Quality Assurance: Test Development & Execution

CSE 403
Lecture 23

Slides derived from a talk by Ian King

Test Specifications

- What questions do I want to answer about this code? Think of this as experiment design
- In what dimensions will I ask these questions?
  - Functionality
  - Security
  - Reliability
  - Performance
  - Scalability
  - Manageability

Test specification: example

- CreateFile method
  - Should return valid, unique handle for
    - initial 'open' for appropriate resource
    - subsequent calls for shareable resource
    - for files, should create file if it doesn't exist
  - Should return NULL handle and set error indicator if resource is
    - nonexistent device
    - inappropriate for 'open' action
    - in use and not shareable
    - unavailable because of error condition (e.g. no disk space)
  - Must recognize valid forms of resource name
    - Filename, device, ?

Test Plans

- How will I ask my questions? Think of this as the "Methods" section
- Understand domain and range
- Establish equivalence classes
- Address domain classes
  - Valid cases
  - Invalid cases
  - Boundary conditions
  - Error conditions
  - Fault tolerance/stress/performance

Test plan: goals

- Enables development of tests
- Proof of testability – if you can't design it, you can't do it
- Review: what did you miss?

Test plan: example

- CreateFile method
  - Valid cases
    - execute for each resource supporting 'open' action
      - opening existing device
      - opening existing file
      - opening (creating) nonexistent file
    - execute for each such resource that supports sharing
      - multiple method calls in separate threads/processes
      - multiple method calls in single thread/process
  - Invalid cases
    - nonexistent device
    - file path does not exist
    - in use and not shareable
  - Error cases
    - insufficient disk space
    - invalid form of name
    - permissions violation
  - Boundary cases
    - e.g. execute to/past system limit on open device handles
    - device name at/past name length limit (MAXPATH)
  - Fault tolerance
    - execute on failed/corrupted filesystem
    - execute on failed but present device

Performance testing

- Test for performance behavior
  - Does it meet requirements?
    - Customer requirements
    - Definitional requirements (e.g. Ethernet)
- Test for resource utilization
  - Understand resource requirements
- Test performance early
  - Avoid costly redesign to meet performance requirements

Security Testing

- Is data/access safe from those who should not have it?
- Is data/access available to those who should have it?
- How is privilege granted/revoked?
- Is the system safe from unauthorized control?
  - Example: denial of service
- Collateral data that compromises security
  - Example: network topology

Stress testing

- Working stress: sustained operation at or near maximum capability
- Goal: resource leak detection
- Breaking stress: operation beyond expected maximum capability
- Goal: understand failure scenario(s)
  - "Failing safe" vs. unrecoverable failure or data loss

Globalization

- Localization
  - UI in the customer's language
  - German overruns the buffers
  - Japanese tests extended character sets
- Globalization
  - Data in the customer's language
  - Non-US values ($ vs. Euro, ips vs. cgs)
  - Mars Global Surveyor: mixed metric and SAE

Test Cases

- Actual "how to" for individual tests
- Expected results
- One level deeper than the Test Plan
- Automated or manual?
- Environmental/platform variables

Test case: example

- CreateFile method
  - Valid cases
    - English
      - open existing disk file with arbitrary name and full path, file permissions allowing access
        - create directory 'c:\foo'
        - copy file 'bar' to directory 'c:\foo' from test server; permissions are 'Everyone: full access'
        - execute CreateFile('c:foo\bar', etc.)
        - expected: non-null handle returned

Test Harness/Architecture

- Test automation is nearly always worth the time and expense
- How to automate?
  - Commercial harnesses
  - Roll-your-own
  - Record/replay tools
  - Scripted harness
- Logging/Evaluation

Test Schedule

- Phases of testing
  - Unit testing (may be done by developers)
  - Component testing
  - Integration testing
  - System testing
- Dependencies – when are features ready?
  - Use of stubs and harnesses
- When are tests ready?
  - Automation requires lead time
- The long pole – how long does a test pass take?

