SOFTENG 254:
Quality Assurance

Lecture 1b: Quality

Paramvir Singh
School of Computer Science

- PAQ
- Agenda
- V & V
- The V Model
- Quality
- Correctness
- Terminology
- Example
- RIP model
- “Faults” not “Bugs”
- Fault-free code
- Testing vs. Debugging
- Testing
- Terminology
- Reality
- Bugs
- Key Points

Potential Assessment Question

Which of the following would be **least** useful to include in a definition of “software quality”?

Quality software:

(a) is not painful to use.

(b) has no bugs.

(c) is what the customer wants.

(d) is fit for purpose.

Justify your answer.

- PAQ
- Agenda
- V & V
- The V Model
- Quality
- Correctness
- Terminology
- Example
- RIP model
- “Faults” not “Bugs”
- Fault-free code
- Testing vs. Debugging
- Testing
- Terminology
- Reality
- Bugs
- Key Points

Agenda

- Verification and Validation
- What is testing

- PAQ
- Agenda
- V & V
- The V Model
- Quality
- Correctness
- Terminology
- Example
- RIP model
- “Faults” not “Bugs”
- Fault-free code
- Testing vs. Debugging
- Testing
- Terminology
- Reality
- Bugs
- Key Points

V & V

Verification ensure that the system is being built according to the process, that every activity has been carried out correctly — that the thing has been built right

Validation ensure that the system has implemented all of the requirements — the right thing has been built

IV&V Independent verification and validation — carried out by an independent body

- PAQ
- Agenda
- V & V
- The V Model
- Quality
- Correctness
- Terminology
- Example
- RIP model
- “Faults” not “Bugs”
- Fault-free code
- Testing vs. Debugging
- Testing
- Terminology
- Reality
- Bugs
- Key Points

The V Model

- Validation — the right thing has been built

⇒ need to check that requirements are correct

⇒ *customer* or *acceptance* testing

- Verification — the thing has been built right

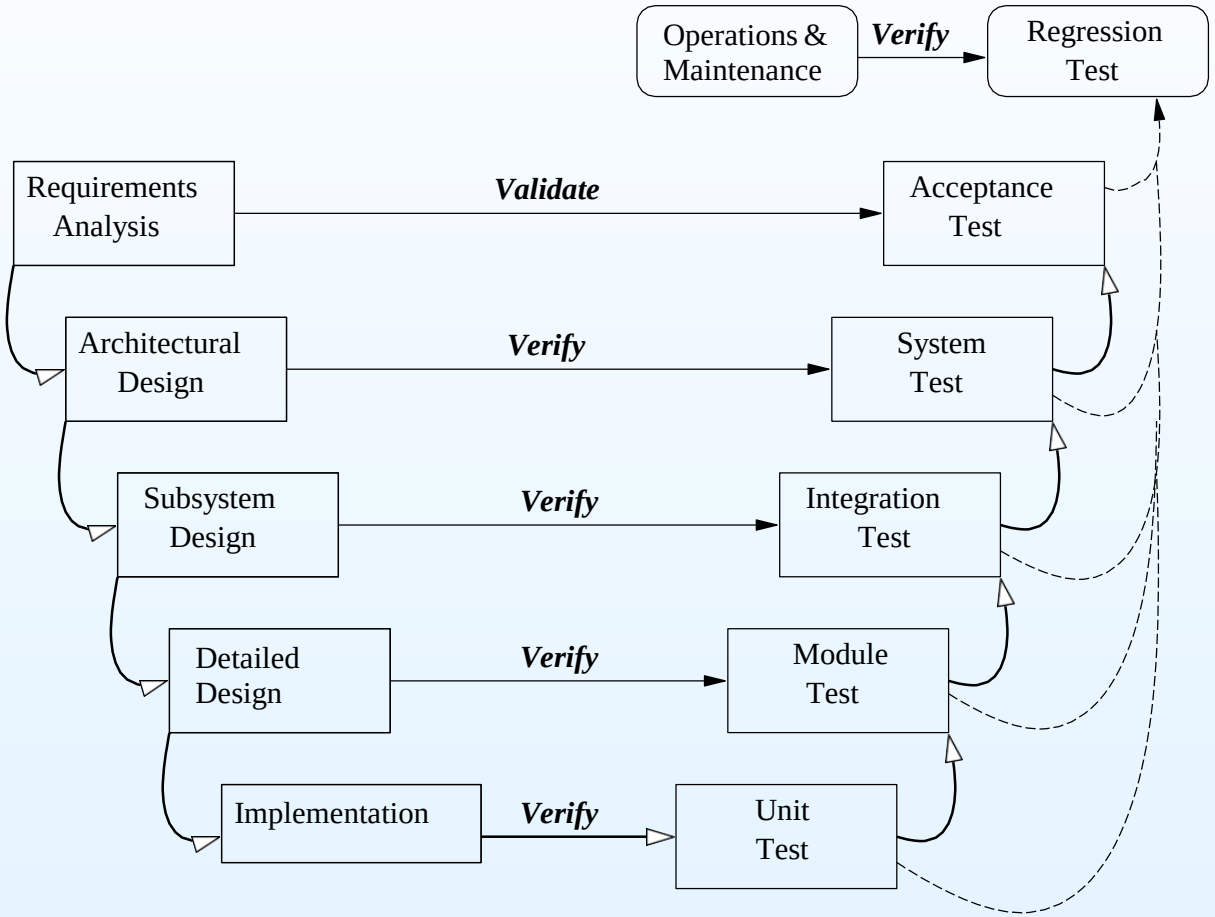
⇒ need to check that each stage of the lifecycle is done correctly

