

Black Box Software Testing

Fall 2006

Exploratory Testing

Cem Kaner, J.D., Ph.D.

Professor of Software Engineering

Florida Institute of Technology

Copyright (c) Cem Kaner 2006

This work is licensed under the Creative Commons Attribution-ShareAlike License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/2.0/> or send a letter to Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.

These notes are partially based on research that was supported by NSF Grant EIA-0113539 ITR/SY+PE: "Improving the Education of Software Testers." Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Much of the material in these slides was provided or inspired by James Bach, Michael Bolton, Jonathan Bach, Rebecca Fiedler, and Mike Kelly.

Overview

I coined the phrase “exploratory testing” in 1983 to describe the practice of some of the best testers in Silicon Valley. Naturally, the concept has evolved (and diverged) over the years.

ET has been a lightning rod for criticism, some of it justified. This lecture considers where I think we are now, controversies, misunderstandings and valid concerns surrounding ET.

ET has become fashionable. To accommodate non-students who have asked for access to the ET videos, I cross-references to some points from prior videos.


If you are taking the full BBST course, I trust/hope that these cross-references will provide a helpful review.

Some key points

- ET is an approach to testing, not a technique
 - You can use any test technique in an exploratory way or a scripted way
 - You can work in an exploratory way at any point in testing
- Effective testing requires the application of knowledge and skill
 - This is more obvious (but maybe not more necessary) in the exploratory case
 - Training someone to be an explorer probably involves greater emphasis on higher levels of knowledge

Outline

- An opening contrast: Scripted testing
- The nature of testing
- The other side of the contrast:
Exploration
- Exploratory testing: Learning
- Exploratory testing: Design
- Exploratory testing: Execution
- Exploratory testing: Interpretation
- Exploratory testing after 23 years



*An opening
contrast:
Scripted testing*

Scripted testing

A script specifies

- the test operations
- the expected results
- the comparisons the human or machine should make

These comparison points are

- useful, ***but fallible and incomplete***, criteria for deciding whether the program passed or failed the test

Scripts can control

- manual testing by humans
- automated test execution or comparison by machine

Key benefits of scripts

Scripts require a big investment. What do we get back?

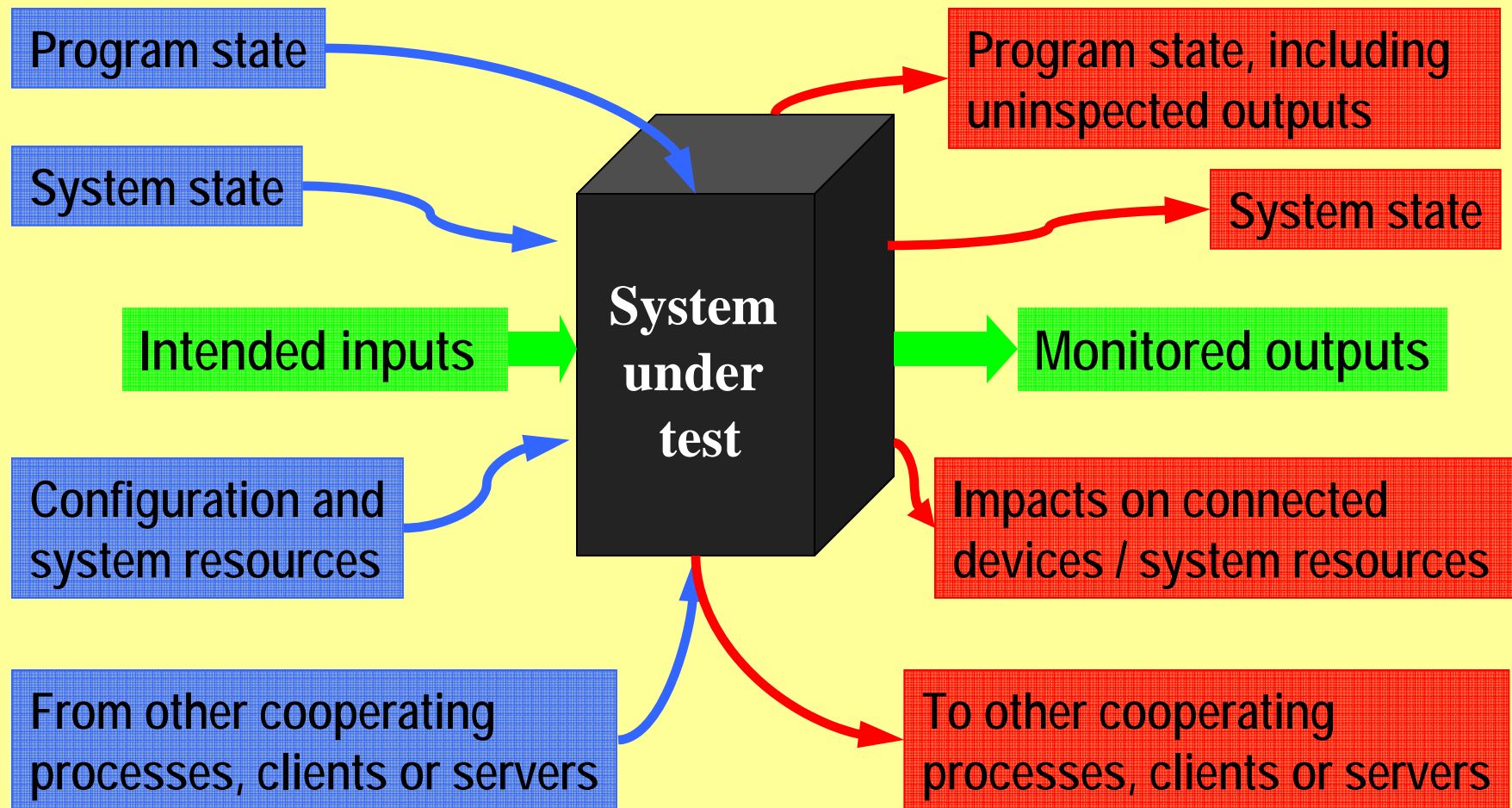
The scripting process provides *opportunities* to achieve several key benefits:

- Careful thinking about the design of each test, optimizing it for its most important attributes (power, credibility, whatever)
- Review by other stakeholders
- Reusability
- Known comprehensiveness of the set of tests
- If we consider the set sufficiently comprehensive, we can calculate as a metric the percentage completed of these tests.

Reminder from the oracle lecture:

Programs fail in many ways

Based on notes from Doug Hoffman



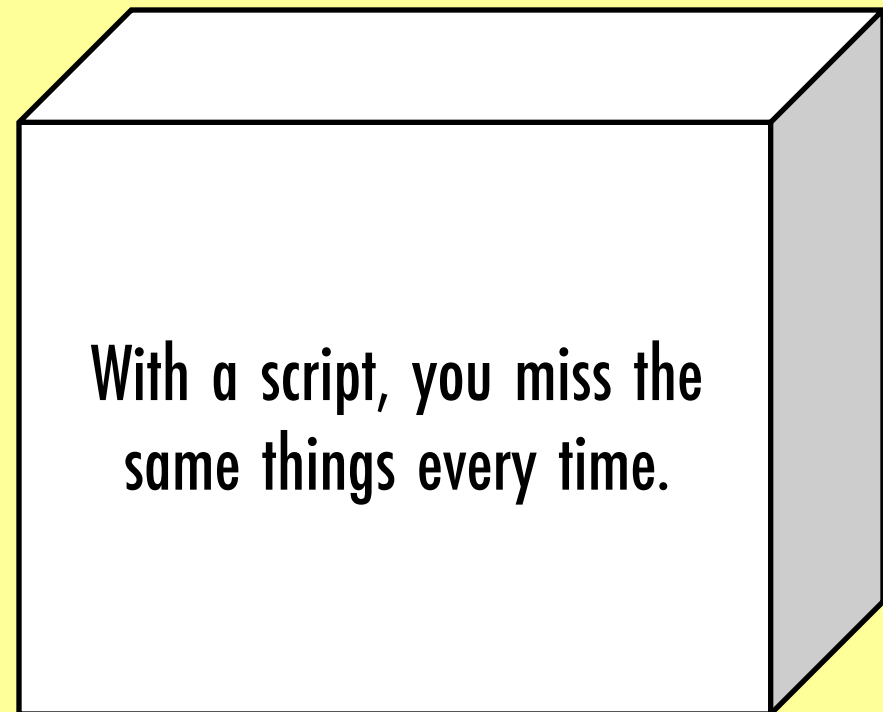
Scripts are hit and miss ...

People are finite capacity information processors

- Remember our demonstration of inattention blindness
- We pay attention to some things
 - and therefore we do NOT pay attention to others
 - Even events that “should be” obvious will be missed if we are attending to other things.

Computers focus only on what they are programmed to look at:

- They are inattentionally blind by design



Time sequence in scripted testing

- Design the test early
- Execute it many times later
- Look for the same things each time

Risk profiles evolve over time

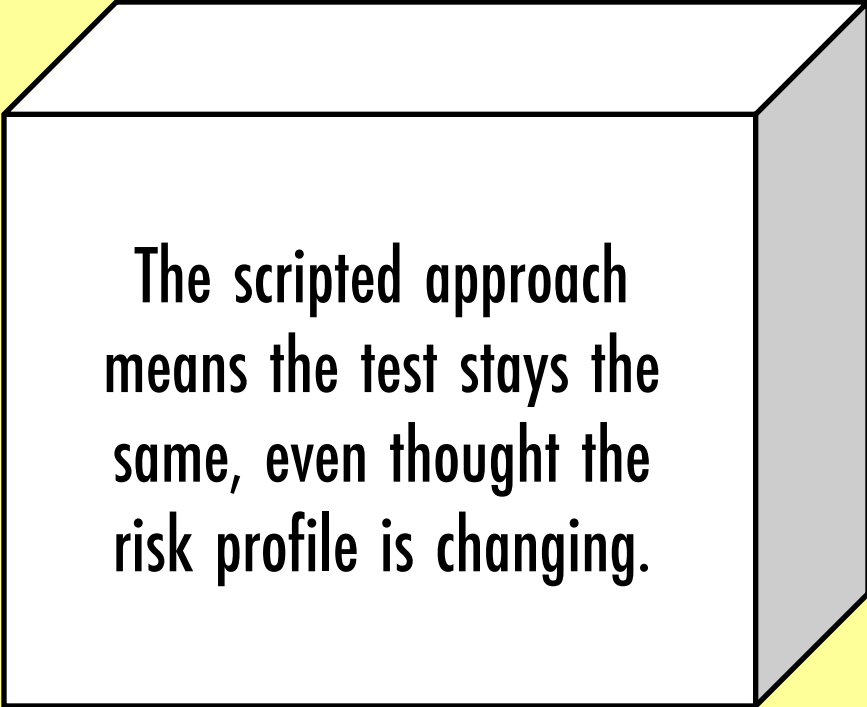
Specifying the full set of tests at the start of the project is an invitation to failure:

- The requirements / specifications are almost certain to change as the program evolves
- Different programmers tend to make different errors. (This is a key part of the rationale behind the PSP.) A generic test suite that ignores authorship will overemphasize some potential errors while underemphasizing others.
- The environment in which the software will run (platform, competition, user expectations, new exploits) changes over time.

