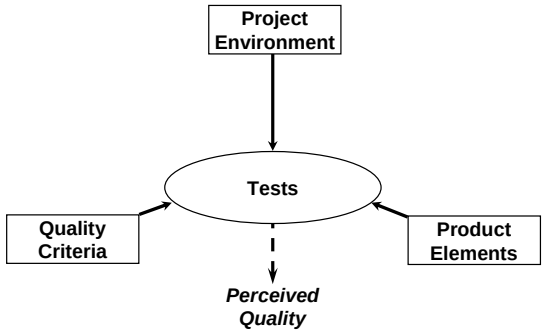
Rapid Testing

Rapid testing is a *mind-set*
and a *skill-set* of testing
focused on how to do testing
more quickly,
less expensively,
with *excellent results*.

This is a general testing methodology. It adapts to any kind of project or product.

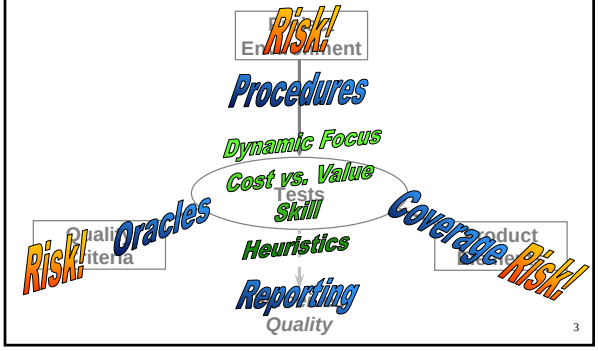
1

A Heuristic Test Strategy Model



2

A Heuristic Test Strategy Model

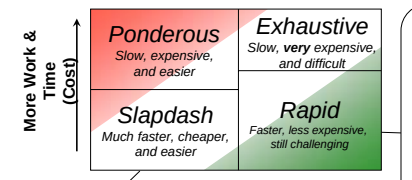


3

How does Rapid Testing compare with other kinds of testing?

When testing is turned into an elaborate set of rote tasks, it becomes ponderous without really being thorough.

Management likes to talk about exhaustive testing, but they don't want to fund it and they don't know how to do it.



Rapid testing may not be exhaustive, but it is thorough enough and quick enough. It's less work than ponderous testing. It might be less work than slapdash testing.

It fulfills the mission of testing.

You can always test quickly... But it might be poor testing.

Better Thinking & Better Testing (Value)

4

Excellent Rapid Technical Work Begins with You

When the ball comes to you…

Do you know you have the ball?

Can you receive the pass?

Do you know your options?

Do you know what your role and mission is?

Is your equipment ready?

Can you read the situation on the field?

Do you know where your teammates are?

Are you aware of the criticality of the situation?

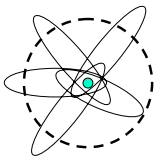
Can you let your teammates help you?

Are you ready to act, *right now*?

5

…but you don't have to be great at *everything*.

- **Rapid test teams are about diverse talents cooperating**
  - We call this the *elliptical team,* as opposed to the team of perfect circles.
  - Some important dimensions to vary:
    - Technical skill
    - Domain expertise
    - Temperament (e.g. introvert vs. extrovert)
    - Testing experience
    - Project experience
    - Industry experience
    - Product knowledge
    - Educational background
    - Writing skill
  - Diversity makes exploration far more powerful
  - Your team is more powerful because of your unique individual contribution



6

No, not the database

ORACLE

Not the database

An oracle is...

a principle or mechanism
by which
we recognize a problem

But wait…

How can we be
SURE
that we're seeing a problem?

WE CAN'T.

Certainty isn't available.

But we DO have *heuristics*

Heuristics are fallible, "fast and frugal" methods of solving problems, making decisions, or accomplishing tasks.

"The engineering method is the use of *heuristics* to cause the best change in a poorly understood situation within the available resources."
Billy Vaughan Koen
Discussion of the Method

Heuristics: Generating Solutions Quickly

- **adjective:**
  "serving to discover or learn."
- **noun:**
  "A **fallible** method for **solving a problem** or **making a decision**."

"Heuristic reasoning is not regarded as final and strict but as provisional and plausible only, whose purpose is to discover the solution to the present problem."
- George Polya, *How to Solve It*

Oracles

An oracle is a **heuristic** principle or mechanism by which you recognize a problem.