⇒

  - Architecture/Whole design is correct — *system* testing
  - Implementation is correct, that is, each bit is correct — *unit* testing, and the bits have been put together correctly *integration* testing
  - Maintenance doesn’t break anything that used to work — *regression* testing

The V Model

- PAQ
- Agenda
- V & V
- The V Model
- Quality
- Correctness
- Terminology
- Example
- RIP model
- “Faults” not “Bugs”
- Fault-free code
- Testing vs. Debugging
- Testing
- Terminology
- Reality
- Bugs
- Key Points



- PAQ
- Agenda
- V & V
- The V Model
- Quality
- Correctness
- Terminology
- Example
- RIP model
- “Faults” not “Bugs”
- Fault-free code
- Testing vs. Debugging
- Testing
- Terminology
- Reality
- Bugs
- Key Points

Quality: Fitness for purpose

- ensuring fitness for purpose depends on what artifact we are considering, what part of the lifecycle the artifact comes from, and what quality attribute we care about
- e.g. checking that the code doesn’t fail in some horrible way is different to checking that the requirements are the right ones
- e.g. checking that the test suite is of good quality is different than checking that the documentation is of good quality
- e.g. checking that the system is fast enough is different than checking that it is secure enough

- PAQ
- Agenda
- V & V
- The V Model
- Quality
- Correctness
- Terminology
- Example
- RIP model
- “Faults” not “Bugs”
- Fault-free code
- Testing vs. Debugging
- Testing
- Terminology
- Reality
- Bugs
- Key Points

Quality: Fitness for purpose

- ensuring fitness for purpose depends on what artifact we are considering, what part of the lifecycle the artifact comes from, and what quality attribute we care about
- e.g. checking that the code doesn’t fail in some horrible way is different to checking that the requirements are the right ones
- e.g. checking that the test suite is of good quality is different than checking that the documentation is of good quality
- e.g. checking that the system is fast enough is different than checking that it is secure enough
- SOFTENG 254 Part 1:

  $\Rightarrow$ artifact $=$ code

  $\Rightarrow$ lifecycle $=$ implementation, operation

  $\Rightarrow$ quality $=$ correctness

- PAQ
- Agenda
- V & V
- The V Model
- Quality
- Correctness
- Terminology
- Example
- RIP model
- “Faults” not “Bugs”
- Fault-free code
- Testing vs. Debugging
- Testing
- Terminology
- Reality
- Bugs
- Key Points

What is (code) correctness?

- code does what it is supposed to
- code behaves the way we expect it to (but what if our expectation is “wrong”?)
- checks the functionality requirements are met by the code
- code that behaves in a way that it is not supposed to is said to have failed or that a failure has occurred
- something in the code must have caused the failure, a fault

- PAQ
- Agenda
- V & V
- The V Model
- Quality
- Correctness
- Terminology
- Example
- RIP model
- “Faults” not “Bugs”
- Fault-free code
- Testing vs. Debugging
- Testing
- Terminology
- Reality
- Bugs
- Key Points

Failures vs. Faults vs. Errors.

