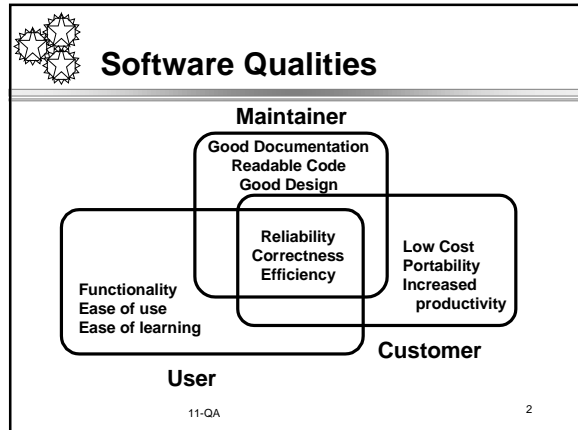



# Quality Assurance


11-QA 1

# Software Quality Assurance

- Use of analysis to validate artifacts
  - requirements, designs, code, test plans
- Technical reviews
- Document reviews
- Compliance to standards
- Control of changes


11-QA 3



# Costs of Poor Quality

- Increased time to find and fix problems
- Increased cost to distribute modifications
- Increased customer support
- Product liability
- Failure in the market place


11-QA 4



# Software Reviews

- Individuals read and comment on the software artifacts
- Very human intensive
- Overriding evidence shows that it
  - improves quality and productivity
  - reduces cost
- It is usually one of the first activities to be dropped when schedules get tight

11-QA 5



# Software Reviews (cont.)

- **Applicable to all software artifacts**
  - code inspections
  - requirements and design reviews
  - walk-throughs
- **Recent research shows that**
  - particular kind of review, size of team, etc. doesn't matter
  - need at least one good, dedicated person doing the review

11-QA 6



## Typical Review Team

- Developer -- presents the material
- Moderator -- keeps the review on track
  - makes sure everyone abides by the process
- Secretary -- takes minutes, documents problems found
- Optionally
  - Apprentice -- learning about the project
  - Domain expert -- familiar with the domain and can verify assumptions

11-QA

7



## Software Review Guidelines

- Review the artifact
  - don't attack the developer
- Stick to an agenda
- Limit debate
  - watch out for "religious" issues
  - watch out for stylistic issues that don't affect maintainability
- Identify problems, not solutions
- Keep accurate notes
- Establish and follow evaluation guidelines
- Limit number of participants

11-QA

8



## Technical Review Guidelines (cont.)

- Prepare beforehand
  - both developers and reviewers
- Allocate resources for reviews
  - people and time
- Possible outcomes
  - accept product as is
  - reject until modified
  - reject product outright
  - accept product provisionally

11-QA

9



## Sample evaluation Guidelines: Code Inspection

- Has the design been correctly translated to code?
- Are language features used appropriately?
- Are coding standards followed?
  - be careful! make sure the standard makes a difference
- Are documentation standards followed?
- Are there misspellings or typos?
- Are the comments accurate and unambiguous?

11-QA

10



## Sample evaluation Guidelines: Code Inspection (cont.)

- Are data types and declarations appropriate?
- Are all constants correct?
- Are all variables initialized before being used?
- Are there overly complex conditions?
- Is there unreachable code?
- Are there obvious inefficiencies?

11-QA

11



## QA Terminology

- |               |                |
|---------------|----------------|
| ● Correctness | ● Fault        |
| ● Reliability | ● Error        |
| ● Testing     | ● Verification |
| ● Debugging   | ● Validation   |
| ● Failure     | ● V&V          |

11-QA

12



## Terminology

- **Correctness:** artifact is *consistent* with its specification
  - Specification could be wrong or incomplete
  - Rarely is software known to be correct
  - Rarely is the specification correct
- **Reliability:** *probability* that the software is correct
  - Statistical measure based on past performance
    - ◆ e.g., mean time to failure

11-QA

13



## More terminology

- **Testing:** entails *executing* the software on selected test cases
  - Evaluate the results (oracle)
  - Evaluate the performance
  - Evaluate the ease of use
- Common Wisdom: Testing reveals bugs but does not guarantee the absence of bugs
  - How should you select test cases?
  - How do you know when to stop testing?

