Exploratory Testing Dynamics

IQAA Quality Enrichment Conference

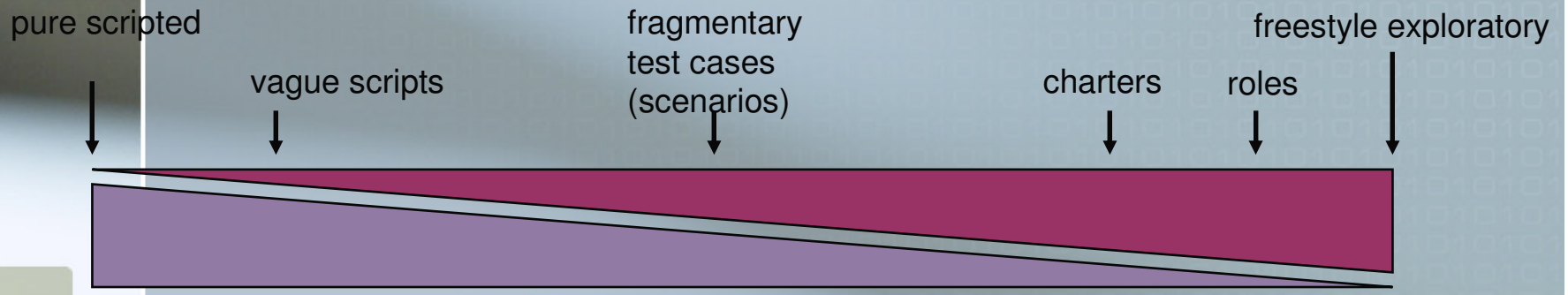
Michael Kelly – October 13, 2006

Acknowledgements

- The outline for this talk is based on the work of James and Jon Bach and their work on Exploratory Testing Dynamics.
- I also pirated material from Jon Bach's recent presentations on Exploratory Testing.
- The Exploratory Testing Research Summit also helped shape this presentation. Participants included:
  - James Bach
  - Jonathan Bach
  - Scott Barber
  - Michael Bolton
  - Elisabeth Hendrickson
  - Cem Kaner
  - Michael Kelly
  - Jonathan Kohl
  - James Lyndsay
  - Robert Sabourin

Exploratory Testing

- Exploratory testing is the opposite of *scripted* testing.



- Both scripted and exploratory testing are better thought of as test *approaches*, rather than techniques.

What are the dynamics of ET?

- Exploratory testing is often considered mysterious and unstructured. Not so! You just need to know what to look for.
- The following are some of the many dynamics that comprise exploratory testing.
- These are the skills that comprise professional exploration of technology.

The Dynamics of ET

- Part 1: Skills and Tactics
- Part 2: Evolving Work Products
- Part 3: Testing Considerations

Part 1

Skills and Tactics of Exploratory Testing

Skills and Tactics

- Principles:
  - Related to exploratory testing
  - Critical to exploration (if you take it away, you have some sort of sick exploration)
  - Unique
  - Observable, assessable, and improvable

Skills and Tactics

These are the skills that comprise professional exploration of technology that can be observed, measured, and improved.

Modeling
Resourcing
Chartering

Questioning
Observing
Manipulating
Collaboration

Generating/Elaborating
Overproduction/Abandonment
Abandonment/Recovery
Refocusing
Alternating
Branching/Backtracking
Conjecturing

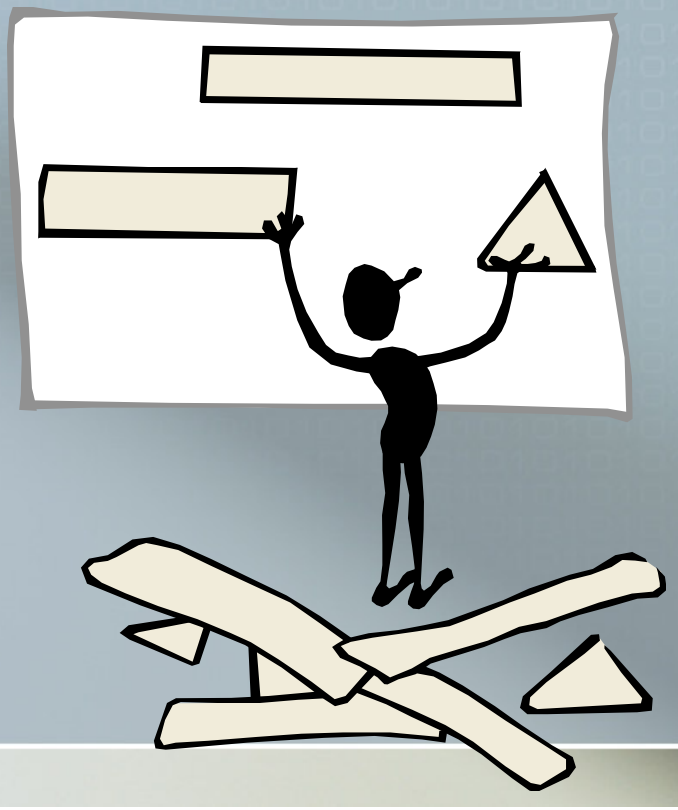
Recording
Reporting

Modeling

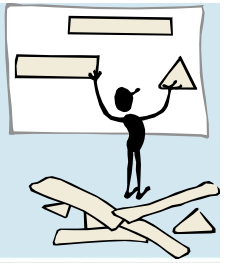
Composing, describing, and working with mental models of the things you are exploring. Identifying relevant dimensions, variables, and dynamics. A good mental model may manifest itself as having a "feel" for the product; intuitively grasping how it works.

