Introduction: Ian King

1. Manager of Test Development for Smart Personal Objects (SmartWatch)
2. Previous projects at Microsoft:
   1. MSN 1.x online service, Site Server 3.0, TransPoint online service, Speech API 5.0, Windows CE Base OS
3. Student, Professional Masters Program in Computer Science

Testers: A Classic View

What makes a good tester?
1. Analytical
   1. Ask the right questions
   2. Develop experiments to get answers
2. Methodical
   1. Follow experimental procedures precisely
   2. Document observed behaviors, their precursors and environment
3. Brutally honest
   1. You can’t argue with the data

How do test engineers fail?
1. Desire to “make it work”
2. Impartial judge, not “handyman”
3. Trust in opinion or expertise
   1. Trust no one – the truth (data) is in there
4. Failure to follow defined test procedure
   1. How did we get here?
5. Failure to document the data
6. Failure to believe the data
### Testability

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

### Test Categories

1. Functional
   - Does it work? Valid/invalid, error conditions, boundaries
2. Performance
   - How fast/big/high/etc.?
3. Security
   - Access only to those authorized
   - Those authorized can always get access
4. Stress
   - Working stress
   - Breaking stress – how does it fail?
5. Reliability/Availability

### Test Documentation

1. Test Plan
   - Scope of testing
   - Product assumptions
   - Dependencies
   - Tools and Techniques
   - Acceptance criteria
   - Encompasses all categories
2. Test Cases
   - Conditions precedent
   - Actual instructions, step by step
   - Expected results
   - Sorted by category

### Manual Testing

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

### Automated Testing

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

### Tools and Techniques

...
Types of Automation Tools: Record/Playback
1. Record “proper” run through test procedure (inputs and outputs)
2. Play back inputs, compare outputs with recorded values
3. Advantage: requires little expertise
4. Disadvantage: little flexibility - easily invalidated by product change
5. Disadvantage: update requires manual involvement

Types of Automation Tools: Scripted Record/Playback
1. Fundamentally same as simple record/playback
2. Record of inputs/outputs during manual test input is converted to script
3. Advantage: existing tests can be maintained as programs
4. Disadvantage: requires more expertise
5. Disadvantage: fundamental changes can ripple through MANY scripts

Types of Automation Tools: Script Harness
1. Tests are programmed as modules, then run by harness
2. Harness provides control and reporting
3. Advantage: tests can be very flexible
4. Advantage: tests can exercise features similar to customers’ code
5. Disadvantage: requires considerable expertise and abstract process

Types of Automation Tools: Model Based Testing
1. Model is designed from same spec as product
2. Tests are designed to exercise model
3. Advantage: great flexibility
4. Advantage: test cases can be generated algorithmically
5. Disadvantage: requires considerable expertise and high-level abstract process
6. Disadvantage: two opportunities to misinterpret specification/design

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

Instrumented Code: Test Hooks
1. Code that enables non-invasive testing
2. Code remains in shipping product
3. May be enabled through
   - Special API
   - Special argument or argument value
   - Registry value or environment variable
4. Example: Windows CE IOCTLs
5. Risk: silly customers....
**Instrumented Code:** Diagnostic Compilers

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

**Instrumented platforms**

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

**Environment Management Tools**

1. Predictably simulate real-world situations
   1. MemHog
   1. DiskHog
   1. CPU ‘eater’
   1. Data Channel Simulator
1. Reliably reproduce environment
   1. Source control tools
   1. Consistent build environment
   1. Disk imaging tools

**Test Monkeys**

1. Generate random input, watch for crash or hang
   1. Typically, ‘hooks’ UI through message queue
   1. Primarily catches “local minima” in state space (logic “dead ends”)
   1. Useless unless state at time of failure is well preserved!

**What is a bug?**

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

**Finding and Managing Bugs**
What do I do once I find one?

- Bug tracking is a valuable tool
- Ensures the bug isn’t forgotten
- Highlights recurring issues
- Supports formal resolution/regression process
- Provides important product cycle data
- Can support ‘higher level’ metrics, e.g. root cause analysis
- Valuable information for field support

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?
- Collateral information: return values/output, debugger, etc.
- Environment
  - Test platforms must be reproducible
  - “It doesn’t do it on my machine”

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

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 - blocking
  - 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!
To beta, or not to beta

1. Quality bar for beta release: features mostly work if you use them right
   - **Pro:**
     1. Get early customer feedback on design
     2. Real-world workflows find many important bugs
   - **Con:**
     1. Do you have time to incorporate beta feedback?
     2. A beta release takes time and resources

Developer Preview

1. Different quality bar than beta
   - Known defects, even crashing bugs
   - Known conflicts with previous version
   - Setup/uninstall not completed
2. Goals
   - Review of feature set
   - Review of API set by technical consumers

Dogfood

1. “So good, we eat it ourselves”
2. Advantage: real world use patterns
3. Disadvantage: impact on productivity
4. At Microsoft: we model our customers
   1. 60K employees
   2. Broad range of work assignments, software savvy
   3. Wide ranging network (worldwide)

When can I ship?

1. Test coverage is “sufficient”
2. Bug slope, find vs. fix lead to convergence
3. Severity mix is primarily low-sev
4. Priority mix is primarily low-pri