CITS5501 Software Testing and Quality Assurance
Introduction Pt 2 - Concepts in testing

Unit coordinator: Arran Stewart

Software defects

- What does it mean for software to be *wrong* or *defective* in some way?

Software defects

- What does it mean for software to be *wrong* or *defective* in some way?
- Could be that it doesn't do something you expect it to do, or that the documentation says it should do

Software defects

- What does it mean for software to be *wrong* or *defective* in some way?
- Could be that it doesn't do something you expect it to do, or that the documentation says it should do
- Could be that it does it, but poorly (slowly, insecurely, etc.)

Software defects

- Have you experienced defects or problems when using a software-based system?

Software defects

- Have you experienced defects or problems when using a software-based system?
- What about when using a *library*?

Software defects

- Have you experienced defects or problems when using a software-based system?
- What about when using a *library*?
- Are they the same, or different?

“Shrink-wrap” software

- Software which is not customized, but simply made available to mass markets, is sometimes called “shrink-wrapped” software, or said to be provided pursuant to a “click-through” license.
- It is a mass-market commodity (like buying a container of milk): consumers of the software have the option to “take it or leave it” – the provider doesn’t customize it to their particular needs.
- Small, medium and large businesses also buy “shrink-wrap” products.
- When a consumer uses a “shrink-wrap” product, they may rely on the documentation to know what the product should do.
- But if it works exactly according to the documentation, and that’s not what the customer wants, that’s still a problem.

COTS

- More customizable than "shrink-wrap" software is "*Commercial off-the-shelf*" software.
- It *can* be used "out of the box"; but in practice needs to be set up and configured properly to meet a customer's needs.
- This is more like buying something like an alarm system, or a backyard watering and irrigation system – they are available "off-the-shelf", but setting them up properly typically requires some expertise.

Custom or “bespoke” software

- The polar opposite of “shrink-wrap” software is *custom* or “bespoke” software
- This is software that is developed specifically for one customer’s needs (though the provider may have standard components it uses to provide particular solutions).
- The system either needs to be written from scratch (unusual), or at least, requires implementation of software (not just mere “configuration”)
- Exactly what the software should do will (hopefully) be spelled out in a contract, as well as procedures and cost for making changes to the system.

Software components

- Shrink-wrap, COTS and bespoke software are all intended to have *users*.
- For software libraries and components, however, the "users" are other programmers – who often have different sorts of requirements to others.

Scope

- We will initially focus on software that has clear *requirements* – in particular, software libraries and components.

Requirements

- "The function `int square(int n)` should take a single `int` as a parameter, n, and should return the *square* of n – that is, $n \times n$".

Requirements

- "The function `int square(int n)` should take a single `int` as a parameter, n, and should return the *square* of n – that is, $n \times n$".
- Suppose we are given a function someone else has coded, that is supposed to meet this requirement – how can we tell if it does or not?

Requirements

- What about this one?

Requirements

- What about this one?
- "The function `int squareRoot(int n)` should take a single `int` as a parameter, *n*, and should return the *integral square root* of *n* – that is, the largest number *x* such that $x \times x \leq n$.

Requirements

- What about this one?
- "The function `int squareRoot(int n)` should take a single `int` as a parameter, n, and should return the *integral square root* of n – that is, the largest number x such that $x \times x \leq n$.
- Are the requirements clear? What happens for negative numbers?

Requirements

Another example:

- "Objects of class `Student` should store the following data about a particular student – their name (a string) and their date of birth (a `Date` object).

  The `Student` class should have:

  - A constructor take a string and date to initialize the name and date of birth fields
  - A method, `int getAge()`, which returns the students current age (in years)."

- Are these requirements testable?

Software faults, errors & failures

- Software failure: External, incorrect behavior with respect to the requirements or other description of the expected behavior
- Software fault: A static defect in the software
- Software error: An incorrect internal state that is the manifestation of some fault

Software failure

- A software *failure* is the observable way in which software fails to meet its requirements.
- Examples:
  - You call `square(3)` and it returns 0.
  - You try to log into the LMS with your correct username and password, and get a page saying "Server failure"
  - Microsoft Word crashes, losing your work, 5 minutes before you have an assignment due
- Describing a failure doesn't describe *why* or how something went wrong, just that it did.

Software failure, cont'd

- Another example: desired behaviour for a passenger train is that it should stay on the tracks.
- If it is derailed, that can be considered a *failure*.