"It works!"

really means...

"...it appeared at least once to meet some requirement to some degree."
"...uh, when I ran it."
"...on my machine."

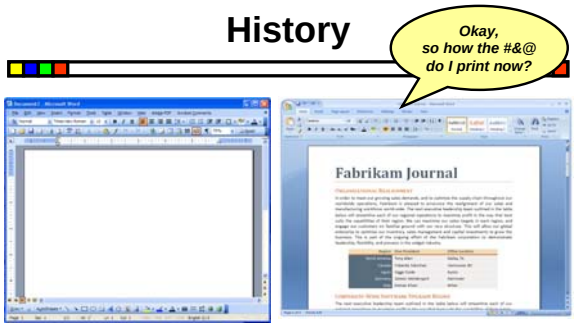
Oracles Link Observations with Problems

Without an oracle you **cannot** recognize a problem

and conversely...

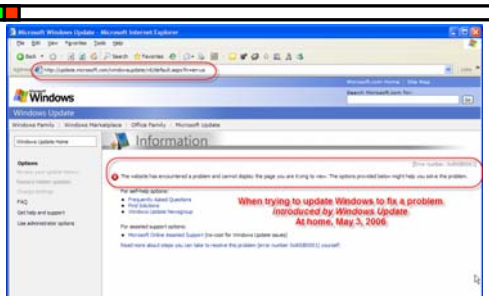
If you think you see a problem, you **must** be using an oracle… so what is it?

History



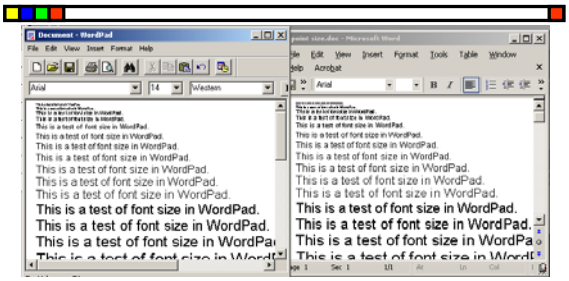
If a product is inconsistent with previous versions of itself, we suspect that there might be a problem.

Image



If a product is inconsistent with an image that the company wants to project, we suspect a problem.

Comparable Products



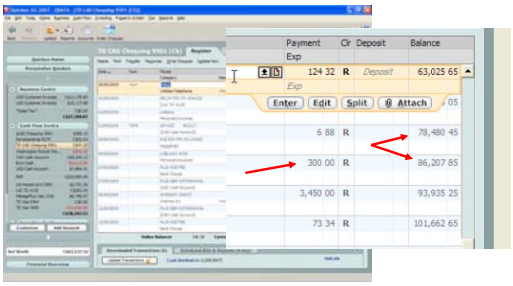
WordPad Word

When a product seems inconsistent with a comparable product, we suspect that there might be a problem.

Claims

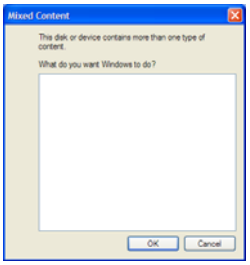
When a product is inconsistent with claims that important people make about it, we suspect a problem.

User Expectations



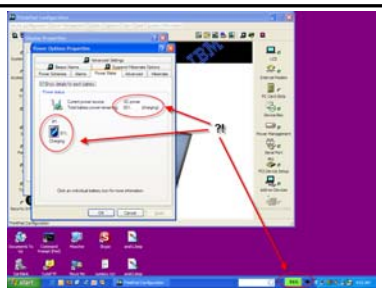
When a product is inconsistent with expectations that a reasonable user might have, we suspect a problem.

Purpose



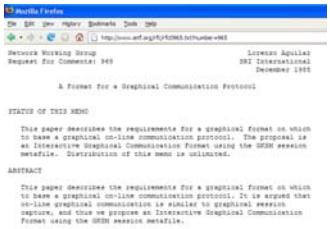
When a product is inconsistent with its designers' explicit or implicit purposes, we suspect a problem.

Product



When a product is inconsistent internally—as when it contradicts itself—we suspect a problem.

Statutes and Standards



When a product is inconsistent with laws or widely accepted standards, we suspect a problem.