Fault The abnormal aspect of the code that, if executed, will put the system into an error state

- forget to initialise a field in one constructor so that it is `null` when it should not be

Error The system is in a state different from what it is supposed to be, which if not dealt with, will lead to a failure

- if the constructor is executed, the system is in an error state (but another constructor may be used)

Failure Externally observed incorrect behaviour of code

- if the field is ever dereferenced, then `NullPointerException` (but it may never be dereference)

- PAQ
- Agenda
- V & V
- The V Model
- Quality
- Correctness
- Terminology
- Example
- RIP model
- “Faults” not “Bugs”
- Fault-free code
- Testing vs. Debugging
- Testing
- Terminology
- Reality
- Bugs
- Key Points

Example

```
public class IntList {
  private int[] _values;
  public IntList(int[] xs) {...}
 /**
  * Return the number of zeros that appear in the specified array
  * @param xs The array to examine
  * @return The number of zeros that occur in the list
  */
  public int numZeros() {
     int count = 0;
    for (int i = 1; i < _values.length; i++) {
      if (_values[i] == 0) {
        count++;
      }
    }
    return count;
  }
}

  ...
  int[] input = { 1, 0, 2, 5, 0};
  IntList list = new IntList(input);
  System.out.println(list.numZeros());   ⇒ 2 ---   correct (but error)
  ...
  int[] input = { 0, 1, 2, 5, 0};  list
  = new IntList(input);
  System.out.println(list.numZeros());   ⇒ 1 ---   incorrect (failure)
```

Fault

- PAQ
- Agenda
- V & V
- The V Model
- Quality
- Correctness
- Terminology
- Example
- RIP model
- “Faults” not “Bugs”
- Fault-free code
- Testing vs. Debugging
- Testing
- Terminology
- Reality
- Bugs
- Key Points

Example continued

```
int[] input ={ 1, 0, 2, 5, 0};
IntList list =new IntList(input);
System.out.println(list.numZeros());   ⇒ 2 ---  correct (but error)
```

1. `int count = 0` count is supposed to have value 0
2. `int i = 1;` i is supposed to have value 0, so error
3. `if (_values[i] == 0)` i is 1
4. `count++` count is 1, which is correct
5. `i < _values` i is 1
6. `i++` i is 2
7. `if (_values[i] == 0)`
8. ...

- PAQ
- Agenda
- V & V
- The V Model
- Quality
- Correctness
- Terminology
- Example
- RIP model
- “Faults” not “Bugs”
- Fault-free code
- Testing vs. Debugging
- Testing
- Terminology
- Reality
- Bugs
- Key Points

RIP model

- In order for a failure to be observed:

Reachability There must be an input that will reach the location containing the fault

Infection Having executed the location containing the fault, the program must be in an error state

Propagation The error (infected) state must propagate to cause some output of the program to be incorrect.

- PAQ
- Agenda
- V & V
- The V Model
- Quality
- Correctness
- Terminology
- Example
- RIP model
- "Faults" not "Bugs"
- Fault-free code
- Testing vs. Debugging
- Testing
- Terminology
- Reality
- Bugs
- Key Points

"Faults" not "Bugs"

- "Bugs" implies *sometime during the night when no one was looking a bug crept into the code*
- Translation: "It's not my fault. I didn't do it. I wasn't even there."
- Reality:

Faults do not appear spontaneously. Software does not 'wear out'.
Faults are due to actions by humans.

- PAQ
- Agenda
- V & V
- The V Model
- Quality
- Correctness
- Terminology
- Example
- RIP model
- “Faults” not “Bugs”
- Fault-free code
- Testing vs. Debugging
- Testing
- Terminology
- Reality
- Bugs
- Key Points

Producing Fault-free code

- Avoid faults, by using tools and processes that reduce the chance of faults being caused
- Detect faults — check the code to determine whether or not there are any faults in it.
- fault detection can be done statically — e.g., review the source code and hope to spot faults
- dynamic fault detection — e.g., execute the code and see whether any failures occur

⇒ testing

- PAQ
- Agenda
- V & V
- The V Model
- Quality
- Correctness
- Terminology
- Example
- RIP model
- “Faults” not “Bugs”
- Fault-free code
- Testing vs. Debugging
- Testing
- Terminology
- Reality
- Bugs
- Key Points

Testing vs. Debugging

- Testing is (trying) to produce failures
- Debugging is (trying to) find the fault that caused the failure
  - Find the point where the program enters the error state

- PAQ
- Agenda
- V & V
- The V Model
- Quality
- Correctness
- Terminology
- Example
- RIP model
- “Faults” not “Bugs”
- Fault-free code
- Testing vs. Debugging
- Testing
- Terminology
- Reality
- Bugs
- Key Points

What is testing?

