards define a set of development criteria that guide the manner in which software is engineered; if the criteria are not followed, lack of quality will almost surely result
  - A set of implicit requirements often goes unmentioned; if software fails to meet implicit requirements, software quality is suspect
- Software quality is no longer the sole responsibility of the programmer
  - It extends to software engineers, project managers, customers, salespeople, and the SQA [Software Quality Assurance] group
  - Software engineers apply solid technical methods and measures, conduct formal technical reviews, and perform well-planned software testing

Software quality – a definition (cont'd)

- query: What if there are no (or few) formal requirements?
- query: What if it meets the requirements, but customers are unhappy with it?

Quality Defined (continued)

Some sub-types of quality, taken from manufacturing:

- Quality of design (the quality we're *intending* to offer)
  - The characteristic that designers specify for an item
  - This encompasses requirements, specifications, and the design of the system
- Quality of conformance (i.e., implementation)
  - The degree to which the design specifications are followed during manufacturing
  - This focuses on how well the implementation follows the design and how well the resulting system meets its requirements

Quality – a user perspective

- Ultimately, if the user is not satisfied, nothing else matters
- Robert Glass's formulation:

```
user satisfaction = compliant product +
                    good quality +
                    delivery on time & within budget
```

Quality – a user perspective (cont'd)

- What if the user wanted something that's inadvisable? Or illegal?

Quality – a user perspective (cont'd)

- What if the user wanted something that's inadvisable? Or illegal?
  - "Store all user-names and passwords in a text file, so we can easily manage and change them."

Quality Control

- Involves a series of inspections, reviews, and tests used throughout the software process
- Ensures that each work product meets the requirements placed on it
- Includes a feedback loop to the process that created the work product
  - This is essential in minimizing the errors produced
- Combines measurement and feedback in order to adjust the process when product specifications are not met
- Requires all work products to have defined, measurable specifications to which practitioners may compare to the output of each process

The Cost of Quality

MS Word - it sometimes crashes

- could it be made better quality?

The Cost of Quality

MS Word - it sometimes crashes

- could it be made better quality?
- could Microsoft apply, say, formal methods to get NASA-level quality from MS Word?

The Cost of Quality

MS Word - it sometimes crashes

- could it be made better quality?
- could Microsoft apply, say, formal methods to get NASA-level quality from MS Word?
- should they?

The Cost of Quality

MS Word - it sometimes crashes

- could it be made better quality?
- could Microsoft apply, say, formal methods to get NASA-level quality from MS Word?
- should they?
- What would be the costs? What would be the benefits?

The Cost of Quality

Cost of quality . . .

- includes all costs incurred in the pursuit of quality or in performing quality-related activities
- is studied to
  - Provide a baseline for the current cost of quality
  - Identify opportunities for reducing the cost of quality
  - Provide a normalized basis of comparison (which is usually dollars)
- involves various *kinds* of quality costs (see next slides)
- increases dramatically as the activities progress from
  - Prevention ⇒ Detection ⇒ Internal failure ⇒ External failure

Kinds of Quality Costs

- Prevention costs *(ensure mistakes never creep in)*
  - Quality planning, formal technical reviews, test equipment, training
- Appraisal costs *(check whether they have)*
  - Inspections, equipment calibration and maintenance, testing
- Failure costs *(oops)*
  - subdivided into internal failure costs and external failure costs
  - Internal failure costs
    - Incurred when an error is detected in a product prior to shipment
    - Include rework, repair, and failure mode analysis
  - External failure costs
    - Involves defects found after the product has been shipped
    - Include complaint resolution, product return and replacement, help line support, and warranty work

The SQA (Software Quality Assurance) Group

In an organisation with end-users/customers:

- Serves as the customer's in-house representative
- Assists the software team in achieving a high-quality product
- Views the software from the customer's point of view
  - Does the software adequately meet quality factors?
  - Has software development been conducted according to pre-established standards?
  - Have technical disciplines properly performed their roles as part of the SQA activity?
- Performs a set of of activities that address quality assurance planning, oversight, record keeping, analysis, and reporting (See next slide)

SQA Activities

- Prepares an SQA plan for a project
- Participates in the development of the project's software process description
- Reviews software engineering activities to verify compliance with the defined software process
- Audits designated software work products to verify compliance with those defined as part of the software process
- Ensures that deviations in software work and work products are documented and handled according to a documented procedure
- Records any noncompliance and reports to senior management
- Coordinates the control and management of change
- Helps to collect and analyze software metrics

Software Reliability, Availability, and Safety

Reliability and Availability

- Software *failure*
  - Defined: Nonconformance to software requirements
  - Given a set of valid requirements, all software failures can be traced to design or implementation problems (i.e., nothing wears out like it does in hardware)

Reliability

Software reliability

- Defined: The probability of failure-free operation of a software application in a specified environment for a specified time
- Estimated using historical and development data
- A simple measure is MTBF = MTTF + MTTR = Uptime + Downtime
  (MTBF = mean time between failures)
  (MTTF = mean time to failure)
  (MTTR = mean time to repair)
- Example:
  - MTBF = 68 days + 3 days = 71 days
  - Failures per 100 days = (1/71) * 100 = 1.4

Availability

Software availability

- Defined: The probability that a software application is operating according to requirements at a given point in time
- Availability = [MTTF/ (MTTF + MTTR)] * 100%
- Example:
  - Avail. = [68 days / (68 days + 3 days)] * 100 % = 96%

Software Safety

- Focuses on identification and assessment of potential hazards to software operation
- It differs from software reliability
  - Software reliability uses statistical analysis to determine the likelihood that a software failure will occur; however, the failure may not necessarily result in a hazard or mishap
  - Software safety examines the ways in which failures result in conditions that can lead to a hazard or mishap; it identifies faults that may lead to failures
- Software failures are evaluated in the context of an entire computer-based system and its environment through the process of fault tree analysis or hazard analysis

SQA Plan

Purpose

- Provides a road map for instituting software quality assurance in an organization
- Developed by the SQA group to serve as a template for SQA activities that are instituted for each software project in an organization

SQA structure

Structured as follows:

- The purpose and scope of the plan
- A description of all software engineering work products that fall within the purview of SQA
- All applicable standards and practices that are applied during the software process
- SQA actions and tasks (including reviews and audits) and their placement throughout the software process
- The tools and methods that support SQA actions and tasks
- Methods for assembling, safeguarding, and maintaining all SQA-related records
- Organizational roles and responsibilities relative to product quality