Time sequence in scripted testing

- Design the test early
- Execute it many times later
- Look for the same things each time

- The earlier you design the tests, the less you understand the program and its risk profile
 - ***And thus, the less well you understand what to look at***



The scripted approach means the test stays the same, even though the risk profile is changing.

Cognitive sequence in scripted testing

The smart test designer

- who rarely runs the tests

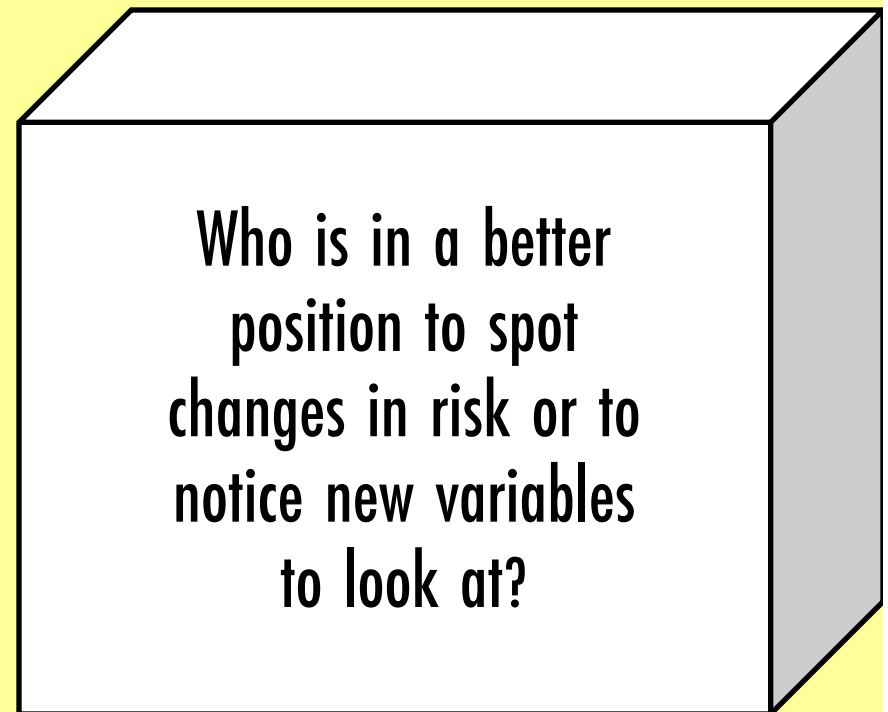
designs the tests for **the cheap tester**

- who does what the designer says and looks for what the designer says to look for
- time and time again
- independently of the risk profile.

This is very cost-effective

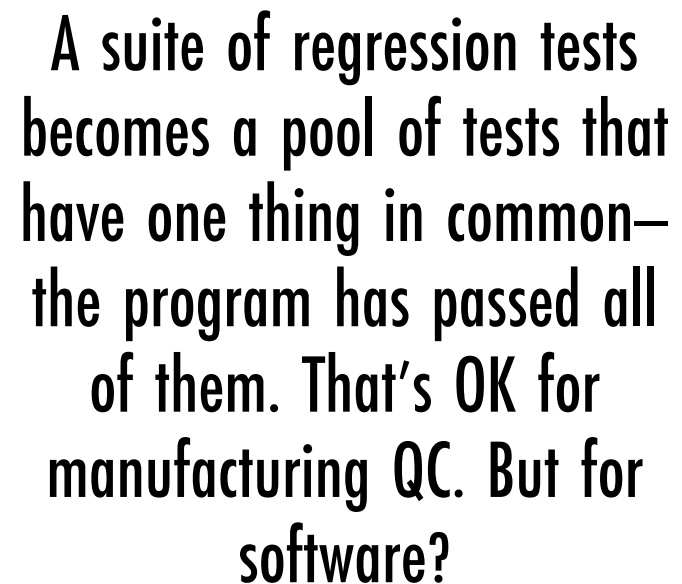
- if the program has no bugs (or only bugs that are clearly covered in the script)

But what if your program has unexpected bugs?



Analogy to Manufacturing QC

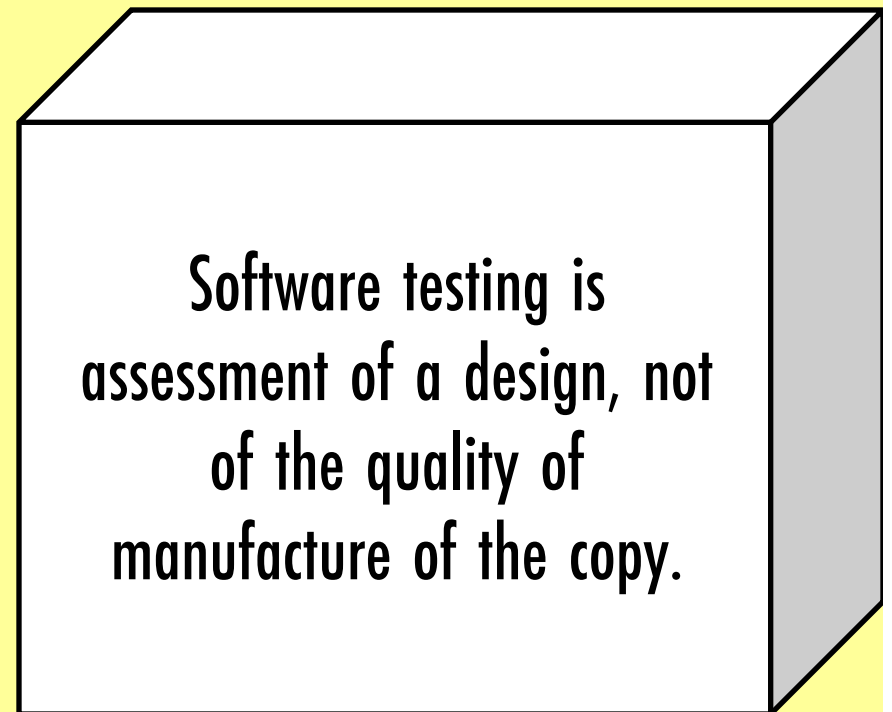
- Scripting makes a lot of sense because we have:
 - Fixed design
 - Well understood risks
 - The same set of errors appear on a statistically understood basis
 - Test for the same things on each instance of the product



A suite of regression tests becomes a pool of tests that have one thing in common—the program has passed all of them. That's OK for manufacturing QC. But for software?

Analogy to Design QC

- The difference between manufacturing defects and design defects is that:
 - A manufacturing defect appears in an individual instance of the product
 - A design defect appears in every instance of the product.
- The challenge is to find new design errors, not to look over and over and over again for the same design error



Manufacturing versus services

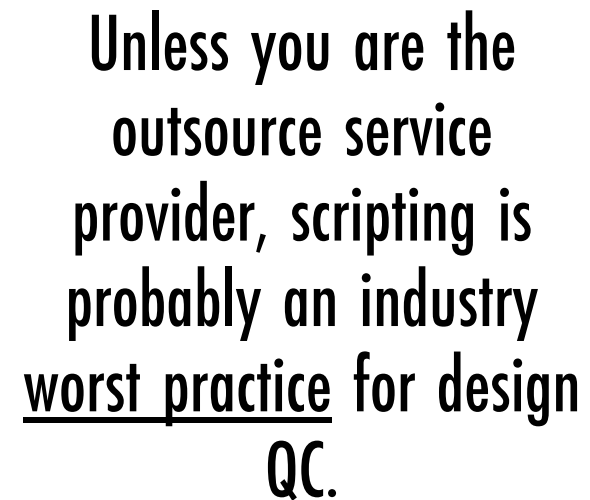
Peter Drucker, *Managing in the Next Society*, stresses that we should manufacture remotely but provide services locally.

The local service provider is more readily available, more responsive, and more able to understand what is needed.

Most software engineering standards (such as the DoD and IEEE standards) were heavily influenced by contracting firms—outsourcers.

If you choose to outsource development, **of course** you should change your practices to make them look as much like manufacturing as possible.

But is **the goal** to support outsourcing?



Unless you are the
outsource service
provider, scripting is
probably an industry
worst practice for design
QC.

What we need for design...

Is a constantly evolving set of tests

- That exercise the software in new ways (new combinations of features and data)
- So that we get our choice of
 - broader coverage of the infinite space of possibilities
 - > adapting as we recognize new classes of possibilities
 - and sharper focus
 - > on risks or issues that we decide are of critical interest today.





*The Nature of
Testing*

Testing is like CSI

CSI: CRIME SCENE INVESTIGATION
THURSDAYS 9PM ET/PT

Choose a CBS Show

EPISODES HANDBOOK VIDEO PRODUCTION INTERACTIVE

HANDBOOK EVIDENCE TOOLS PROCEDURES

AFIS

Automated Fingerprint Identification System: Computer network that scans crime-scene fingerprints and compares them with millions of prints collected by law enforcement agencies around the state, region, country, and world. A print is traced by a fingerprint expert. The tracing is then scanned by the computer, which assigns values to various features of the print. Those values are compared to other prints in the database.

CSI HANDBOOK
Learn more about tools, evidence and procedures used by CSIs.

EPISODES
Missed an episode?
Catch up on previous stories now.

PREVIOUSLY ON CSI:

Room Service
Julian Harper (23), touted as the next Brad Pitt, is taking his bright lights moment to the max. Sex, drugs and a "High Roller Suite" are put at his disposal as he enters the ...more

CSI NEWSLETTER
Be among the first to know. [Subscribe now](#) for email updates

CSI: VIDEO

Behind the Scenes
"CSI" celebrates 100 episodes

MANY tools, procedures, sources of evidence.

- Tools and procedures don't define an investigation or its goals.
- There is too much evidence to test, tools are often expensive, so investigators must exercise judgment.
- The investigator must pick what to study, and how, in order to reveal the most needed information.

Imagine ...

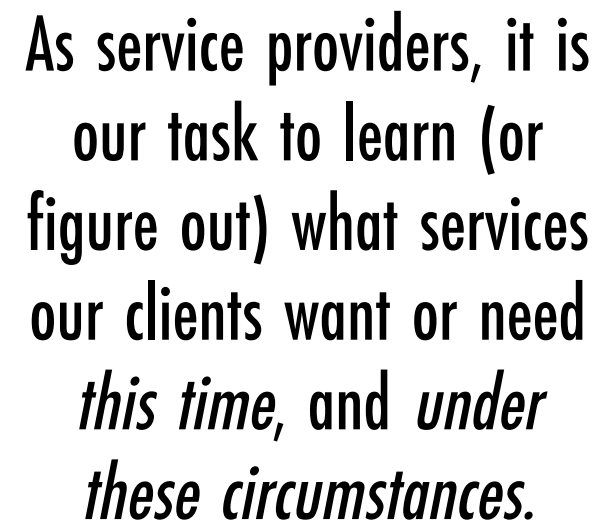
Imagine crime scene investigators

- (real investigators of real crime scenes)
- following a script.

How effective do you think they would be?

Testing is always done within a context

- We test in the face of harsh constraints
 - Complete testing is impossible
 - Project schedules and budget are finite
 - Skills of the testing group are limited
- Testing might be done before, during or after a release.
- Improvement of product or process might or might not be an objective of testing.
- We test on behalf of stakeholders
 - Project manager, marketing manager, customer, programmer, competitor, attorney
 - Which stakeholder(s) **this time**?
 - > What information are they interested in?
 - > What risks do they want to mitigate?