Where The Wild Things Are: Challenges and Pitfalls

- "Everyone knows" – hallway design
- "We won't know until we get there"
- "I don't have time to write docs"
- Feature creep/design "bugs"
- Dependency on external groups

Test Schedule

- Phases of testing
  - Unit testing (may be done by developers)
  - Component testing
  - Integration testing
  - System testing
  - Usability testing

What makes a good tester?

- Analytical
  - Ask the right questions
  - Develop experiments to get answers
- Methodical
  - Follow experimental procedures precisely
  - Document observed behaviors, their precursors and environment
- Brutally honest
  - You can't argue with the data

How do test engineers fail?

- Desire to "make it work"
  - Impartial judge, not "handyman"
- Trust in opinion or expertise
  - Trust no one – the truth (data) is in there
- Failure to follow defined test procedure
  - How did we get here?
- Failure to document the data
- Failure to believe the data

Testability

- Can all of the feature's code paths be exercised through APIs, events/messages, etc.?
  - Unreachable internal states
- Can the feature's behavior be programmatically verified?
- Is the feature too complex to test?
  - Consider configurations, locales, etc.
- Can the feature be tested timely with available resources?
  - Long test latency = late discovery of faults

What color is your box?

- Black box testing
  - Treats the SUT as atomic
  - Study the gazinta's and gozouta's
  - Best simulates the customer experience
- White box testing
  - Examine the SUT internals
  - Trace data flow directly (in the debugger)
  - Bug report contains more detail on source of defect
  - May obscure timing problems (race conditions)

Designing Good Tests

- Well-defined inputs and outputs
  - Consider environment as inputs
  - Consider 'side effects' as outputs
- Clearly defined initial conditions
- Clearly described expected behavior
- Specific – small granularity provides greater precision in analysis
- Test must be at least as verifiable as SUT

Types of Test Cases

- Valid cases
  - What should work?
- Invalid cases
  - Ariane V – data conversion error (http://www.cs.york.ac.uk/hise/safety-critical-archive/1996/0055.html)
- Boundary conditions
  - Fails in September?
  - Null input
- Error conditions
  - Distinct from invalid input

Manual Testing

- Definition: test that requires direct human intervention with SUT
- Necessary when:
  - GUI is present
  - Behavior is premised on physical activity (e.g. card insertion)
- Advisable when:
  - Automation is more complex than SUT
  - SUT is changing rapidly (early development)

Automated Testing

- Good: replaces manual testing
- Better: performs tests difficult for manual testing (e.g. timing related issues)
- Best: enables other types of testing (regression, perf, stress, lifetime)
- Risks:
  - Time investment to write automated tests
  - Tests may need to change when features change

Types of Automation Tools: Record/Playback

- Record “proper” run through test procedure (inputs and outputs)
- Play back inputs, compare outputs with recorded values
- Advantage: requires little expertise
- Disadvantage: little flexibility - easily invalidated by product change
- Disadvantage: update requires manual involvement

Types of Automation Tools: Scripted Record/Playback

- Fundamentally same as simple record/playback
- Record of inputs/outputs during manual test input is converted to script
- Advantage: existing tests can be maintained as programs
- Disadvantage: requires more expertise
- Disadvantage: fundamental changes can ripple through MANY scripts

Types of Automation Tools: Script Harness

- Tests are programmed as modules, then run by harness
- Harness provides control and reporting
- Advantage: tests can be very flexible
- Disadvantage: requires considerable expertise and abstract process

Test Corpus

- Body of data that generates known results
- Can be obtained from
  - Real world – demonstrates customer experience
  - Test generator – more deterministic
- Caveats
  - Bias in data generation
  - Don’t share test corpus with developers!

Instrumented Code: Test Hooks

- Code that enables non-invasive testing
- Code remains in shipping product
- May be enabled through
  - Special API
  - Special argument or argument value
  - Registry value or environment variable
- Example: Windows CE IOCTLs
- Risk: silly customers….

