IBM Software Group

e business on demand software

Myths and Reality of Iterative Testing

Laura Rose

Myths

http://www.mythweb.com

Prototyping reduces project risk
Testing early increases delivery to market
You can't test without a product
Increased productivity through specialization
Programmers program, testers test
Need stable requirements
Can reduce test time to get back on schedule
Finding and fix all defects creates a quality product
Regression testing assures nothing is broke

Myths

Works as designed
It's an Intermittent Bug
Don't have resources or time to test
Test in a controlled environment
Customer is always right
If we're finding bugs, we doing important testing
Thorough testing mean 100 requirement coverage
Automate, Automate, Automate
Iterative Development doesn't work

Best Practice of Prototyping

Reality: Prototyping often is production code

- Bypassed normal design practices
  - No requirements reviews,
  - No design reviews
  - No code reviews,
  - No unit testing
  - No code inspections
  - No integration/functional testing

Tantalus

Best Practice of Prototyping

Reality: Prototyping often is production code

- Bypassed normal design practices
  - No requirements reviews,
  - No design reviews
  - No [illegible]
  - No unit testing
  - No code inspections
  - No integration/functional testing

→ *Indeterminate foundation.*

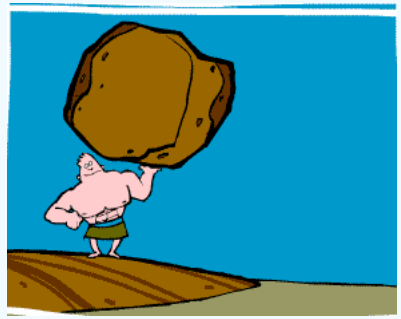
Tantalus

Best Practice of Prototyping

Reality: Prototyping often is production code

- Ways to Avoid
  - All production code needs to follow best practices
    - Assure its stability, reliability, and maintainability
  - Use your prototype to test your development procedures

Hercules



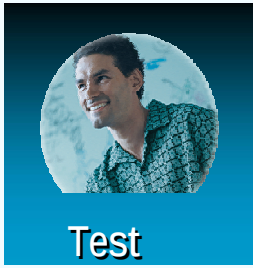
Testing early increases delivery to market

Reality: Testing is *not* the time-consuming activity in a development lifecycle

- Diagnosing and fixing defects are the bottlenecks.
- Ways to Avoid
  - Avoid creating them
    - Take requirements seriously
    - Test requirement to remove ambiguity
  - Move the detection of problems closer to their creation
    - Code inspections, code reviews, static analysis, unit testing

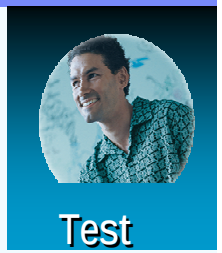
Sisyphus

Requirement Review Checklist



- **List of attributes to test against:**
  - **Complete.** Is anything missing or forgotten? Is it thorough? Does it include everything necessary to make it stand alone?
  - **Accurate:** Is the proposed solution correct? Does it properly define the goal? Are there any errors?
  - **Precise, Unambiguous, and Clear.** Is the description exact and not vague? Is there a single interpretation? Is it easy to read and understand?
  - **Consistent.** Is the description of the feature written so that it doesn't conflict with or other items in the specification?
  - **Relevant.** Is the statement necessary to specify the feature? Is there extra information that should be left out? Is the feature traceable to an original customer need?
  - **Feasible.** Can the feature be implemented with the available personnel, tools, and resources within the specified budget and schedule?
  - **Code-free.** Does the specification stick with defining the product and not the underlying software design, architecture and code?
  - **Testable.** Can the feature be tested? Is enough information provided that a tester could create tests to verify its operation?

Software Testing by Ron Patton.



Problem Words in a specification:

- **Always, Every, All, None, Never:** If you see words such as these that denote something as certain and absolute, make sure that it is indeed certain. Think of cases that violate them, when reviewing the spec.
- **Certainly, Therefore, Clearly, Obviously, Ordinarily, Customarily, Most, Mostly**. These words tend to persuade you into accepting something as a given. Don't fall into the trap.
- **Some, sometimes, Often Usually, Ordinarily, Customarily, Most, Mostly:** There words are too vague. It's impossible to test a feature that operates "sometimes".
- **Etc, And So Forth, And So On, Such As**. Lists that finish with these words aren't testable. There needs to be no confusion as to how the series is generated and what appears next in the list.
- **Good, Fast, Cheap, Efficient, Small, Stable.** These are unquantifiable terms. They aren't testable. If they appear in a specification, they must be further defined to explain exactly what they mean
- **Handled, Processed, Rejected, Skipped, Eliminated.** These terms can hide large amounts of functionality and need to be specified.
- **If… Then (but missing Else).** Look for statements that have "If…Then" clauses but don't have a matching "else". Ask yourself what will happen if the "if" doesn't happen.