As service providers, it is our task to learn (or figure out) what services our clients want or need *this time*, and *under these circumstances*.

Examples of important context factors

- Who are the stakeholders with influence
- What are the goals and quality criteria for the project
- What skills and resources are available to the project
- What is in the product
- How it could fail
- Potential consequences of potential failures
- Who might care about which consequence of what failure
- How to trigger a fault that generates a failure we're seeking
- How to recognize failure
- How to decide what result variables to attend to
- How to decide what *other* result variables to attend to in the event of intermittent failure
- How to troubleshoot and simplify a failure, so as to better
 - motivate a stakeholder who might advocate for a fix
 - enable a fixer to identify and stomp the bug more quickly
- How to expose, and who to expose to, undelivered benefits, unsatisfied implications, traps, and missed opportunities.

Testing is always a search for information

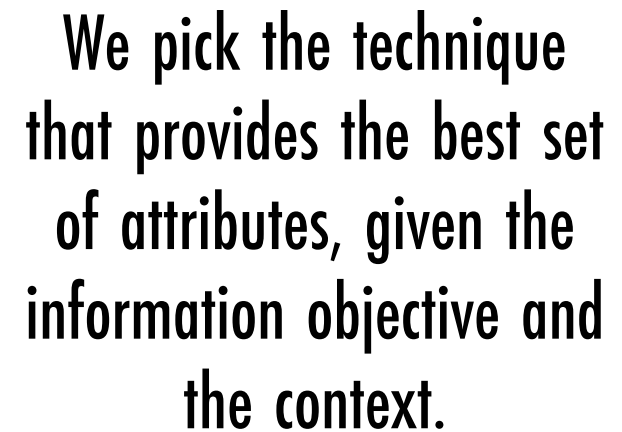
- Find important bugs, to get them fixed
- Assess the quality of the product
- Help managers make release decisions
- Block premature product releases
- Help predict and control product support costs
- Check interoperability with other products
- Find safe scenarios for use of the product
- Assess conformance to specifications
- Certify the product meets a particular standard
- Ensure the testing process meets accountability standards
- Minimize the risk of safety-related lawsuits
- Help clients improve product quality & testability
- Help clients improve their processes
- Evaluate the product for a third party

Different objectives require different testing tools and strategies and will yield different tests, different test documentation and different test results.

Ten common black-box test techniques

- Function testing
- Specification-based testing
- Domain testing
- Risk-based testing
- Scenario testing
- Regression testing
- Stress testing
- User testing
- State-model based testing
- High-volume automated testing

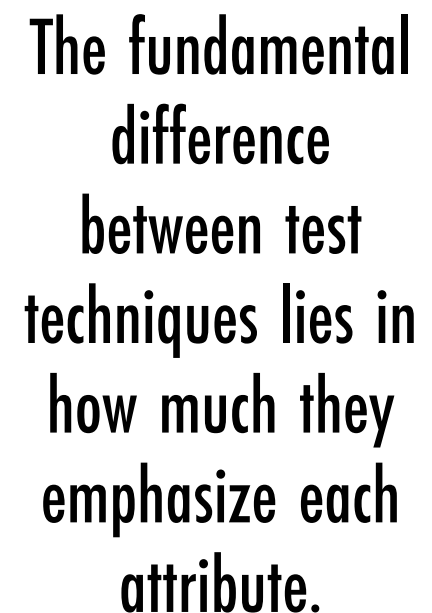
For more details, see the lecture on test design.



We pick the technique that provides the best set of attributes, given the information objective and the context.

Test attributes

- **Power:** If a problem exists, the test will reveal it
- **Valid:** If the test reveals a problem, it is a genuine problem
- **Value:** Reveals things your clients want to learn
- **Credible:** Client will believe people will do what's done in this test
- **Motivating:** Client will want to fix problems exposed by this test
- **Representative:** of events *most likely* to be encountered by the user (xref: Musa's *Software Reliability Engineering*)
- **Non-redundant:** Represents a larger set that address the same risk
- **Performable:** Test can be performed as designed.
- **Maintainable:** Easy to update to reflect product changes
- **Repeatable:** Easy and inexpensive to reuse the test
- **Potential disconfirmation:** Critical case for proving / disproving a key assumption or relationship (xref Karl Popper, *Conjectures & Refutations*)
- **Coverage:** Exercises product in ways not handled by other tests
- **Easy to evaluate**
- **Supports troubleshooting:** Provides useful information for the debugging programmer
- **Appropriately complex:** As programs get more stable, you can use more complex tests to better simulate use by experienced users
- **Accountable:** You can explain, justify, & prove you ran it.
- **Cost:** Includes time and effort as well as direct costs.
- **Opportunity cost:** Developing and performing this test prevents you from doing other work.



The fundamental difference between test techniques lies in how much they emphasize each attribute.

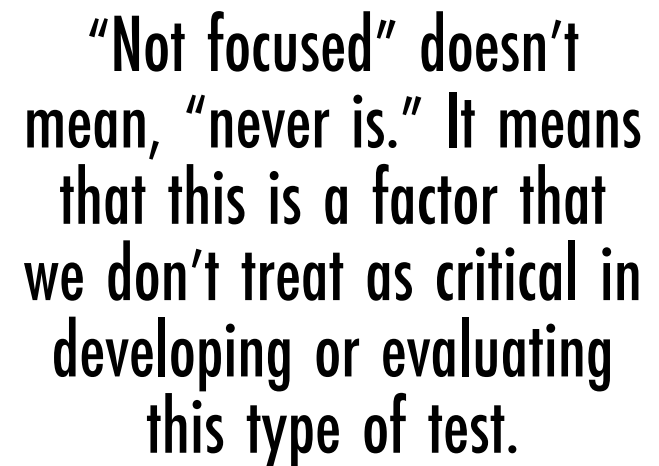
Differences in emphasis: Examples

Domain testing

- Focused on non-redundancy, validity, power, and variables-coverage. Tests are typically highly repeatable, simple, and should be easy to maintain.
- Not focused on representativeness, credibility, or motivational effect.

Scenario testing

- Focused on validity, complexity, credibility, and motivational effect.
- Not focused on power, maintainability, or coverage.



“Not focused” doesn’t mean, “never is.” It means that this is a factor that we don’t treat as critical in developing or evaluating this type of test.

Quality and errors

Quality is value to some person

-- Jerry Weinberg

Under this view:

- Quality is inherently subjective
 - Different stakeholders will perceive the same product as having different levels of quality

**Testers look for
different things ...
... for different
stakeholders**

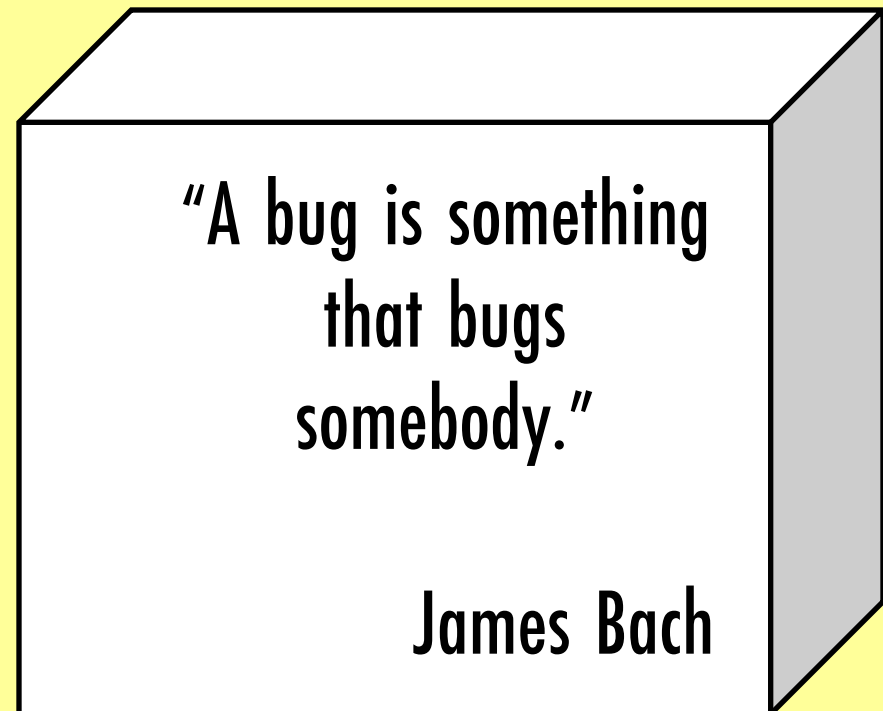
Software error

An attribute of a software product

- that reduces its value to a favored stakeholder
- or increases its value to a disfavored stakeholder
- without a sufficiently large countervailing benefit.

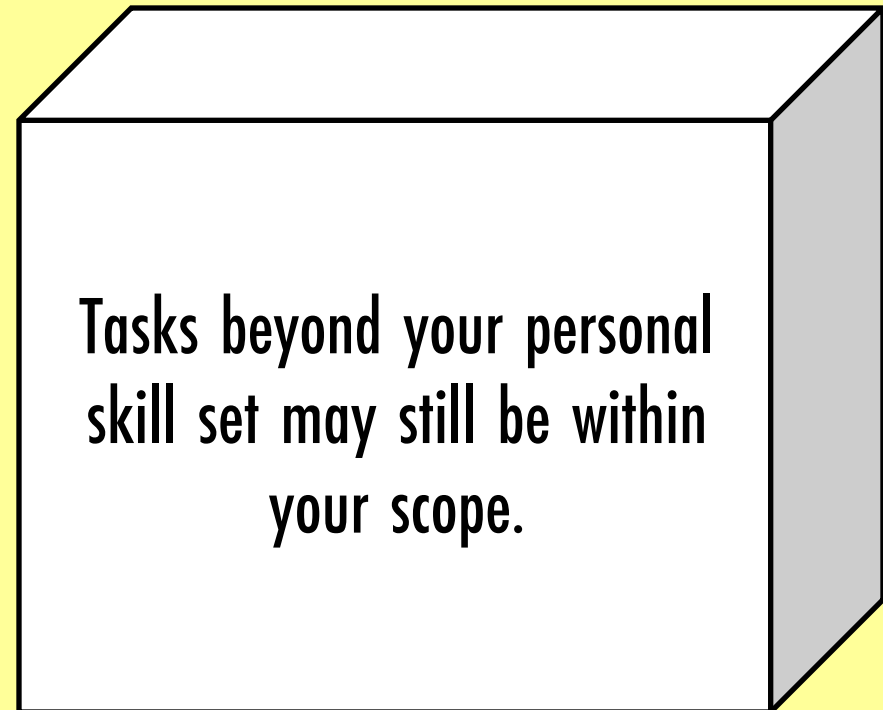
An error:

- May or may not be a coding error
- May or may not be a functional error



Reject the “Not My Job” definition of testing

- Testing is **not only** about doing tasks some programmer can imagine for you or meeting objectives some programmer wishes on you
 - *unless that programmer is your primary stakeholder*
- The tester who looks only for coding errors misses all the other ways in which a program is of lower quality than it should be.
- Anything that threatens a product’s value to a stakeholder with influence threatens quality in a way important to the project.
 - *You might be asked to investigate any type of threat, including security, performance, usability, suitability, etc.*



Software testing

- is an empirical
- technical
- investigation
- conducted to provide stakeholders
- with information
- about the quality
- of the product or service under test



*The Other
Side of the
Contrast:
Exploring*

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 her work
- by treating
 - test-related learning,
 - test design,
 - test execution, and
 - test result interpretation
- as mutually supportive activities
- that run in parallel throughout the project.