We like consistency when…

- the present version of the system *is consistent* with **past versions** of itself.
- the system *is consistent* with **an image** that the organization wants to project.
- the system *is consistent* with **comparable systems**.
- the system *is consistent* with **what important people say** it's supposed to be.
- the system *is consistent* with **what users seem to want**.
- each element of the system *is consistent* with comparable **elements in the same system**.
- the system *is consistent* with implicit and explicit purposes.
- the system *is consistent* with relevant laws or standards.

unless it's a problem.

- We like it when the system ***is not consistent*** with patterns of familiar problems.

But...

- All of the consistency oracles are heuristic.

Can work.

Might fail.

All Oracles Are Heuristic

An oracle doesn't tell you that there IS a problem. An oracle tells you that you *might be seeing a problem.*

Consistency heuristics rely on the quality of your models of the product and its context.

Rely solely on documented, anticipated sources of oracles, and your testing will likely be slower and weaker.

Train your mind in *patterns* of oracles and your testing will likely be faster and your coverage better.

How Do I Keep Track? HICCUPPS!

- History
- Image
- Comparable Products
- Claims
- User Expectations
- Purpose
- Product
- Statutes

…plus for "Familiar Problems", add that inconsistent F!

Remember…

For skilled testers, good testing isn't just about pass vs. fail.

For skilled testers, testing is about ***problem vs. no problem.***

What IS Coverage?

Coverage is "how much of the product we have tested."

It's the extent to which we have traveled over *some map* of the product.

...but what does it mean to "map" a product?

Talking about coverage means talking about

models

Models

- **A model is a heuristic idea, activity, or object…**

  such as an *idea in your mind*, a *diagram*, a *list of words*, a *spreadsheet*, a *person*, a *toy*, an *equation*, a *demonstration*, or a *program*

- **…that represents (literally, *re*-presents) another idea, activity, or object…**

  such as something complex that you need to work with or study

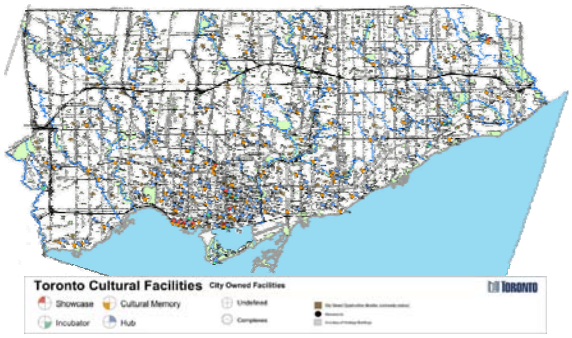
- **…whereby understanding something about the model may help you to understand or manipulate the thing that it represents.**

- A *map* is a model that helps to navigate across a terrain.
- *2+2=4* is a model for adding two apples to a basket that already has two apples.
- *Atmospheric models* help predict where hurricanes will go.
- A *fashion model* helps understand how clothing would look on actual humans.
- *Your beliefs about what you test are a model of what you test.*

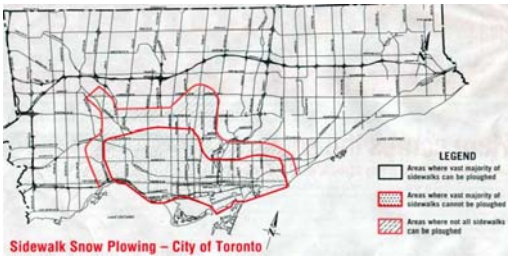
A Map of the Toronto Subway

Here's Another One

A Map of Toronto's Cultural Facilities

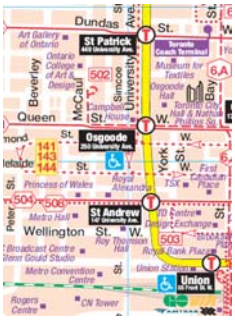


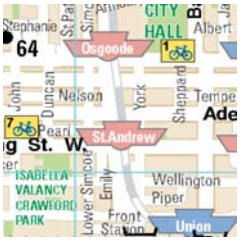
So You Want Your Sidewalk Plowed?



A Bike Ride?

What Is Covered Incidentally?





Different Maps Show Different Things

- The information that we care about may be incidental to the "purpose" of the map

It depends on what we're looking for...
It depends on where we're looking...
It depends on what it means to "cover" the map!

There are as many kinds of test coverage as there are ways to model the system.