Software Testing by Ron Patton.

You can't test if you don't have a product to test

Reality: Testing isn't limited to code

- Iterative testing isn't limited to testing code
  - the architecture and development framework,
  - the design, and the customer usage flows,
  - the requirements, and the test plans,
  - the deployment structure,
  - the support and service techniques,
  - the diagnostic and troubleshooting methods,
  - the procedures you follow to produce the product

Zeus

- Ways to Avoid
  - Include Quality Acceptance Criteria for all deliverables

Specialization

Reality: Huge Risk having just one developer maintain and develop an area

- no one else can maintain and understand that feature
- Creates bottlenecks and delays

- Ways to avoid
  - Increase your pool of resources
    - Pair programming
    - pair testing,
    - code reviews,
    - design reviews

Achilles

Programmers program, testers test

Reality: The primary task of everyone on the team is to produce a product that customers will value

- Ways to Avoid:
  - Be in the Present to avoid misplaced attention
    - Emailing others during code inspections?
    - Coding during requirement review meetings?
    - Walking away from a test that's running
      - Miss performance issues, time delays and pop-up errors

Midas

Stable requirements

Reality: Actual goal is to produce a product that customers will value

- Ways to Avoid
  - Frequent customer interaction
  - Expect modification and course changes

Atlas

Reduce test time to make schedule

Reality: Need to increase test time

- Delay in code delivery means we've underestimated the project complexity,
- Probably we underestimated test coverage and test effort
- Slashing test time is a very poor decision

- Ways to includes test time without affecting overall schedule
  - Testing earlier and continuously
  - Quality of the product determines the amount of testing

Titans

Fix all defects to create high quality product

Reality: Fixing defects isn't an added-value activity

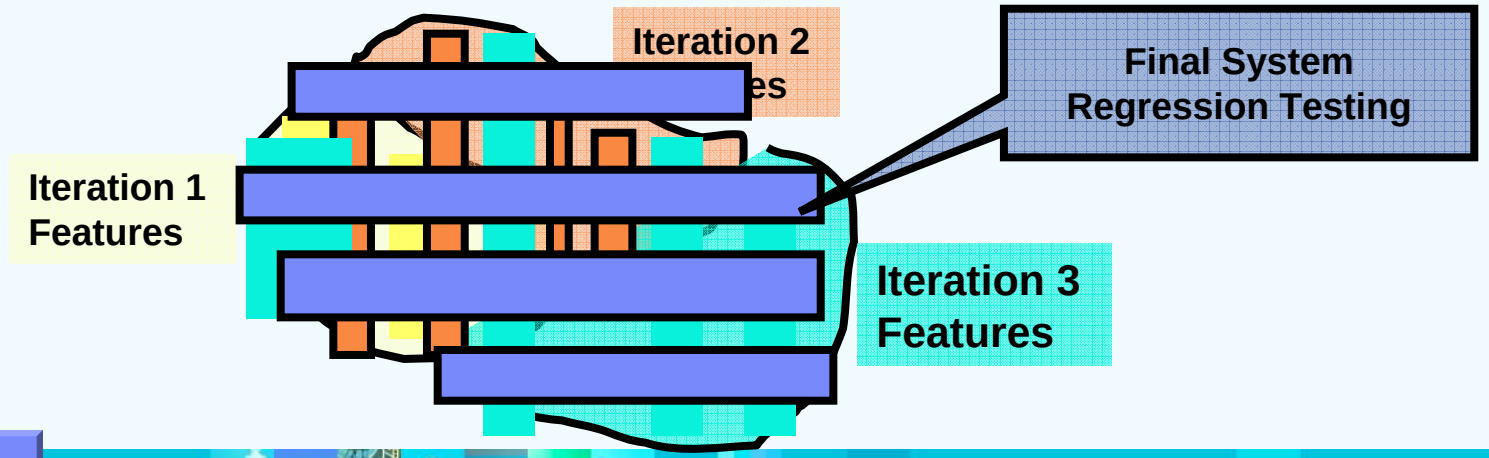
- Only 34% of all features are used by customers
  - Finding and fixing defects related to these features does not add customer value
- Ways to Avoid
  - Greenthreads or end-to-end scenario testing
  - Customer Interaction programs