11-QA

14



## More terminology

- **Failure:** an erroneous result
  - incorrect outputs/response for given inputs/stimuli
  - fails to meet real-time constraints
- **Error:** incorrect concept
  - may cause failures if not corrected
- **Fault:** the cause of one or more failures
  - discovered after release

11-QA

15



## More terminology

- **Debugging:** the process of finding the cause of a “bug” and a way to fix it
  - w/o introducing additional bugs!
- **Verification:** process of proving, using mathematical reasoning, that a program is “correct”
  - proofs vs. modelchecking
  - is expensive and is not always possible
  - is not foolproof

11-QA

16



## More terminology

- **Validation:** process associated with showing that the software performs reasonably well
- **V & V:** verification & validation?
  - more typically equated with validation

11-QA

17



## Many different kinds of testing

- **Unit testing:** test individual components
  - test stubs simulate called components
  - test harness simulates “outer” context and maintains stubs
- **Integration testing:** combine components and test them
  - follows build plan
- **System testing:** test whole system

11-QA

18



## More kinds of testing

- **Acceptance testing:** testing to determine if the product is acceptable
- **Regression testing:** retesting after the system has been modified
  - determine "old" test cases that must be re-executed
  - determine what new test cases are required

11-QA

19



## More kinds of testing

- **Black box / functional testing:**
  - testing based on specifications
- **White box / structural testing:**
  - testing based on looking at the artifact
- Both black box and white box testing are needed

11-QA

20



## Testing is hard work

- Typically 50% of software development effort goes into testing
  - up to 85% for life-critical software
- How to identify "good" test cases?
  - high probability of finding a new error
  - hits "boundary" conditions
  - "weirdo" cases
    - ◆ often reveal bad assumptions and/or lack of rigor
- Objective is to find errors
  - test case is "successful" if it finds a new error

11-QA

21



## Testing is hard work (cont.)

- Psychologically difficult for a programmer to test his/her own code thoroughly
- Exhaustive testing requires testing all combinations of input values
  - Sorting an array of size 10 containing integers in the range 1 . . 10 has 10! combinations (3,628,800 cases)

11-QA

22



## Testing

- **CAN:**
  - Uncover errors
  - Show specifications are met for specific test cases
  - Be an indication of overall reliability
  - Increase reliability (why??)
- **CANNOT:**
  - Prove that a program is error-free
  - Serve as verification (why??)

11-QA

23



## Testing Principles

- Tests should be traceable to requirements
- Tests should be planned long before testing begins
- Exhaustive testing is not possible
  - 80% of all errors typically occur in 20% of the modules
  - test cases should be chosen to maximize likelihood of finding an error

11-QA

24



## Testing Principles (cont.)

- Testing should be done by someone other than the developers
  - Developers do original testing
  - SQA does independent testing
    - ◆ usually black box testing
- Automated testing tools should be used
  - Reduce testing costs
  - Reduce likelihood of human error

11-QA

25



## Testability

- **Simple software is easier to test**
  - minimize coupling, maximize cohesion
- **Output is sufficient to determine correct behavior**
- **Performs its own tests for internal errors**
  - raises meaningful exceptions
- **All code is reachable**
- **Independent modules can be tested in isolation**
- **Documentation is complete and accurate**

11-QA

26



## Quality is an on-going concern

- **You can't build quality into a system after the fact**
- **Quality should be a consideration during every phase of development**
- **Plan for testing / validation in all phases**
  - requirements -> functional test cases
  - design -> functional and structural test cases
  - code -> enhanced func & struc test cases
  - maintenance -> further enhanced func & struc test cases

11-QA

27



## Debugging

- Find the cause of a failure and fix it
  - an art, not a science
- Debugging is difficult because
  - symptom may appear long after the fault occurs
  - symptom may be difficult to reproduce
  - symptom may be intermittent
- Unit testing helps localize errors

11-QA

28



## SQA Summary

- U.S. software costs \$200 billion/year
- Need to
  - improve software quality
  - reduce costs
    - ◆ V&V is over 50% of the cost
- Improving V&V should reduce costs significantly while improving quality

11-QA

29



## Introduction to Formal Verification

How many tests do you have to do to show  $d=nx$ , always?? (or that the loop even works...)



