Why Does Testing Take So Long?

Michael Bolton
DevelopSense
http://www.developsense.com
michael@developsense.com

If you're a tester,
you've been asked…

Why is testing taking so long?

or…

When are you going to be done testing?

…and if you *haven't* been asked,
just stick around for a while.

But Hold On A Sec…

Are we ever done testing?

- There are always more conditions to check
- There are always more operations to perform
- There are always more platforms to set up
- There are always more variations of timing to try

Testing Doesn't Stop On Its Own

We decide to stop testing.

Whether for a particular test, a given test cycle, or a ~~test~~ development project, we stop testing *when we decide* to stop testing.

So when might we decide to stop?

1. The "Time's Up!" Heuristic.

When we've exhausted the time that we initially allocated for testing, we stop.

Now that we know more, might we want to allocate time for more investigation?

So when might we decide to stop?

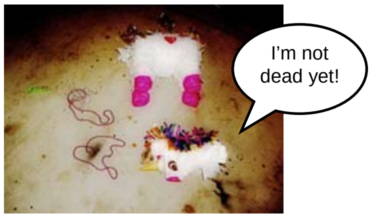
2. The Pinata Heuristic.

We beat on the piñata until the candy starts falling out. The first dramatic problem we find is enough to warrant stopping.

Might there be more interesting candy still inside?

So when might we decide to stop?

3. The Dead Horse Heuristic.



When there are so many bugs in the product that we can't get any useful information, there's no point in continuing to test.

Might we see an even more dramatic problem if we continue?
Might all the problems we see be based on one issue?

So when might we decide to stop?

4. The Mission Accomplished Heuristic.

When we've answered the questions that we originally set out to answer, we stop testing.

Might the answers we've found trigger new questions that we should be asking?

So when might we decide to stop?

5. The Mission Abandoned Heuristic.

When our client tells us to stop testing, we stop.

If we think there are important reasons to continue, does our client know about them?

So when might we decide to stop?

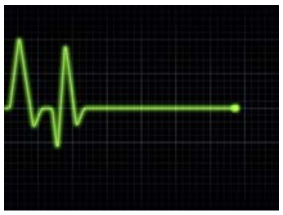
6. The "I Feel Stuck!" Heuristic.

If we're confused, or ill-equipped, or have insufficient information, or are blocked by some bug, we stop.

With some help, could we get unstuck?
Could we proceed on some other path?

So when might we decide to stop?

7. The Flatline Heuristic.



When nothing changes in a load or stress test, no matter how we vary the input, we stop testing.

Are we really varying the input sufficiently?
What might happen if we kept going?

So when might we decide to stop?

8. The Pause That Refreshes.
9. Customary Conclusion
10. Change Tack

INTERESTING
GRAPHICS
COMING
SOON

Actually, all the previous heuristics are founded on this one:

So when might we stop?

11. The Cost vs. Value Heuristic.

?

?

?

?

Are there any interesting questions left worth answering?
Are our tools and practices sufficiently inexpensive?
Are we finding information that is worth the cost of continuing?
Are there alternative choices that might be better for our current context? Or could we really stop testing now?

The fact is…

Testing is done when management decides to ship the product.

The decision to ship a product

IS NOT…
- made by the testers
- governed by rules
- a technical decision
- based on whether **testing** is finished

IS…
- made by the client
- governed by heuristics
- a business decision
- based on whether **development** is finished

Testing doesn't make the decision
Testing helps to inform the decision

Another fact is…

Management decides to ship the product when DEVELOPMENT (not just testing) is done.

So the real question is…

Why is development taking so long?

Isn't that a question for the whole team?

Test Session Effectiveness

- A “perfectly effective” testing session is one entirely dedicated to test design, test execution, and learning
  - a “perfect” session is the exception, not the rule
- Test design and execution tend to contribute to test coverage
  - varied tests tend to provide more coverage than repeated tests
- Setup, bug investigation, and reporting take time away from test design and execution

Modeling Test Effort

Suppose that testing a feature takes two minutes
- this is a highly arbitrary and artificial assumption—that is, it’s *wrong*, but we use it to model an issue and make a point

- Suppose also that it takes an extra eight minutes to investigate and report a bug that we found with a test
  - another stupid, sweeping generalization in service of the point
- In a 90-minute session, we can run 45 feature tests—*as long as we don’t find any bugs*

How Do We Spend Time?
(assuming all tests below are *good* tests)

Module	Bug reporting/investigation (time spent on tests that find bugs)	Test design and execution (time spent on tests that find no bugs)	Number of tests
A (good)	0 minutes (no bugs found)	90 minutes (45 tests)	45
B (okay)	10 minutes (1 bug, 1 test)	80 minutes (40 tests)	41
C (bad)	80 minutes (8 bugs, 8 tests)	10 minutes (5 tests)	13

Investigating and reporting bugs means….