> *An activity in which a system or a component is executed under specific conditions, the results are observed or recorded, and an evaluation is made of some aspect of the system or component.* **[IEEE]**

- randomly executing the system is not testing
- executing the system but ignoring the results is not testing
- recording the results but not actually determining what they tell us about the system is not testing
- ⇒ testing must be done in an organised, *systematic*, manner, following a process

- PAQ
- Agenda
- V & V
- The V Model
- Quality
- Correctness
- Terminology
- Example
- RIP model
- “Faults” not “Bugs”
- Fault-free code
- Testing vs. Debugging
- Testing
- Terminology
- Reality
- Bugs
- Key Points

Terminology II

- Implementation under test (IUT), System under test (SUT), Module under test (MUT), Component under test (CUT) . . .
- Test suite — a set of tests
- Unit test — IUT is “small” and “incomplete” typically a procedure/function/method but lines between unit and module often blurred
- Module test — MUT collection of units typically a file/class
- Integration test — IUT is a combination of modules
- System test — IUT is the complete system
- Acceptance test — tests specified by the customer
- Regression test — run tests that showed no failure in the past to check that no failure has been introduced during maintenance operations.

- PAQ
- Agenda
- V & V
- The V Model
- Quality
- Correctness
- Terminology
- Example
- RIP model
- “Faults” not “Bugs”
- Fault-free code
- Testing vs. Debugging
- Testing
- Terminology
- Reality
- Bugs
- Key Points

More terminology

- Test case — specifies the *pretest state* of the IUT, the *inputs*, and the *expected state or behaviour*
- **Test** — carry out the test case
  1. Get the IUT into the pretest state
     - May require input values (aka prefix values) separate from those required for the actual test
  2. Supply the test inputs (aka test case values)
  3. Execute the IUT with the test case values
  4. Perform any necessary post-test actions to terminate the test
     - May also require extra inputs values (aka postfix values)
  5. Compare the observed state or behaviour of the IUT with the expected state or behaviour
  6. Report results. If the resulting state or behaviour matches the expected results, then the test has *passed*, otherwise it *fails*.

- PAQ
- Agenda
- V & V
- The V Model
- Quality
- Correctness
- Terminology
- Example
- RIP model
- “Faults” not “Bugs”
- Fault-free code
- Testing vs. Debugging
- Testing
- Terminology
- Reality
- Bugs
- Key Points

On the Usefulness of Testing

- “Program testing can be used to show the presence of bugs, but never to show their absence.” **Notes on Structured Programming**, Edsger W. Dijkstra, 1969.
- Proofs of correctness are often as hard (if not harder) to get right than writing the code. . .

- PAQ
- Agenda
- V & V
- The V Model
- Quality
- Correctness
- Terminology
- Example
- RIP model
- “Faults” not “Bugs”
- Fault-free code
- Testing vs. Debugging
- Testing
- Terminology
- Reality
- Bugs
- Key Points

Photo # NH 96566-KN First Computer "Bug", ~~1945~~ 1947?

9/9

0800 Arctan started

1000 " stopped - arctan ✓

{ 1.2700 9.037 847 025
9.037 846 ?95 correct

13" wc (032) MP - MC ~~1.9821[illegible]000~~ ~~2.13047641[illegible]~~) 4.615925059(-2)

(033) PRO 2 2.13047641[illegible]5

correct 2.13067641[illegible]5

Relays 6-2 in 033 failed special speed test

In relay . 11.00 test .

Relays changed

1100 Started Cosine Tape (Sine check)

1525 Started Multy Adder Test.

1545 

Relay #70 Panel F
(moth) in relay.

First actual case of bug being found.

~~1945~~ 1630 Arctangent started.

1700 closed down.

(Not actually found by Rear Admiral Grace Hopper, she just liked the story)

- PAQ
- Agenda
- V & V
- The V Model
- Quality
- Correctness
- Terminology
- Example
- RIP model
- “Faults” not “Bugs”
- Fault-free code
- Testing vs. Debugging
- Testing
- Terminology
- Reality
- Bugs
- Key Points

Key Points

- In general, quality can be described as “fitness for purpose,” and depends on the artifact, the lifecycle stage, and the quality attribute
- Testing of code is probably the most-studied aspect of software engineering
- Testing cannot “prove” correctness, all it can prove is existence of faults.