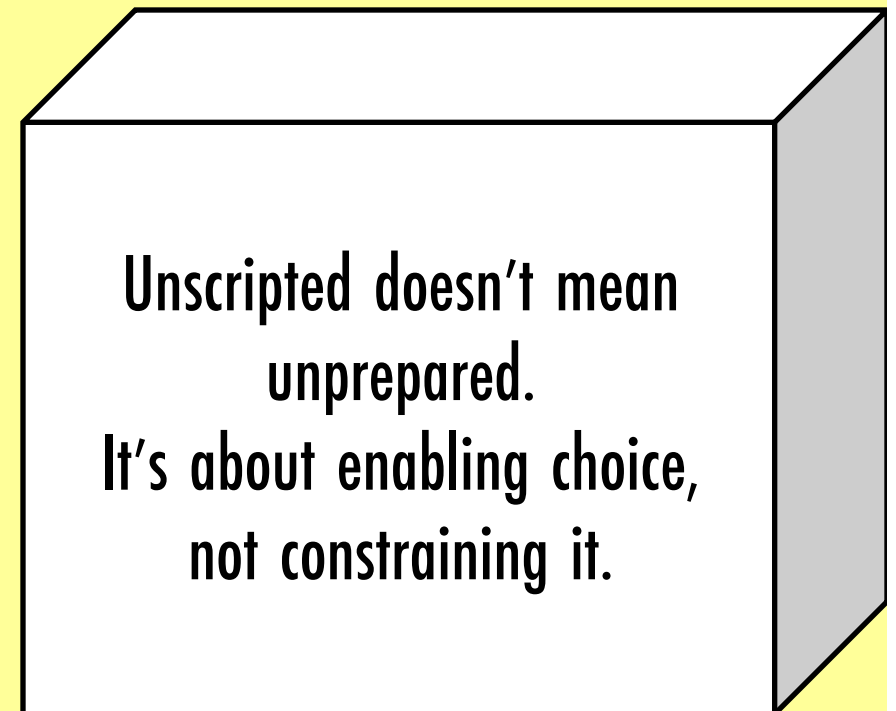
Time sequence in exploration

In contrast with scripting, we:

- Design the test as needed
- Execute the test at time of design or reuse it later
- Vary the test as appropriate, whenever appropriate.

Not scripting doesn't mean not preparing:

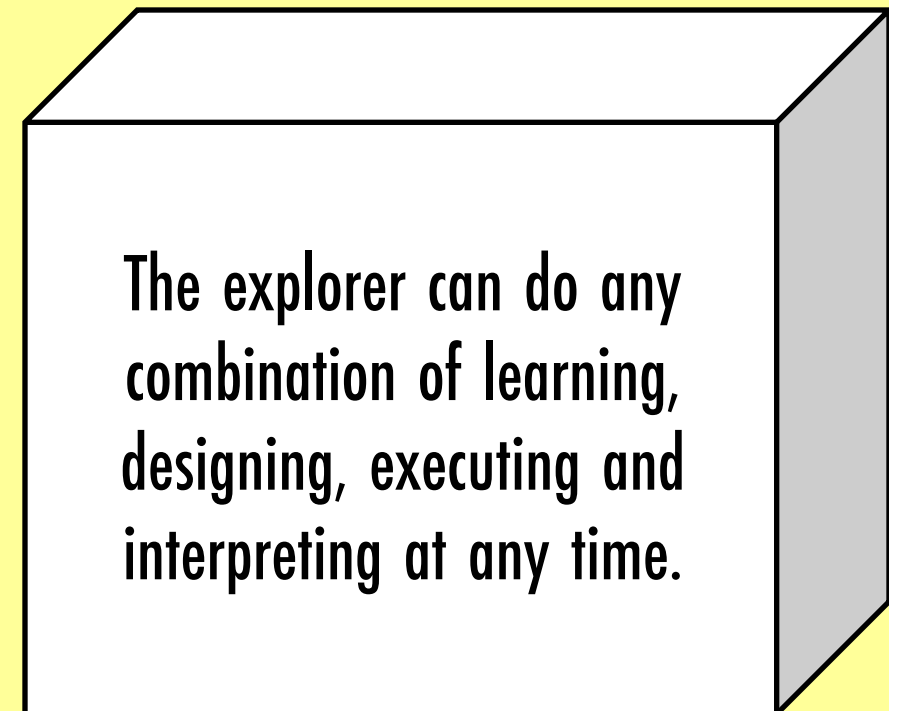
- We often design support materials in advance and use them many times throughout testing, such as
 - data sets
 - failure mode lists
 - combination charts.



Cognitive sequence in exploration

This is the fundamental difference between exploratory and scripted testing.

- **The exploratory tester is always responsible for managing the value of her own time.**
 - At any point in time, this might include:
 - > Reusing old tests
 - > Creating and running new tests
 - > Creating test-support artifacts, such as failure mode lists
 - > Conducting background research that can then guide test design



Exploratory testing

- **Learning:** Anything that can guide us in what to test, how to test, or how to recognize a problem.
- **Design:** “to create, fashion, execute, or construct according to plan; to conceive and plan out in the mind” (Websters)
 - Designing is not scripting. The representation of a plan is not the plan.
 - Explorers’ designs can be reusable.
- **Execution:** Doing the test and collecting the results. Execution can be automated or manual.
- **Interpretation:** What do we learn from program as it performs under our test
 - about the product and
 - about how we are testing the product?



*Exploratory
Testing:
Learning*

Exploratory testing: Learning

- **Learning:** Anything that can guide us in what to test, how to test, or how to recognize a problem, such as:
 - the **project context** (e.g., development objectives, resources and constraints, stakeholders with influence), **market forces** that drive the product (competitors, desired and customary benefits, users), hardware and software **platforms**, and **development history** of prior versions and related products.
 - **risks, failure history, support record** of this and related products and how this product currently behaves and fails.

Examples of learning activities

- **Study competitive products** (how they work, what they do, what expectations they create)
- **Research the history** of this / related products (design / failures / support)
- **Inspect the product under test** (and its data) (create function lists, data relationship charts, file structures, user tasks, product benefits, FMEA)
- **Question:** Identify missing info, imagine potential sources and potentially revealing questions (interview users, developers, and other stakeholders, use reference materials to supplement answers)
- **Review written sources:** specifications, other authoritative documents, culturally authoritative sources, persuasive sources
- **Try out potentially useful tools**

Examples of learning activities

- **Hardware / software platform:** Design and run experiments to establish lab procedures or polish lab techniques. Research the compatibility space of the hardware/software (see, e.g. Kaner, Falk, Nguyen's (Testing Computer Software) chapter on Printer Testing).
- **Team research:** brainstorming or other group activities to combine and extend knowledge
- **Paired testing:** mutual mentoring, foster diversity in models and approaches.

Examples of learning activities

- **Create and apply models:** A model is a simplified representation of a relationship, process or system. The simplification makes some aspects of the thing modeled clearer, more visible, and easier to work with.
- A model is often an expression of something we don't understand in terms of something we (think we) do understand
- All tests are based on models:
 - Many models are implicit
 - When the behavior of a program “feels wrong,” it is clashing with your internal model of the program (and how it should behave)

What are we modeling?

- A physical process emulated, controlled or analyzed by software under test
- A business process emulated, controlled or analyzed by software under test
- Software being emulated, controlled, communicated with or analyzed by the software under test
- Device(s) this program will interact with
- The stakeholder community
- The uses / usage patterns of the product
- The transactions that this product participates in
- The development project
- The user interface of the product
- The objects created by this product

What aspects of them are we modeling?

- Capabilities
- Preferences
 - Competitive analysis
 - Support records
- Focused chronology
 - Achievement of a task or life history of an object or action
- Sequences of actions
 - Such as state diagrams or other sequence diagrams
 - Flow of control
- Flow of information
 - Such as data flow diagrams or protocol diagrams or maps
- Interactions / dependencies
 - Such as combination charts or decision trees
 - Charts of data dependencies
 - Charts of connections of parts of a system
- Collections
 - Such as taxonomies or parallel lists
- Motives
 - Interest analysis
 - Who is affected how, by what?

What makes these models, *models*?

- The representation is simpler than what is modeled: It emphasizes some aspects of what is modeled while hiding other aspects
- You can work with the representation to make descriptions or predictions about the underlying subject of the model
- Using the model is easier or more convenient to work with, or more likely to lead to new insights than working with the original.

A model of learning

	COGNITIVE PROCESSES					
KNOWLEDGE DIMENSIONS	Remember	Understand	Apply	Analyze	Evaluate	Create
Facts						
Concepts						
Procedures						
Cognitive strategies						
Models						
Skills						
Attitudes						
Metacognition						

This is an adaptation of Anderson/Krathwohl's learning taxonomy. For a summary and links, see <http://www.satisfice.com/kaner/?p=14>

Focusing on models

- All tests **are** based on models
 - But any cognitive or perceptual psychologist will tell you that all perceptions and all judgments are based on models
 - > Most of which are implicit

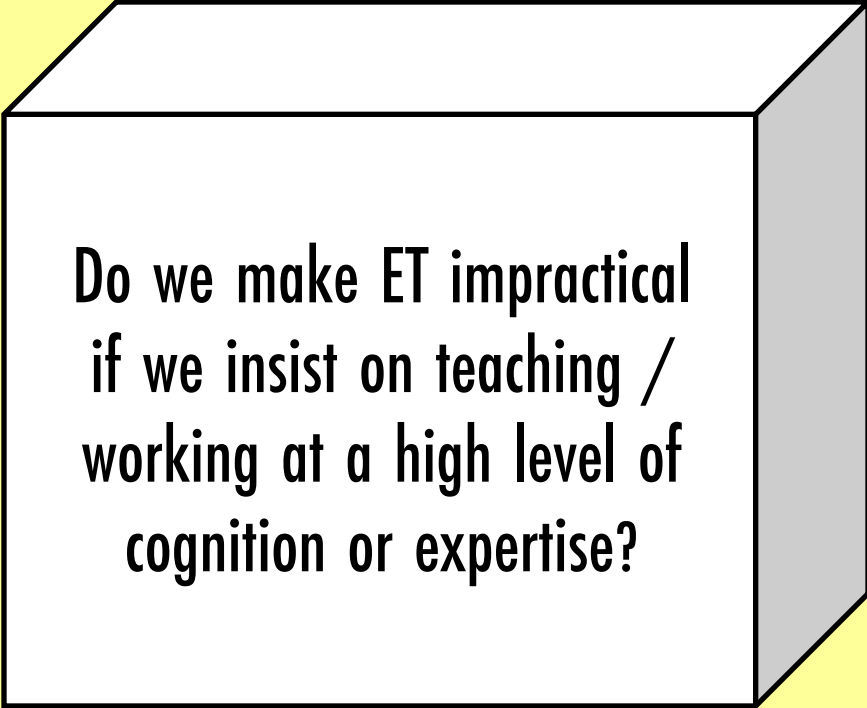
A model of learning

	COGNITIVE PROCESSES					
KNOWLEDGE DIMENSIONS	Remember	Understand	Apply	Analyze	Evaluate	Create
Facts						
Concepts						
Procedures						
Cognitive strategies						
Models						
Skills						
Attitudes						
Metacognition						

This is an adaptation of Anderson/Krathwohl's learning taxonomy. For a summary and links, see <http://www.satisfice.com/kaner/?p=14>

Focusing on models

- All tests **are** based on models
 - But any cognitive or perceptual psychologist will tell you that all perceptions and all judgments are based on models
 - > Most of which are implicit
- So the question is,
 - Is it **useful** to focus on discovering, evaluating, extending, and creating models
 - Or are we sometimes better off leaving the models in the background while we focus on the things we are modeling?