Structure
Business Risk
Time
Functions
Platform
Technical Risk
Data
Operations

And each could be...

Intentional or... Accidental

One Way to Model Coverage: Product Elements (with Quality Criteria)

SFDPOT -- San Francisco Depot

Product Elements

- Structure
- Function
- Data
- Platform
- Operations
- Time

Quality Criteria

Capability, Reliability, Usability, Security, Scalability, Performance, Installability, Compatibility, Supportability, Testability, Maintainability, Portability, Localizability

To test a *very simple* product meticulously, *part* of a complex product meticulously, or to maximize test *integrity*…

Focus!

1. Start the test from a *known* (clean) state.
2. Prefer *simple, deterministic* actions.
3. Trace test steps to a *specified model*.
4. Follow *established and consistent* lab procedures.
5. Make *specific* predictions, observations and records.
6. Make it *easy to reproduce* (automation may help).

40

To find *unexpected problems*, *elusive problems* that occur in sustained field use, or more problems *quickly* in a complex product…

That's a PowerPoint bug!

De-Focus!

1. Start from *different states* (not necessarily clean).
2. Prefer *complex, challenging* actions.
3. Generate tests from a *variety* of models.
4. *Question* your lab procedures and tools.
5. Try to *see everything* with open expectations.
6. Make the test *hard to pass*, instead of easy to reproduce.

41

General Focusing Heuristics

- use test-first approach or unit testing for better *code* coverage
- work from prepared test coverage outlines and risk lists
- use diagrams, state models, and the like, and cover them
- apply specific test techniques to address particular coverage areas
- make careful observations and match to expectations

Follow your procedures.

To do this *more rapidly*, make *preparation* and *artifacts* fast and frugal: leverage existing materials and avoid repeating yourself. Emphasize doing; relax planning. You'll make discoveries along the way!

General Defocusing Heuristics

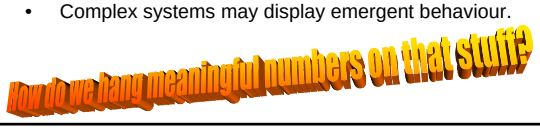
- diversify your models; intentional coverage in one area can lead to unintentional coverage in other areas—this is a Good Thing
- diversify your test techniques
- be alert to problems other than the ones that you're actively looking for
- welcome and embrace distraction
- do some testing that is *not* oriented towards a specific risk
- use high-volume, randomized automated tests

Extent of Coverage

- Smoke and sanity
  - Can this thing even be tested at all?
- Common, core, and critical
  - Can this thing do the things it *must* do?
  - Does it handle happy paths and regular input?
  - *Can* it work?
- Complex, harsh, extreme and exceptional
  - Will this thing handle challenging tests, complex data flows, and malformed input, etc.?
  - *Will* it work?

What About Quantifying Coverage Overall?

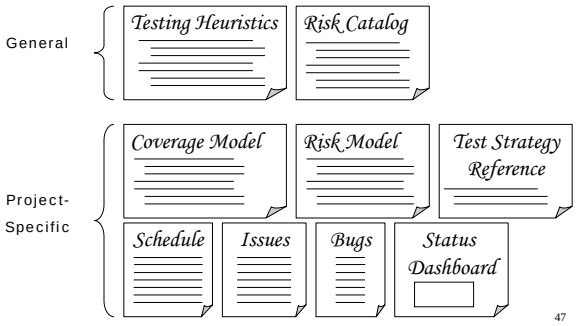
- A nice idea, but we don't know how to do it in a way that is consistent with *basic* measurement theory
  - If we describe coverage by counting test cases, we're committing reification error.
  - If we use percentages to quantify coverage, we need to establish what 100% looks like.
    - But we might do that with respect to *some specific* models.
  - Complex systems may display emergent behaviour.



How Might We Organize, Record, and Report Coverage?