Procrustes

Regression testing assures nothing is broke

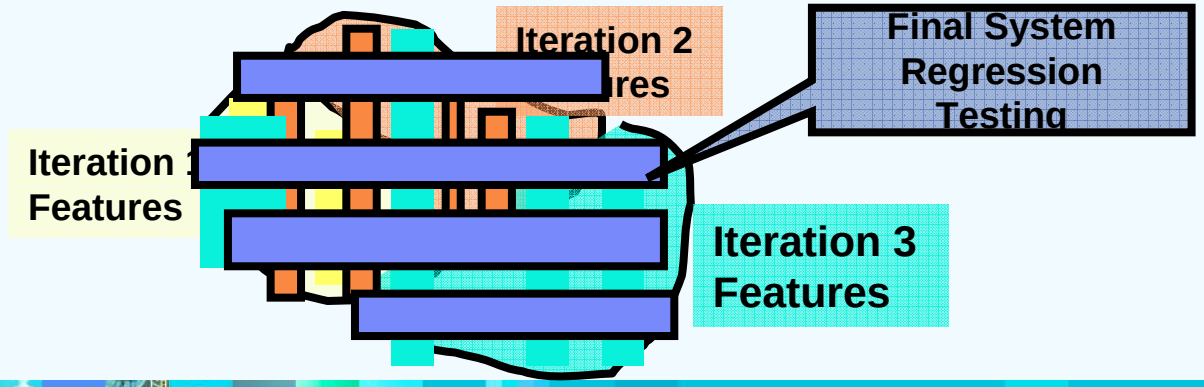
Reality: Regression testing doesn't assure that you haven't broken anything.

- Reality
  - Regression testing covers items you've already tested
  - You can't test everything
  - Therefore, regression testing can't assure nothing has broken
- Test what makes sense in each phase and iteration



Regression testing assures nothing is broke

- Iterative regression testing
  - Implement Equivalence Class Testing in your regression suite
  - Intelligently select the most appropriate, high profile, and frequently encounter customer coverage paths
  - Sample lower risk areas across a set of iterations
  - Implement automation into your development organization.
  - Increase your confidence level in components by running them through Code Profiling and Static Analysis tools.



Works as Designed

Reality: Customer doesn't care if it's working as you coded it.

- Sometimes we just know too much
  - Explain away the problem
- Sometimes we don't know enough
  - Intermittent bug
- Customer is still negatively affected by the bug

- Avoid
  - Better diagnostic and serviceability routines
  - Self-monitoring and self-correcting routines

Prometheus

When we encounter this response....	... we should ...
We already know this isn't right.	Identify explicit things that need to be fixed by the release date and what needs to be fixed in a subsequent release. Identify owners and deadlines.
It's on our list for future consideration (with no scheduled date).	Realize that if it has a "future consideration" tag (or similar), it's not real. Work with tech support, field consultants, and customers to illustrate the importance of the bug or feature. Once the team is assured of the customer value, schedule a release date and owner.
It's in someone else's code outside our department.	Create a SWAT team that includes "outside department" staff in addition to development staff. Schedule a release date and owner of the change.
Customer or pilot error.	Study the documentation or usage flow. Customer errors are often the result of some unclear and unintuitive step. Explicitly identify the change that needs to occur, schedule it, and set an owner.
It's working as designed.	Request a full review of the design in the workflow perspective. Walk through how the customer or role will play out from beginning to end.
It's working this way because XXX.	Realize that if XXX isn't "because the customer wants it this way," then XXX is irrelevant. Eliminate that reason from the discussion and move on to the next point.

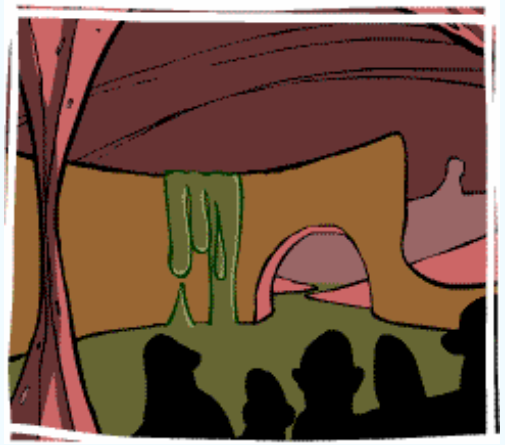
We don't have time to test everything

Reality: We don't need to test everything.

- Can't test everything
- Can't test forever
- Need a pragmatic approach

- Avoid
  - Customers' business focus
  - System testing at an external customer lab
  - Internal deployment of your tools
  - Benefits
    - Complex real-world environments
    - no additional system admin.

River Styx



Test in a controlled environment

Reality: The more the test environment resembles the final production environment, the more reliable the testing

- Customer's environment is 100% controlled
  - do all your testing in a controlled environment
- Avoid
  - Customer reviews of business usage
  - System testing at customer sites

Amazon.