Do we make ET impractical if we insist on teaching / working at a high level of cognition or expertise?



*Exploratory
Testing:
Design*

Exploratory testing: Design

- **Learning:** Anything that can guide us in what to test, how to test, or how to recognize a problem.
- **Design:** “to create, fashion, execute, or construct according to plan; to conceive and plan out in the mind” (Websters)
 - Designing is not scripting. The representation of a plan is not the plan.
 - Explorers’ designs can be reusable.
- **Execution:** Doing the test and collecting the results. Execution can be automated or manual.
- **Interpretation:** What do we learn from program as it performs under our test
 - about the product and
 - about how we are testing the product?

Examples of design activities

- Map test ideas to FMEA or other lists of variables, functions, risks, benefits, tasks, etc.
- Map test techniques to test ideas
- Map tools to test techniques.
- Map staff skills to tools / techniques, develop training as necessary
- Develop supporting test data
- Develop supporting oracles
- Data capture: notes? Screen/input capture tool? Log files? Ongoing automated assessment of test results?
- Charter: Decide what you will work on and how you will work

Design: Challenge of relevance

- The challenge of exploratory testing is often to take a test idea (especially potential problem)
 - maybe learned from study of competitor's product, or support history, or failure of other products on this operating system or written in this programming language
- And turn the test idea into one or more tests

How do we map from a test idea to a test?

Design: Challenge of relevance

- We often go from technique to test
 - Find all variables, domain test each
 - Find all spec paragraphs, make a relevant test for each
 - Find all lines of code, make a set of tests that collectively includes each
- It is much harder to go from a risk to a test
 - The program will crash?
 - The program will have a wild pointer?
 - The program will have a memory leak?
 - The program will be hard to use?
 - The program will corrupt its database?

How to map from a test idea to a test?

- I don't have a general answer.
- Cross-mapping of knowledge is one of (perhaps *the*) most difficult cognitive tasks.
 - Ability to do this is often discussed in terms of “G” (“general intelligence”, the hypothetical dominant factor that underlies IQ scores)

How to map from a test idea to a test?

- When it is not clear how to work backwards to the relevant test, four tactics sometimes help:
 - Ask someone for help
 - Ask Google for help. (Look for discussions of the type of failure; look for discussions of different faults and see what types of failures they yield)
 - Review your toolkit of techniques, searching for a test type with relevant characteristics
 - Turn the failure into a story and gradually evolve the story into something you can test from
- There are no guarantees in this, but you get better at it as you practice, and as you build a broader inventory of techniques.

Design: Challenge of relevance

If you don't have a technique at hand, you will often have to invent your own.

Or at least, polish a test idea into a good test.

This is especially true with stories that give an initial approach to a risk but need work.

Example:

Joe bought a smart refrigerator that tracks items stored in the fridge and prints out grocery shopping lists. One day, Joe asked for a shopping list for his usual meals in their usual quantities and the fridge crashed with an unintelligible error message.

How would you test for this bug?

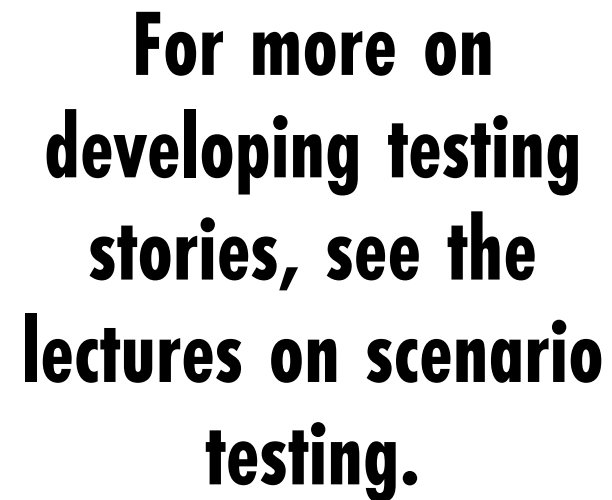
Design: Challenge of relevance: Evolving the test case from the story

- We start with Joe and his failure.
- We generate hypotheses for situations that *might* lead to a failure like that:
 - Wild pointer
 - Stack overflow
 - Unusual timing condition
 - Unusual collection of things in the fridge
- Refine each hypothesis into harsher and harsher tests until we're satisfied that if the program passes *this* series of tests, the hypothesis driving the tests is probably the wrong one.

Design: Challenge of relevance: Evolving the test case from the story

To achieve this, we might:

- Look for a potentially promising technique
- Work up a starting example of this type of test that appears relevant to the failure under consideration
- Try out the test
 - If you get the failure this easily, stop
 - Otherwise, polish the test
 - > Consider the strengths of this class of test
 - > Stretch the test on the attributes not normally emphasized by this technique.



**For more on
developing testing
stories, see the
lectures on scenario
testing.**



*Exploratory
Testing:
Execution*

Exploratory testing: Execution

- **Learning:** Anything that can guide us in what to test, how to test, or how to recognize a problem.
- **Design:** “to create, fashion, execute, or construct according to plan; to conceive and plan out in the mind” (Websters)
 - Designing is not scripting. The representation of a plan is not the plan.
 - Explorers’ designs can be reusable.
- **Execution:** Doing the test and collecting the results.
Execution can be automated or manual.
- **Interpretation:** What do we learn from program as it performs under our test
 - about the product and
 - about how we are testing the product?

Examples of execution activities

- Configure the product under test
- Branch / backtrack: Let yourself be productively distracted from one course of action in order to produce an unanticipated new idea.
- Alternate among different activities or perspectives to create or relieve productive tension
- Pair testing: work and think with another person on the same problem
- Vary activities and foci of attention
- Create and debug an automated series of tests
- Run and monitor the execution of an automated series of tests

Scripted execution

	COGNITIVE PROCESSES					
KNOWLEDGE DIMENSIONS	Remember	Understand	Apply	Analyze	Evaluate	Create
Facts						
Concepts						
Procedures						
Cognitive strategies						
Models						
Skills						
Attitudes						
Metacognition						

The individual contributor (tester rather than “test planner” or manager)

Exploratory execution

	COGNITIVE PROCESSES					
KNOWLEDGE DIMENSIONS	Remember	Understand	Apply	Analyze	Evaluate	Create
Facts						
Concepts						
Procedures						
Cognitive strategies						
Models						
Skills						
Attitudes						
Metacognition						

The individual contributor (tester rather than “test planner” or manager)



*Exploratory
Testing:
Interpretation*

Exploratory testing: Interpreting

- **Learning:** Anything that can guide us in what to test, how to test, or how to recognize a problem.
- **Design:** “to create, fashion, execute, or construct according to plan; to conceive and plan out in the mind” (Websters)
 - Designing is not scripting. The representation of a plan is not the plan.
 - Explorers’ designs can be reusable.
- **Execution:** Doing the test and collecting the results. Execution can be automated or manual.
- **Interpretation:** What do we learn from program as it performs under our test
 - about the product and
 - about how we are testing the product?

Interpretation activities

- Part of interpreting the behavior exposed by a test is determining whether the program passed or failed the test.
- A mechanism for determining whether a program passed or failed a test is called an **oracle**. We discuss oracles in detail, on video and in slides
- Oracles are heuristic: they are incomplete and they are fallible. One of the key interpretation activities is determining which oracle is useful for a given test or test result

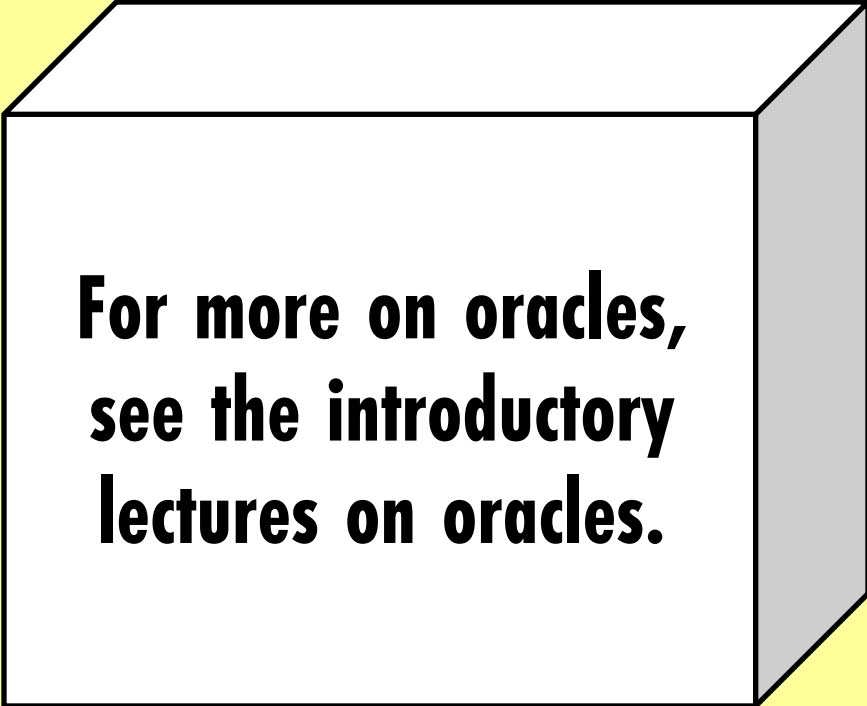
Interpretation: Oracle heuristics

Consistent within Product: Behavior consistent with behavior of comparable functions or functional patterns within the product.

Consistent with Comparable Products: Behavior consistent with behavior of similar functions in comparable products.

Consistent with a Model's Predictions: Behavior consistent with expectations derived from a model.

Consistent with History: Present behavior consistent with past behavior.



**For more on oracles,
see the introductory
lectures on oracles.**

Interpretation: Oracle heuristics

Consistent with our Image:

Behavior consistent with an image that the organization wants to project.

Consistent with Claims: Behavior consistent with documentation or ads.

Consistent with Specifications or Regulations: Behavior consistent with claims that must be met.

Consistent with User's Expectations: Behavior consistent with what we think users want.

Consistent with Purpose: Behavior consistent with apparent purpose.

Another set of activity descriptions

- Jon Bach, Mike Kelly, and James Bach are working on a broad listing / tutorial of ET activities. See *Exploratory Testing Dynamics* at <http://www.quardev.com/whitepapers.html>
- We reviewed preliminary drafts at the Exploratory Testing Research Summit (spring 2006) and Consultants Camp 2006 (August), looking specifically at teaching issues.
- This short paper handout provides an outline for what should be a 3-4 day course. It's a stunningly rich set of skills.
- In this abbreviated form, the lists are particularly useful for audit and mentoring purposes, to highlight gaps in your test activities or those of someone whose work you are evaluating.