- automated tools (e.g. profilers, coverage tools)
- annotated diagrams (as shown in earlier slides)
- coverage matrices
- bug taxonomies
- Michael Hunter's You Are Not Done Yet list
- James Bach's Heuristic Test Strategy Model
  - described at www.satisfice.com
  - articles about it at www.developsense.com
- Mike Kelly's MCOASTER model
- coverage outlines and risk lists
- session-based test management
  - http://www.satisfice.com/sbtm

What Does Rapid Testing Look Like?
Concise Documentation Minimizes Waste



Rapid Testing Documentation

- Recognize
  - a requirements *document* is not *the requirements*
  - a test plan *document* is not *a test plan*
  - a test *script* is not a *test*
  - doing, rather than planning, produces results
- Determine where your documentation is on the continuum: product or tool?
  - Keep your *tools* sharp and lightweight
  - Obtain consensus from others as to what's necessary and what's excess in *products*
- Ask whether *reporting* test results takes priority over *obtaining* test results
  - note that in some contexts, it might
- Eliminate unnecessary clerical work

What IS Exploratory Testing?

- Simultaneous test design, test execution, and learning.
  - James Bach, 1995

But maybe it would be a good idea to underscore why that's important…

What IS Exploratory Testing?

- I follow (and to some degree contributed to) Kaner's definition, which was refined over several peer conferences through 2007:

Exploratory software testing is…

- a style of software testing
- that emphasizes the personal freedom and responsibility
- of the individual tester
- to continually optimize the value of his or her work
- by treating test design, test execution, test result interpretation, and test-related learning
- as mutually supportive activities
- that run in parallel
- throughout the project.

So maybe it would be a good idea to keep it brief most of the time…

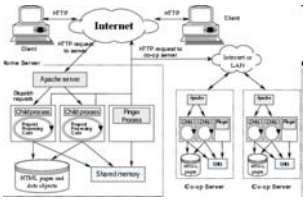
See Kaner, "Exploratory Testing After 23 Years", www.kaner.com/pdfs/ETat23.pdf

Why Exploratory Approaches?

- Systems are far more than collections of functions
- Systems typically depend upon and interact with many external systems

Why Exploratory Approaches?

- Systems are too complex for individuals to comprehend and describe
- Products evolve rapidly in ways that cannot be anticipated



In the future, developers will likely do more verification and validation at the unit level than they have done before.

Testers must explore, discover, investigate, and learn about the *system*.

Why Exploratory Approaches?

- Developers are using tools and frameworks that make programming more productive, but that may manifest more emergent behaviour.
- Developers are increasingly adopting unit testing and test-driven development.
- The traditional focus is on verification, validation, and confirmation.

The new focus must be on exploration, discovery, investigation, and learning.

Why Exploratory Approaches?

- We don't have time to waste
- preparing wastefully elaborate written plans
- for complex products
- built from many parts
- and interacting with many systems
- (many of which we don't understand…
- or control)
- where everything is changing over time
- and there's *so much learning* to be done
- and the *result*, not the plan, is paramount.

Exploratory Testing

The way we practice and teach it, exploratory testing…

- **IS NOT** "random testing" (or sloppy, or slapdash testing)
- **IS NOT** "unstructured testing"
- **IS NOT** procedurally structured
- **IS NOT** unteachable
- **IS NOT** unmanageable
- **IS NOT** scripted
- **IS NOT** a technique
- **IS** "ad hoc", in the dictionary sense, "to the purpose"
- **IS** structured and rigorous
- **IS** cognitively structured
- **IS** highly teachable
- **IS** highly manageable
- **IS** chartered
- **IS** an approach

What you do next is governed by what you're learning

Contrasting Approaches

Scripted Testing	Exploratory Testing
• Is directed from elsewhere	• Is directed from within
• Is determined in advance	• Is determined in the moment
• Is about confirmation	• Is about investigation
• Is about controlling tests	• Is about improving test design
• Emphasizes predictability	• Emphasizes adaptability
• Emphasizes decidability	• Emphasizes learning
• Like making a speech	• Like having a conversation
• Like playing from a score	• Like playing in a jam session

The tester's mind is in control, not the script.

Exploratory Testing IS Structured

- Exploratory testing, as we teach it, is a structured process conducted by a skilled tester, or by lesser skilled testers or users working under supervision.
- The structure of ET comes from many sources:
  - Test design heuristics
  - Chartering
  - Time boxing
  - Perceived product risks
  - The nature of specific tests
  - The structure of the product being tested
  - The process of learning the product
  - Development activities
  - Constraints and resources afforded by the project
  - The skills, talents, and interests of the tester
  - The overall mission of testing

Not *procedurally* structured, but *cognitively* structured.

In other words, it's not "random", but systematic.

ET is a Structured Process

In excellent exploratory testing, one structure tends to dominate all the others:

The Testing Story

Exploratory testers construct a compelling story of their testing. It is this story that gives ET a backbone.