SLOWER TESTING or...

REDUCED COVERAGE ...or both.

• In the first instance, our *coverage* is great—but if we’re being assessed on the number of bugs we’re finding, we look bad.
• In the second instance, coverage looks good, and we found a bug, too.
• In the third instance, we look good because we’re finding and reporting lots of *bugs*—but our *coverage* is suffering severely. A system that rewards us or increases confidence based on the number of bugs we find might mislead us into believing that our product is well tested.

What Happens The Next Day?

(assume 6 minutes per bug fix verification)

Fix verifications	Bug reporting and investigation today	Test design and execution today	New tests today	Total over two days
0 min	0	45	45	90
6 min	10 min (1 new bug)	74 min (37 tests)	38	79
48 min	40 min (4 new bugs)	2 min (1 test)	5	18

Finding bugs today means….

VERIFYING FIXES LATER

…which means….

EVEN SLOWER TESTING or…

EVEN LESS COVERAGE …or both.

•…and note the optimistic assumption that all of our fixed verifications worked, and that we found no new bugs while running them. Has this ever happened for you?

With a more buggy product

- More time is spent on bug investigation and reporting
- More time is spent on fix verification
- Less time is available for coverage

Not only do we do more work…

…we also know less about the system

With a *less* buggy product…

(that is, one that has had some level of testing already)

- We've got *some* bugs out of the way already
- *Those* bugs won't require investigation and reporting
- *Those* bugs won't block our ability to test more deeply

So, programmers, please consider this heuristic:

Test early, and test often!

Test Early and Often!

- Recurrent themes in agile development (note the small A)
  - test-first programming
  - automated unit tests, builds, and continuous integration
  - testability hooks in the code
  - lots of customer involvement
- The ideas are
  - to increase developers' confidence in and commitment to what they're providing ("at least it does *this*")
  - to allow rapid feedback when it *doesn't* do *this*
  - to permit robust refactoring
  - to increase test coverage and/or reduce testing time

Test a product as you build it!

Testing vs. Investigation

- Note that I just gave you a compelling-looking table, using simple measures, but notice that we still don't really know anything about…
  - the quality and relevance of the tests
  - the quality and relevance of the bug reports
  - the skill of the testers in finding and reporting bugs
  - the complexity of the respective modules
  - luck

…but if we *ask better questions*, instead of letting data make our decisions, we're more likely to *learn important things*.

To ask better questions...
To answer them more quickly...
To find bugs more easily...
To fix found bugs faster...
Focus on testability!

We Testers Humbly Request… (from the developers)

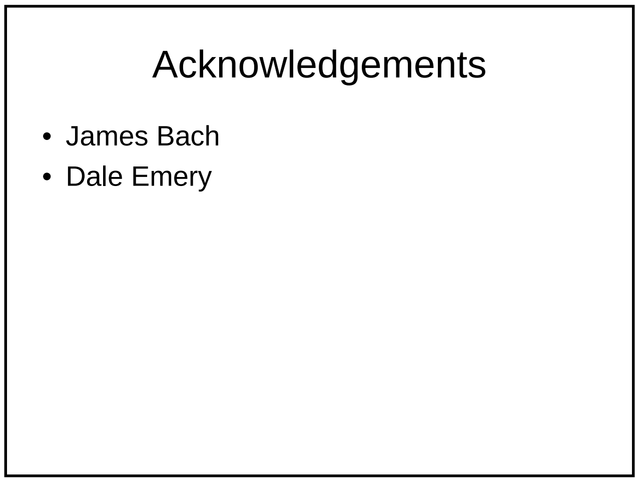
- Developer tests at the unit level
  - use TDD, test-first, automated unit tests, reviews and inspections, step through code in the debugger—whatever increases your own confidence that the code does what you think it does

A less buggy product takes less time to test.

We Testers Humbly Request…
(from the whole team)

- Focus on testability
  - log files
  - scriptable interfaces
  - real-time monitoring capabilities
  - installability and configurability
  - test tools, and help building our own
  - access to "live oracles" and other forms of information

Speed up the decision: "Problem or no problem?"



Acknowledgements

- James Bach
- Dale Emery

Want to know more?

http://www.developsense.com/articles
"How Much Is Enough?"

Michael Bolton
DevelopSense
http://www.developsense.com
michael@developsense.com