*Exploratory
Testing
After 23
Years*

Exploratory testing after 23 years

Areas of agreement	Areas of controversy
Areas of progress	Areas of ongoing concern

Areas of agreement*

- Definitions
- Everyone does ET to some degree
- ET is an approach, not a technique
- ET is the response (the antithesis) to scripting
 - But a piece of work can be a blend, to some degree exploratory and to some degree scripted

* Agreement among the people who agree with me (many of whom are sources of my ideas). This is a subset of the population of ET-thinkers who I respect, and a smaller subset of the pool of testers who feel qualified to write about ET. (YMMV)

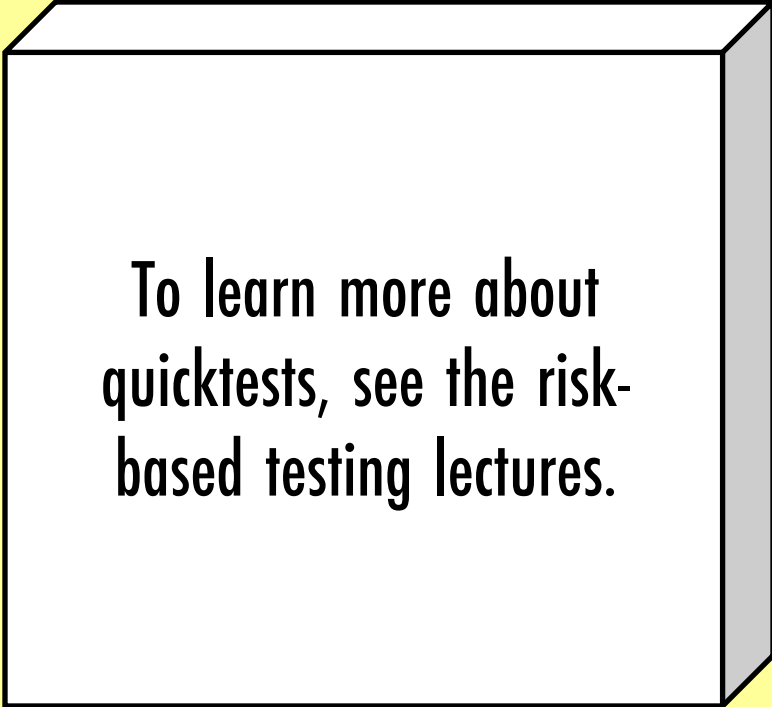
Exploratory testing after 23 years

Areas of agreement	Areas of controversy
Areas of progress	Areas of ongoing concern

Areas of controversy

ET is not quicktesting

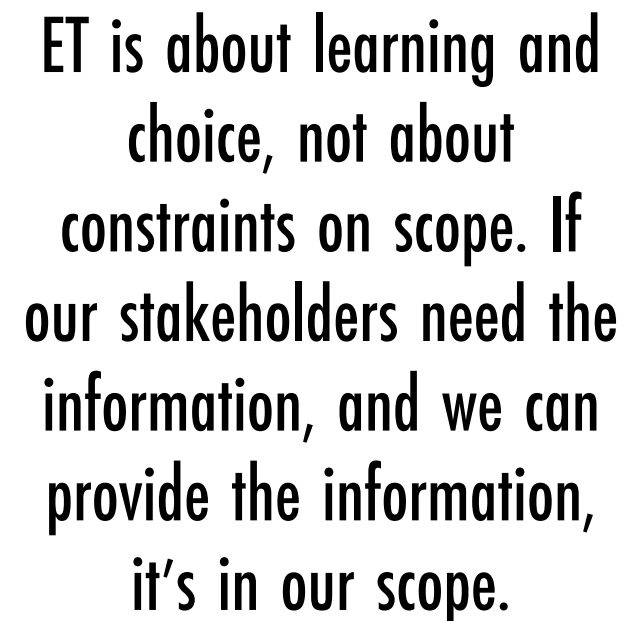
- A quicktest (or an “attack”) is a cheap test that requires little preparation, knowledge or time to perform.
- A quicktest is a technique that starts from a theory of error (how the program could be broken) and generates tests optimized for errors of that type.
 - Example: Boundary analysis (domain testing) is optimized for misclassification errors (IF $A < 5$ miscoded as IF $A \leq 5$)
- Quicktesting may be more like scripted testing or more like ET
 - depends on the mindset of the tester.



To learn more about quicktests, see the risk-based testing lectures.

Areas of controversy

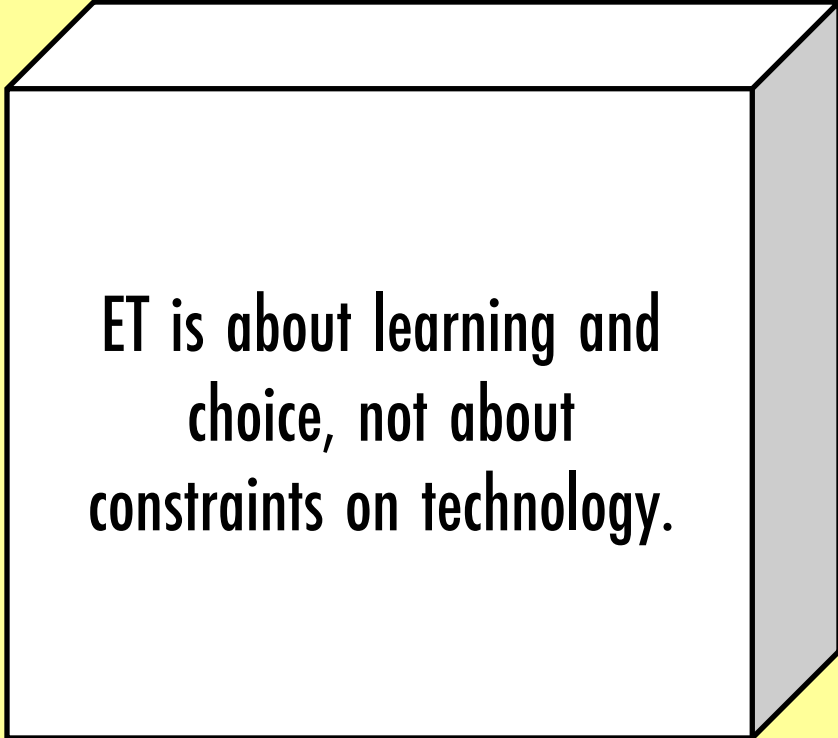
- ET is not quicktesting
- **ET is not only functional testing:**
 - Some programmers define testing narrowly
 - Agile TM system testing focused around customer stories—not a good vehicle for parafunctional attributes
 - Parafunctional work is dismissed as peripheral
 - If quality is value to the stakeholder
 - and if value is driven by usability, security, performance, aesthetics, (etc.)
 - then testers should investigate these aspects of the product.



ET is about learning and choice, not about constraints on scope. If our stakeholders need the information, and we can provide the information, it's in our scope.

Areas of controversy

- ET is not quicktesting
- ET is not only functional testing
- **ET can involve tools of any kind and can be as computer-assisted as anything else we would call “automated”**
 - Along with
 - traditional “test automation” tools,
 - Emerging tool support for ET such as
 - Test Explorer
 - BBTest Assistant
 - and better thought support tools
 - Like Mind Manager and Inspiration
 - Qualitative analysis tools like Atlas.ti



ET is about learning and choice, not about constraints on technology.

The Telenova stack failure

Telenova Station Set I. Integrated voice and data.
108 voice features, 110 data features. 1984.



The Telenova stack failure

```
July 4, 1985      12:01 PM      Ext: 257  
Directory Admin  Messages  Voice Data
```

```
1-(212)662-7777 Connected      Ext: 567  
Transfer Record  Conference Park  Acct
```

```
Please enter selection  
LvMsa      GetMsa      Greeting  Code
```

```
Ted K. waiting      Wt:1 Hd:0  
I'llCall  CallLater PlsWait  Answ
```

```
Select a call & lift handset  Wt:5 Hd:5  
Ted K.      Peter T.  Trunk 6  Trk 2Trk 7
```

```
Kenix 3 Connected for Data  
Transfer  Baud      EndCall  Park Acct
```