58

To test is to compose, edit, narrate, and justify *two* stories.

You must tell a story about the product…

…about how it failed, and how it *might* fail...

…in ways that matter to your various clients.

But you must also tell a story about your testing...

…how you configured, operated and observed it…

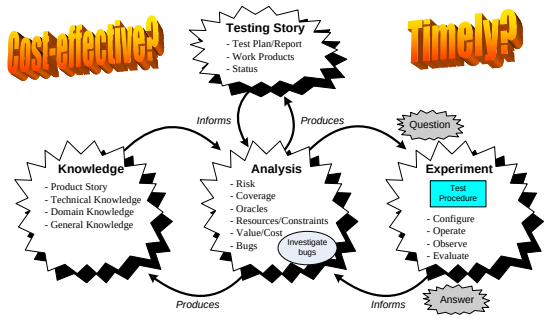
…about what you haven't tested, yet…

…or won't test, at all…

…and about why what you did was good enough.

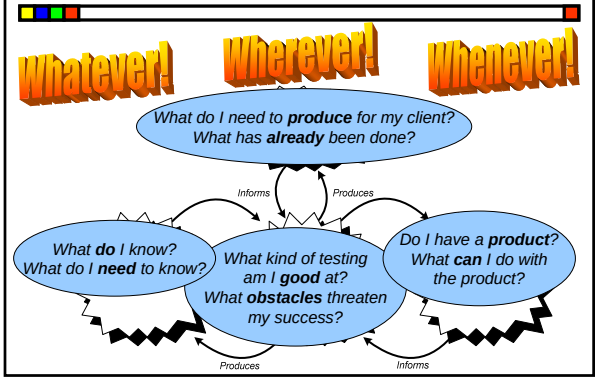
59

The Process of Test Design



How to Start?

Pick a Useful, Fun, or Easy Starting Point



Cost as a Simplifying Factor

Try quick tests as well as careful tests

A *quick test* is a cheap test that has some value but requires little preparation, knowledge, or time to perform.

- Happy Path
- Tour the Product
  - *Sample Data*
  - *Variables*
  - *Files*
  - *Complexity*
  - *Menus & Windows*
  - *Keyboard & Mouse*
- Interruptions
- Undermining
- Adjustments
- Dog Piling
- Continuous Use
- Feature Interactions
- Click on Help

62

Cost as a Simplifying Factor

Try quick tests as well as careful tests

A *quick test* is a cheap test that has some value but requires little preparation, knowledge, or time to perform.

- Input Constraint Attack
- Click Frenzy
- Shoe Test
- Blink Test
- Error Message Hangover
- Resource Starvation
- Multiple Instances
- Crazy Configs
- Cheap Tools

63

Touring the Product: Mike Kelly's FCC CUTS VIDS

- Feature tour
- Complexity tour
- Claims tour
- Configuration tour
- User tour
- Testability tour
- Scenario tour
- Variability tour
- Interoperability tour
- Data tour
- Structure tour

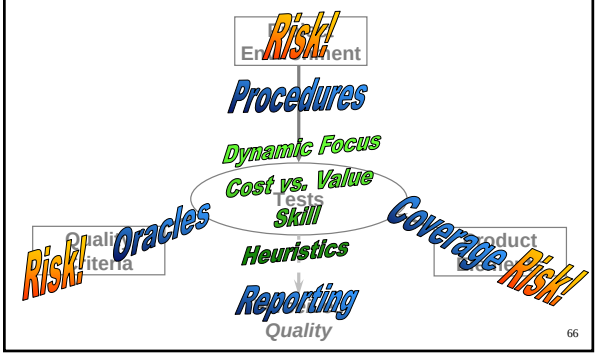
Create your own list!

The Themes of Rapid Testing

- Put the **tester's mind** at the center of testing.
- Learn to **deal with complexity** and ambiguity.
- Learn to **tell a compelling testing story.**
- Develop **testing skills** through practice, not just talk.
- **Use heuristics** to guide and structure your process.
- **Be a service** to the project community, not an obstacle.
- **Consider cost vs. value** in all your testing activity.
- **Diversify** your team and your tactics.
- Dynamically **manage the focus** of your work.
- Your **context should drive your choices**, both of which evolve over time.

65

A Heuristic Test Strategy Model



66