The customer is always right

Reality: You can't make everyone happy with one release

All customers are equally important

- Avoid
  - Maybe it's not the right customer….
  - Target specific customers
  - Focus on few release-defining features

Athena.

If we're finding a lot of bugs, we are doing important testing

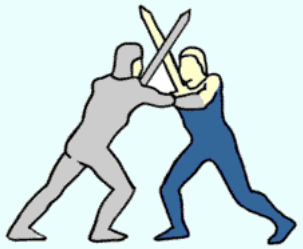
Reality Finding a lot of bugs means the product has a lot of bugs.

- Finding many bugs doesn't tell us
  - quality of the test coverage,
  - the severity of the bugs,
  - the frequency with which customers will actually hit them
  - If the workflows in a product isn’t actually used
    - they don't need to work
- Avoid
  - Incorporating both risk- and customer-based
  - Incorporate analysis into your test plan solution

Thorough testing means testing 100% of the requirements

Reality: Testing 100% isn't enough

- Test for what's missing
- Avoid
  - Get customers involved



Achilles and Hector, locked in battle. These two were the greatest heroes on their respective sides in the Trojan War.

Automate, Automate, Automate

Reality: The more the test environment resembles the final production environment, the more reliable the testing

- If the customer uses your product in an automated environment, Automate, Automate, Automate…..
- Otherwise -- Automate judicially and with ROI in mind
- Combine with
  - ad hoc,
  - exploratory,
  - customer scenario testing
  - Automate setup or breakdown of clean environments
  - Automate security maintenance,

Criteria for Automation and/or Regression Test

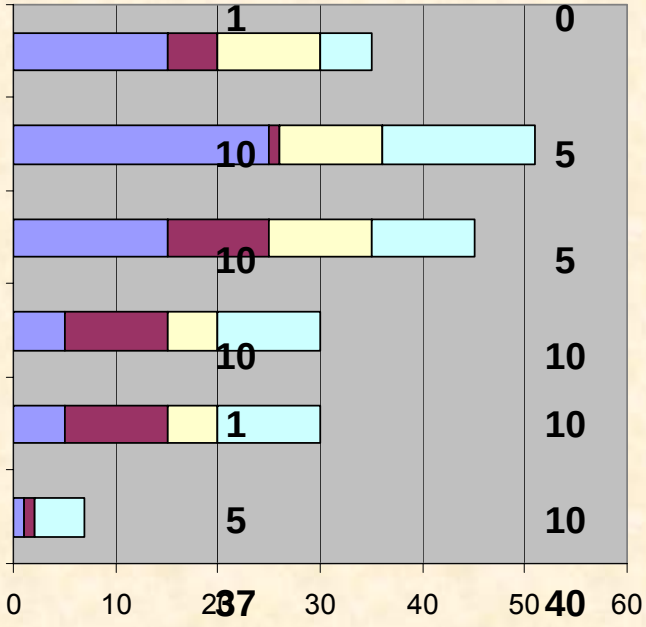
- A test that is not planned to be run more than 3 times – is not a good candidate.
- A test that can be run on multiple platforms and environment
- A test that will be part of the "sanity regression suite“
- Automated setup and clean exit steps
- A test that resides in a "high traffic/ high code change" area
- A test that resides in a code area with multiple function points (high integration point) with more than 1 developer touching the same code.
- A test that is in an area that has several defects logged against it
  - This includes areas that have integration points to areas that have several defects logged against them.

Reminder:

- Regression suites and retest emphasis can (and should) change depending upon the areas of change in the product. Test Harness should be flexible enough to run specific areas upon request (versus everything - all the time).

Best Area to Spend Automation Resources

	#Cycles	Number of Users	Stability	Ease of Implmt	Total
Construction cycle on new design	1	1	0	5	7
Regression testing in new product, directly after construction period	5	10	5	10	30
Regression testing in transition of new product	5	10	5	10	30
Regression testing in maintenance mode	15	10	10	10	45
Build Verification Testing	25	1	10	15	51
Smoke Tests on G11N platforms	15	5	10	5	35
Total	66	37	40	55	



Iterative Development doesn't work

Reality: We don't give it a chance to succeed.

- No experience in the new approach
- No gained trust.
- We panic back into old habits

- Ways to Avoid
  - Incremental success criteria at each iteration
  - Continuous testing and monitoring results against exit criteria
  - Deliver mid-stream to customers

Pan (panic)

Summary –

- Myths are seductive
- Fall prey when we're understaffed and under pressure
- More assumptions, less open to the unexpected

Iterative testing is never being satisfied with the first right answer.

Rational Edge Magazine: Myths and Realities of Iterative Testing