Context-sensitive
display

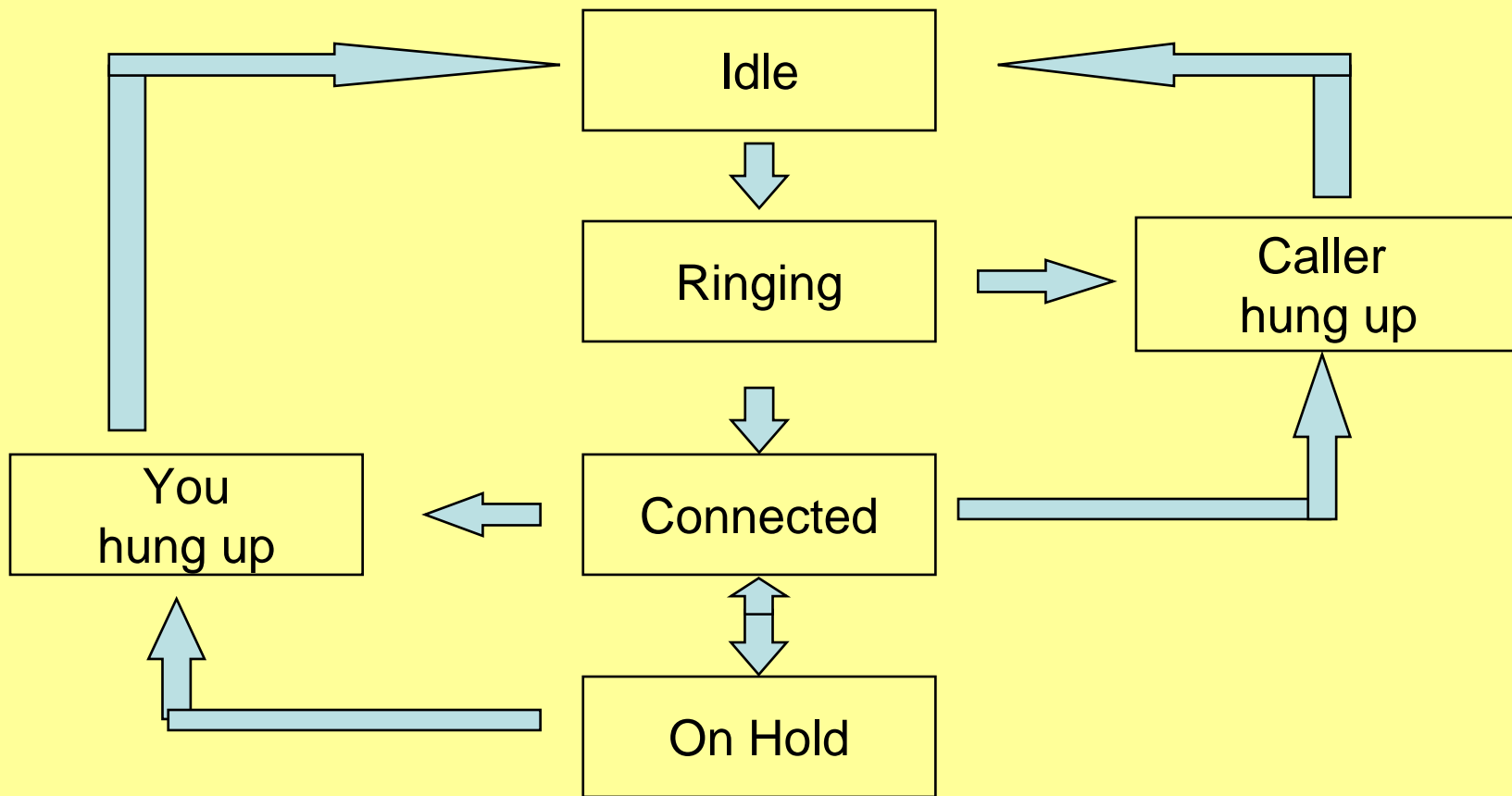
10-deep hold queue

10-deep wait queue



The Telenova stack failure

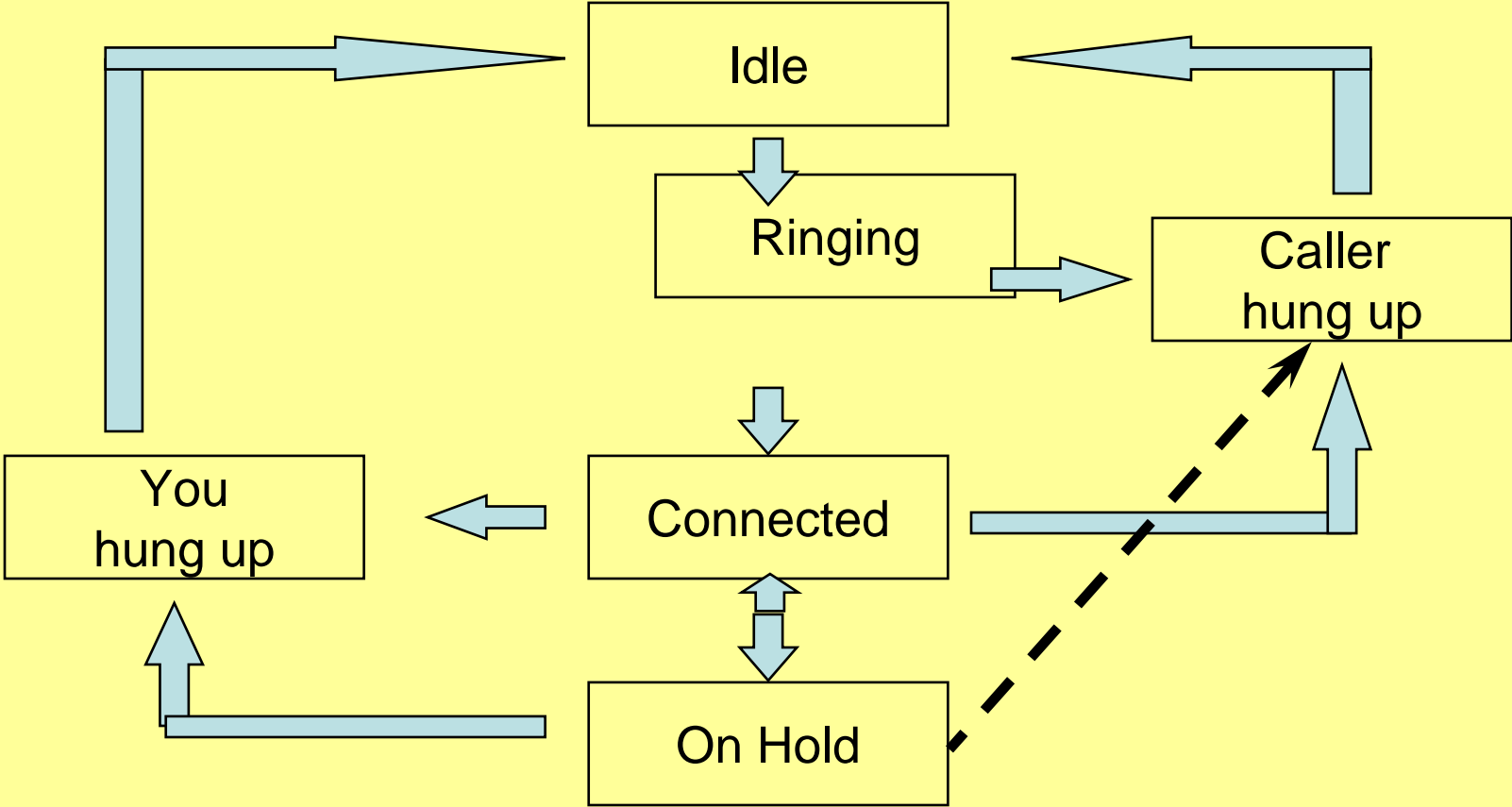
A simplified state diagram showing the bug



The underlying bug:

- Beta customer (stock broker) had random failures
- Could be frequent at peak times
- An individual phone would crash and reboot. Others crashed while the first was rebooting
- One busy day, service was disrupted all afternoon
- We were mystified:
 - All individual functions worked
 - We had tested all lines and branches.
- Ultimately, we found the bug in the hold queue
 - Up to 10 held calls, each adds record to the stack
 - Initially, the system checked stack whenever it added or removed a call, but this took too much system time. We dropped the checks and added:
 - Stack has room for 20 calls (just in case)
 - Stack reset (forced empty) when we knew it *should* be empty
 - Couldn't overflow the stack in the lab because we didn't know how to hold more than 10 calls.

The magic error



Telenova stack failure

**Having found and fixed
the hold-stack bug,
should we assume
we've taken care of the problem
or that if there's one long-sequence bug,
there will be more?**

Hmmm...

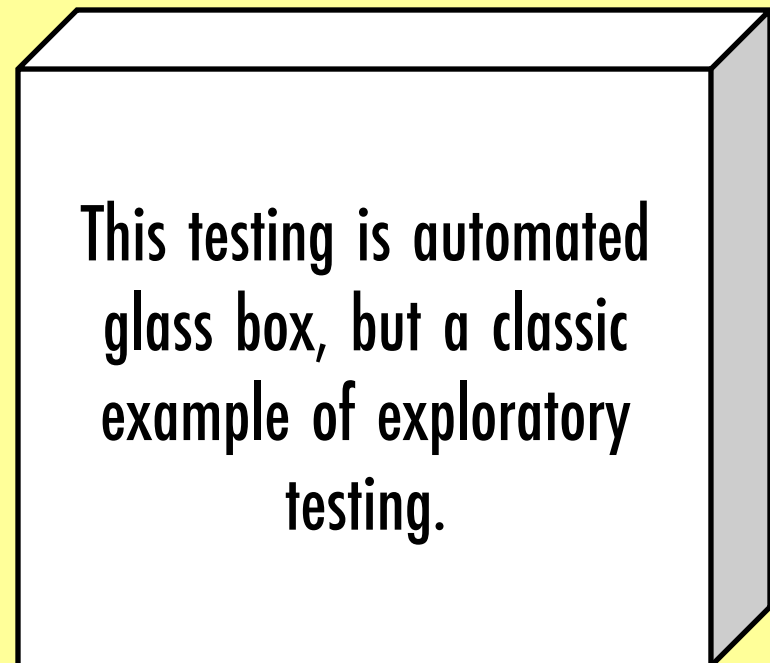


**If you kill a cockroach in your kitchen,
do you assume
you've killed the last bug?
Or do you call the exterminator?**

Simulator with probes

- Telenova (*) created a simulator
 - generated long chains of random events, emulating input to the system's 100 phones
 - could be biased, to generate more holds, more forwards, more conferences, etc.
- Programmers selectively added probes (non-crashing asserts that printed alerts to a log)
 - can't probe everything b/c of timing impact
- After each run, programmers and testers tried to replicate / fix anything that triggered a message
- When logs ran almost clean, shifted focus to next group of features.
- Exposed lots of bugs

(*) By the time this was implemented, I had joined Electronic Arts.



Areas of controversy

- ET is not quicktesting
- ET is not only functional testing
- ET can involve tools of any kind and can be as computer-assisted as anything else we would call “automated”
- **ET is not focused primarily around test execution**
 - I helped create this confusion by initially talking about ET as a test technique.

Controversy: ET is not a technique

In the 1980's and early 1990's, I distinguished between

- The evolutionary approach to software testing
- The exploratory testing technique(s), such as:
 - Guerilla raids
 - Taxonomy-based testing and auditing
 - Familiarization testing (e.g. user manual conformance tests)
 - Scenario tests

Controversy: ET is not a technique

1999 Los Altos Workshop on Software Testing #7 on Exploratory Testing

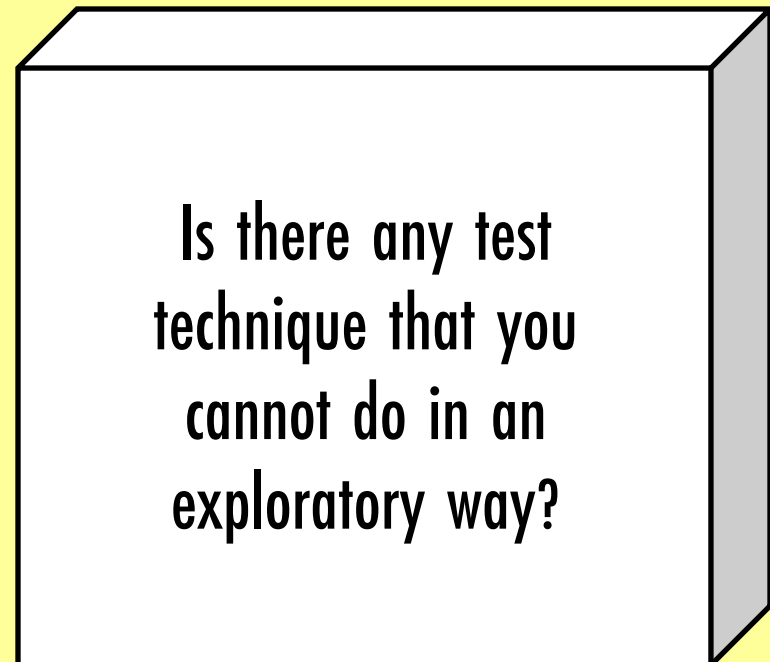
- James Tierney presented observations on MS “supertesters” indicating their strength is heavily correlated with social interactions in the development group (they translate what they learn from the team into tests)
- Bob Johnson and I presented a list of “styles of exploration” (a catalog of what we now call “quicktests”)
- James Bach, Elisabeth Hendrickson, Harry Robinson, and Melora Svoboda gave presentations on models to drive exploratory test design

Controversy: ET is not a technique

At end of LAWST 7, David Gelperin concluded he didn't understand what is unique about "exploratory" testing. Our presentations all described approaches to design and execution of tests that he considered normal testing. What was the difference?