$i = 0$   
 $d = 0$   
 do  $i \neq n$   
 $i = i + 1$   
 $d = d + x$

Need a way to "prove" properties in general.

Proofs      Model Checking

Formal      Mathematically based

11-QA

30

### Example: Proving A Loop Correct

```

i = 0
d = 0
do i ≠ n
  i = i + 1
  d = d + x
od

```

← Claim this calculates  $nx$  (result in  $d$ ) for  $n \geq 0$  and some  $x$

We'll do this using an *invariant*, and a theorem about loops.

11-QA 31

### Weakest Precondition

This is a program statement, like an IF, or DO, or assignment

This is a predicate, (perhaps a logical expression like  $x > y$ )

$wp(S, P)$

$wp()$  is a function that produces a predicate. The predicate  $wp()$  produces describes a set of states. If  $S$  starts in one of these states,  $P$  will be true when it finishes.

**Example:**

$$wp(i := i + 1, i > 1 \wedge i < 5) \equiv i > 0 \wedge i < 4$$

11-QA 32

### Loop Theorem

For an integer function  $t$  bounded by 0, and

Then:

---

Point 2 means  $P$  is invariant and the integer function decreases each time through the loop

The main repetition theorem is due to A.J.M. van Gastelen.

11-QA 33

### Requirement #1

So, let  $P$  be:

← This is called an "invariant"

Direct substitution produces:

$$\underbrace{P}_{i=0, d=0, do\ i \neq n, i=i+1, d=d+x} \underbrace{B}_{i \leq n} \underbrace{t}_{i \leq k} \Rightarrow 0 \leq i < n \wedge d = ix \Rightarrow n > i$$

$$\begin{aligned}
&i = 0 \\
&d = 0 \\
&do\ i \neq n \\
&\quad i = i + 1 \\
&\quad d = d + x
\end{aligned}$$

11-QA 34

### Requirement #2 / "t" part

means  $t$  must go down

The loop replaces  $i$  with  $i+1$ , so that's what we do

$$n - i = k \Rightarrow n - (i + 1) < k \Rightarrow n - i - 1 < k$$

```

i = 0
d = 0
do i ≠ n
  i = i + 1
  d = d + x

```

11-QA 35

### Requirement #2 / invariant part

Need to show that  $P \wedge B$  before the loop

we use the same trick again, and substitute  $i+1$  for  $i$  and  $d+x$  for  $d$  and prove  $P$  is still true.

```

i = 0
d = 0
do i ≠ n
  i = i + 1
  d = d + x

```

11-QA 36



## Showing Invariant Holds

$$0 \leq i \leq n \wedge d = ix \Rightarrow 0 \leq i+1 \leq n \wedge d = (i+1)x$$



$$0 \leq i \leq n \Rightarrow 0 \leq i+1 \leq n$$

But  $B$  is true, so  $i \neq n$

$$0 \leq i < n \Rightarrow 0 \leq i+1 \leq n$$

$$d = ix \Rightarrow d + x = (i+1)x$$

$$d = ix \Rightarrow d + x = ix + x$$

$$d = ix \Rightarrow d = ix$$

11-QA

37



## Payoff

So we have  $P \Rightarrow wp(DO, P \wedge \neg B)$  by the theorem

We know that we can make  $P$  true before the loop, so we have a set of states such that  $P \wedge \neg B$  is true.

$$\underbrace{0 \leq i \leq n \wedge d = ix}_{P} \wedge \underbrace{i = n}_{\neg B}$$



$$0 \leq n \leq n \wedge d = nx \equiv d = nx$$

**And, we are done.**

```
i = 0
d = 0
do i ≠ n
  i = i + 1
  d = d + x
od
```

11-QA

38