Software fault

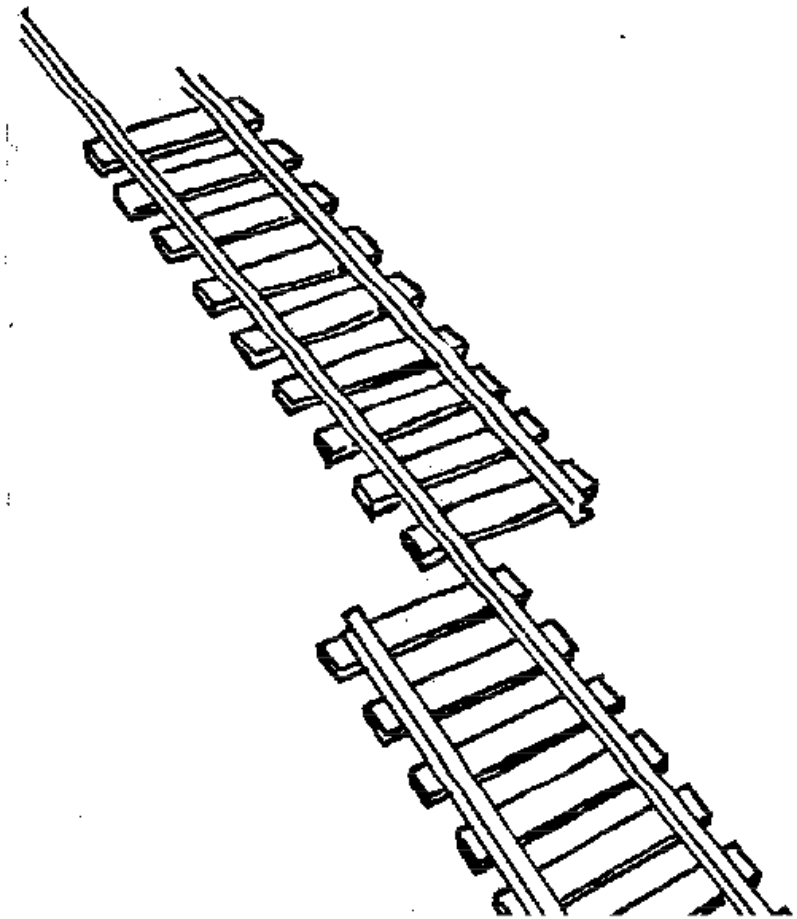
- A software *fault* is some static defect in the system.
- "Static" means it is to be found in the *static artifacts* which make up the system - i.e. the source code.
- For example:

  ```
  int square(n) { min(n * n, 0); }
  ```

  contains a defect – it shouldn't be calling a `min` function at all.

Question

What is this? A failure or a fault?



Errors

- So that's faults and failures.
- The failure is observable behaviour, the fault is something wrong in the soure code (or other static artifact) that gives rise to the failure.
- What about errors?
- They are a little harder.

Errors, cont'd

- An error refers to the state of a running system – an alternative (and clearer) term is *erroneous state*.
- It means a system state that is a *manifestation* at run-time of some fault.
- It is easiest to recognize in the case where there are *invariants* a system must satisfy.

Errors, cont'd

- For instance, suppose we have a *List* data structure, which implements a linked list.
- It can be expensive to work out the *length* of a linked list – the whole list must be traversed – so we might "cache" the length of the list, and just update it every stime something is added to or removed from the list.

```
class LinkedList {
  private Node startNode;
  private int length;

  public getLength() {
    return length;
  }
}
```

Errors, cont'd

- This means that we need to keep the value of the `length` field in sync with the actual length of the list.
- This is expressed as an *invariant* – something that should always hold true of an object, before and after a method call on it.
- In this case, the invariant is that "the `length` field should always hold the actual length of the list".

Errors, cont'd

- If we have, say, an `remove(index i)` method which removes the item at position *i* from the list, and we forget to update the `length` field, then our list is now in an *erroneous state*.

Goals of testing

Based on process maturity:

- Level 0 : There's no difference between testing and debugging
- Level 1 : The purpose of testing is to show correctness
- Level 2 : The purpose of testing is to show that the software doesn't work
- Level 3 : The purpose of testing is not to prove anything specific, but to reduce the risk of using the software
- Level 4 : Testing is a mental discipline that helps all IT professionals develop higher quality software

Level 0 Thinking

- Testing is the same as debugging
- Does not distinguish between incorrect behavior and mistakes in the program
- Does not help develop software that is reliable or safe

Level 1 Thinking

- Purpose is to show correctness
- Correctness is impossible to achieve
- What do we know if no failures?
  - Good software, or bad tests?
- Test engineers have no:
  - Strict goal
  - Real stopping rule
  - Formal test technique
  - Test managers are powerless

Level 2 Thinking

- Purpose is to show failures
- Looking for failures is a negative activity
- Puts testers and developers into an adversarial relationship
- What if there are no failures?

Level 3 Thinking

- Testing can only show the presence of failures
- Whenever we use software, we incur some risk
- Risk may be small and consequences unimportant
- Risk may be great and consequences catastrophic
- Testers and developers cooperate to reduce risk

Level 4 Thinking

A mental discipline that increases quality

- Testing is only one way to increase quality
- Test engineers can become technical leaders of the project
- Primary responsibility to measure and improve software quality
- Their expertise should help the developers