He had a point:

- Can you do domain testing in an exploratory way?
 - *Of course*
- Specification-based testing?
 - *Sure*
- Stress testing? Scenario testing? Model-based testing?
 - *Yes, yes, yes*



Controversy: ET is not a technique

WHET #1 and #2 – James Bach demonstrated that activities we undertake to learn about the product (in order to test it) are inherent in exploration.

- Of course they are
- But this became the death knell for the idea of ET as a technique
- **ET is a way of testing**
 - We learn about the product in its market and technological space (keep learning until the end of the project)
 - We take advantage of what we learn to design better tests and interpret results more sagely
 - We run the tests, shifting our focus as we learn more, and learn from the results.

Areas of controversy

- ET is not quicktesting
- ET is not only functional testing
- ET can involve tools of any kind and can be as computer-assisted as anything else we would call “automated”
- ET is not focused primarily around test execution
- **ET can involve complex tests that require significant preparation**
 - Scenario testing is the classic example
 - To the extent that scenarios help us understand the design (and its value), we learn most of what we’ll learn in the development and first execution. Why keep them?



ET is not just spontaneous testing at the keyboard.

Areas of controversy

- ET is not quicktesting
- ET is not only functional testing
- ET can involve tools of any kind and can be as computer-assisted as anything else we would call “automated”
- ET is not focused primarily around test execution
- ET can involve complex tests that require significant preparation
- **ET is not exclusively black box**
 - “Experimental program analysis: A new paradigm for program analysis” by Joseph Ruthruff (Doctoral symposium presentation at International Conference on Software Engineering, 2006)



ET is not just spontaneous testing at the keyboard.

Exploratory testing after 23 years

Areas of agreement	Areas of controversy
Areas of progress	Areas of ongoing concern

Areas of progress

- We know a lot more about quicktests
 - Well documented examples from Whittaker's *How to Break...* series and Hendrickson's and Bach's courses

Areas of progress

- We know a lot more about quicktests
- We have a better understanding of the oracle problem and oracle heuristics

Areas of progress

- We know a lot more about quicktests
- We have a better understanding of the oracle problem and oracle heuristics
- We have growing understanding of ET in terms of theories of learning and cognition

Areas of progress

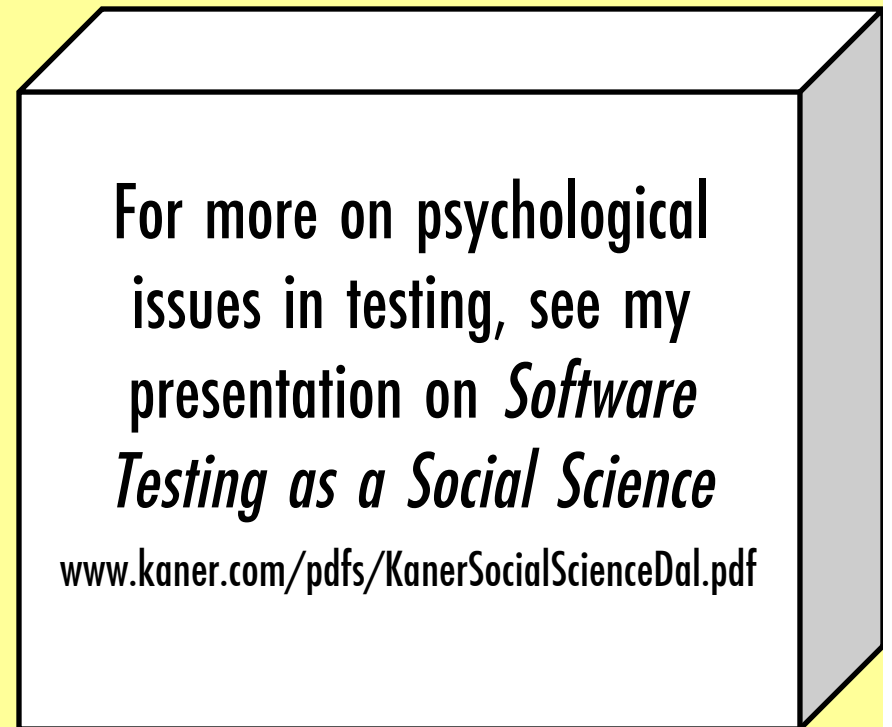
- We know a lot more about quicktests
- We have a better understanding of the oracle problem and oracle heuristics
- We have growing understanding of ET in terms of theories of learning and cognition
- **We have several guiding models**
 - We now understand that models are implicit in all tests
 - Failure mode & effects analysis applied to bug catalogs
 - Bach / Bach / Kelly's activities model
 - Satisfice heuristic test strategy model
 - State models
 - Other ET-supporting models (see Hendrickson, Bach)

Exploratory testing after 23 years

Areas of agreement	Areas of controversy
Areas of progress	Areas of ongoing concern

Areas of ongoing concern

- **Testing is**
 - **more skilled and cognitively challenging**
 - **more fundamentally multidisciplinary**
 - **than popular myths expect**



Areas of ongoing concern

- **Testing is more skilled and cognitively challenging, more fundamentally multidisciplinary, than popular myths expect:**
- **Unskilled testing shows up more starkly with ET**

Areas of ongoing concern

Testing is more skilled and cognitively challenging, more fundamentally multidisciplinary, than popular myths expect:

Unskilled testing shows up more starkly with ET

- Repetition without realizing it
- Areas missed without intent
- Incorrect perception of depth or coverage
- Tester locks down on a style of testing without realizing it
- Wasted time due to reinvention of same tests instead of reuse
- Wasted effort creating test data
- Audit fails because of lack of traceability
- Weak testing because the tester is unskilled and tests are unreviewed
- Difficult to document the details of what was done
- May be difficult to replicate a failure
- Hard to coordinate across testers
- Harder to spot a failure.

The essence of ET is learning (and learning about learning)

	COGNITIVE PROCESSES					
KNOWLEDGE DIMENSIONS	Remember	Understand	Apply	Analyze	Evaluate	Create
Facts						
Concepts						
Procedures						
Cognitive strategies						
Models						
Skills						
Attitudes						
Metacognition						

The individual contributor (tester rather than “test planner” or manager)

Areas of ongoing concern

- Testing is more skilled and cognitively challenging, and more fundamentally multidisciplinary, than popular myths expect
- **What level of skill, domain knowledge, intelligence, testing experience (overall “strength” in testing) does exploratory testing require?**
 - We are still early in our wrestling with modeling and implicit models
 - > How to teach the models
 - > How to teach how to model

The essence of ET is learning

	COGNITIVE PROCESSES					
KNOWLEDGE DIMENSIONS	Remember	Understand	Apply	Analyze	Evaluate	Create
Facts						
Concepts						
Procedures						
Cognitive strategies						
Models						
Skills						
Attitudes						
Metacognition						

The individual contributor (tester rather than “test planner” or manager)

The essence of ET is learning

	COGNITIVE PROCESSES					
KNOWLEDGE DIMENSIONS	Remember	Understand	Apply	Analyze	Evaluate	Create
Facts						
Concepts						
Procedures						
Cognitive strategies						
Models						
Skills						
Attitudes						
Metacognition						

The individual contributor (tester rather than “test planner” or manager)

The essence of ET is learning

	COGNITIVE PROCESSES					
KNOWLEDGE DIMENSIONS	Remember	Understand	Apply	Analyze	Evaluate	Create
Facts						
Concepts						
Procedures						
Cognitive strategies						
Models						
Skills						
Attitudes						
Metacognition						

The individual contributor (tester rather than “test planner” or manager)

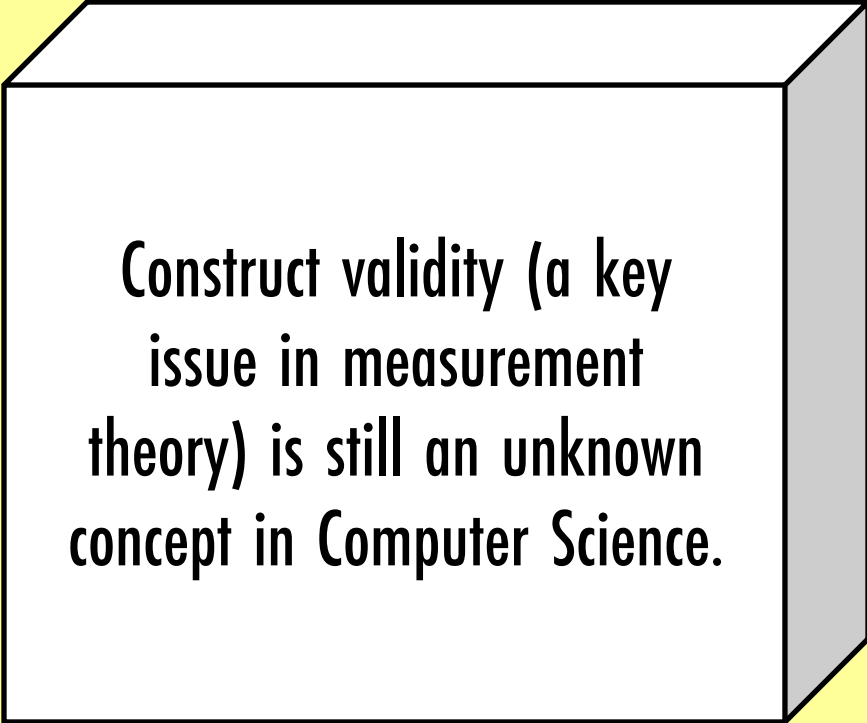
The essence of ET is learning (and learning about learning)

	COGNITIVE PROCESSES					
KNOWLEDGE DIMENSIONS	Remember	Understand	Apply	Analyze	Evaluate	Create
Facts						
Concepts						
Procedures						
Cognitive strategies						
Models						
Skills						
Attitudes						
Metacognition						

The individual contributor (tester rather than “test planner” or manager)

Areas of ongoing concern

- Testing is more skilled and cognitively challenging, and more fundamentally multidisciplinary, than popular myths expect
- What level of skill, domain knowledge, intelligence, testing experience (overall “strength” in testing) does exploratory testing require?
- **We are just learning how to assess individual tester performance**



Construct validity (a key issue in measurement theory) is still an unknown concept in Computer Science.

Areas of ongoing concern

- Testing is more skilled and cognitively challenging, and more fundamentally multidisciplinary, than popular myths expect
- What level of skill, domain knowledge, intelligence, testing experience (overall “strength” in testing) does exploratory testing require?
- We are just learning how to assess individual tester performance
- **We are just learning how to track and report status**
 - Session based testing
 - Workflow breakdowns
 - Dashboards

Areas of ongoing concern

- Testing is more skilled and cognitively challenging, and more fundamentally multidisciplinary, than popular myths expect
- What level of skill, domain knowledge, intelligence, testing experience (overall “strength” in testing) does exploratory testing require?
- We are just learning how to assess individual tester performance
- We are just learning how to track and report status
- **We don't yet have a good standard tool suite**
 - Tools guide thinking
 - Hendrickson, Bach, others have made lots of suggestions

Closing notes

- If you want to attack any approach to testing as unskilled, attack scripted testing
- If you want to hammer any testing approach on coverage, look at the fools who think they have tested a spec or requirements document when they have one test case per spec item, or code with one test per statement / branch / basis path.
- Testing is a skilled, fundamentally multidisciplinary area of work.
- Exploratory testing brings to the fore the need to adapt to the changing project with the information available.