Instrumented Code: Diagnostic Compilers

- Creates ‘instrumented’ SUT for testing
  - Profiling – where does the time go?
  - Code coverage – what code was touched?
    - Really evaluates testing, NOT code quality
  - Syntax/coding style – discover bad coding
    - lint, the original syntax checker
  - Complexity
    - Very esoteric, often disputed (religiously)
    - Example: function point counting

Instrumented platforms

- Example: App Verifier
  - Supports 'shims' to instrument standard system calls such as memory allocation
  - Tracks all activity, reports errors such as unreclaimed allocations, multiple frees, use of freed memory, etc.
- Win32 includes 'hooks' for platform instrumentation

Environment Management Tools

- Predictably simulate real-world situations
- MemHog
- DiskHog
- Data Channel Simulator

Test Monkeys

- Generate random input, watch for crash or hang
- Typically, 'hooks' UI through message queue
- Primarily to catch "local minima" in state space (logic "dead ends")
- Useless unless state at time of failure is well preserved!

Finding and Managing Bugs

What is a bug?

- Formally, a "software defect"
- SUT fails to perform to spec
- SUT causes something else to fail
- SUT functions, but does not satisfy usability criteria
- If the SUT works to spec and someone wants it changed, that's a feature request

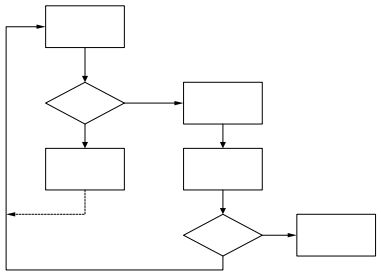
What are the contents of a bug report?

- Repro steps – how did you cause the failure?
- Observed result – what did it do?
- Expected result – what should it have done?
- Any collateral information: return values/output, debugger, etc.
- Environment
  - Test platforms must be reproducible
  - "It doesn't do it on my machine"

Ranking bugs

- Severity
  - Sev 1: crash, hang, data loss
  - Sev 2: blocks feature, no workaround
  - Sev 3: blocks feature, workaround available
  - Sev 4: trivial (e.g. cosmetic)
- Priority
  - Pri 1: Fix immediately
  - Pri 2: Fix before next release outside team
  - Pri 3: Fix before ship
  - Pri 4: Fix if nothing better to do ☺

A Bug's Life



Regression Testing

- Good: rerun the test that failed
  - Or write a test for what you missed
- Better: rerun related tests (e.g. component level)
- Best: rerun all product tests
  - Automation can make this feasible!

Tracking Bugs

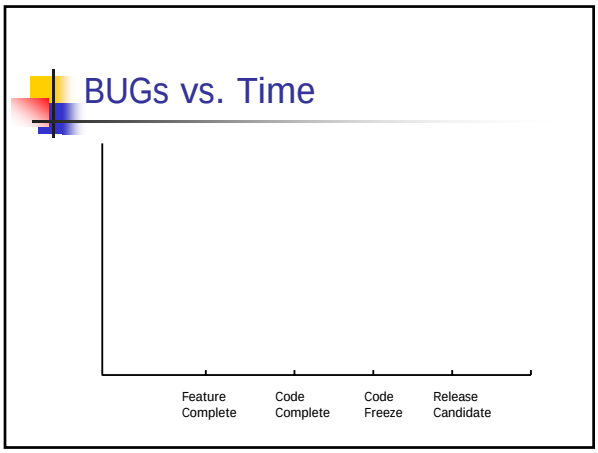
- Raw bug count
  - Slope is useful predictor
- Ratio by ranking
  - How bad are the bugs we're finding?
- Find rate vs. fix rate
  - One step forward, two back?
- Management choices
  - Load balancing
  - Review of development quality

When can I ship?

- Test coverage sufficient
- Bug slope, find vs. fix lead to convergence
- Severity mix is primarily low-sev
- Priority mix is primarily low-pri

Milestones

- Feature complete
  - All features are present
- Code complete
  - Coding is done, except for the bugs
- Code Freeze
  - No more coding
- Release Candidate
  - I think it's ready to ship
- It's out the door


BUGs vs. Time
Feature Complete
Code Complete
Code Freeze
Release Candidate