Examples:

- Drawing a map for someone to get to your house.
- Developing a UCML diagram for a large performance test.
- Describing the political structure of your organization to a new hire.



Modeling



What it looks like when you do it poorly:

- An inability to identify the relevant dimensions, variables, or dynamics of the object you are modeling.
- Developing a model and then never actually using it (without making the conscious decision to abandon it).
- Failing to recognize that all models are flawed and incomplete.

How could you get better at it:

- Learn commonly used formal models: UML, UCML, systems diagramming.
- Give yourself a series of test exercises where you physically (pen and paper) model each problem three or more times and develop test cases off of each model. Notice how the different models inspire different types of tests.
- Partner with someone else. Take turns naming different dimensions of an object.

Resourcing

Obtaining tools and information to support your effort. Exploring sources of such tools and information. Getting people to help you.

Examples:

- Breaking out of functional fixedness:
  - Using a performance test tools to run high-volume functional test automation.
  - Using a functional test tool for performance testing when a performance test tool is too expensive or not practical.
- Influencing a developer or analyst to give you thirty minutes of their time to answer your questions.
- Convincing management that you need two more testers to hit the deadline, then being able to identify and hire them in time.

Resourcing

What it looks like when you do it poorly:

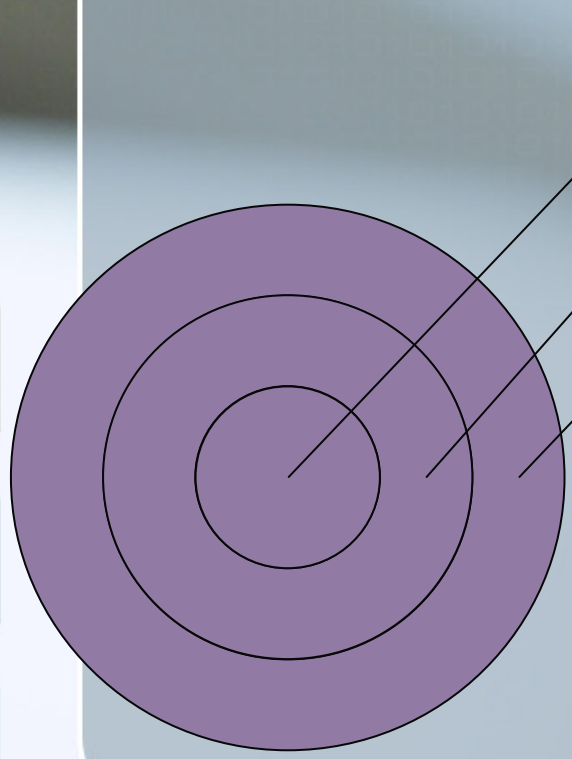
- Inability to get people to help you.
- Spending too much time looking for or setting up resources and not enough time testing.
- Inability to recognize when existing tools already solve the problem.

How could you get better at it:

- Learn the principles of negotiation.
- Learn a programming language (Ruby, Pearl, Java, Python, etc…).
- Set up a weekly review meeting for your team where each person gives a ten to fifteen minute presentation on a tool, artifact, practice, or idea that they reviewed that week.

Chartering

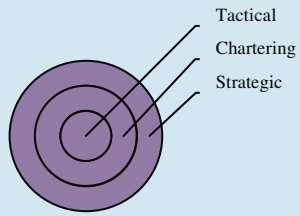
Making your own decisions about what you will work on and how you will work. Understanding your client's needs, the problems you must solve, and assuring that your work is on target. Making your work relevant to the client.



Examples:

- Strategic: developing a test strategy.
- Managing: developing an exploratory testing charter.
- Tactical: deciding what test to run next.

Chartering



What it looks like when you do it poorly:

- The client doesn't understand the work you are doing or why you are doing it.
- You don't understand the work you are doing or why you are doing it.
- You don't know when you are done with your testing. You don't have a good enough understanding of the problem to know what good enough testing is.

How could you get better at it:

- Work with a group of testers, each of you develop a test strategy for a product on your own, then compare and contrast your strategies.
- Charter your day to day activities in 45 minute blocks. Practice your estimation techniques, and practices varying the language of your charters to better capture the work you are actually doing at that time.
- Practice selling your ideas. Use PMs, managers, and skeptics.

Exercise: Greenland

These are the skills that comprise professional exploration of technology that can be observed, measured, and improved.

Modeling

Resourcing

Chartering

Planning Testing

Questioning

Identifying missing information, conceiving of questions, and asking questions in a way that elicits the information that you seek. Knowing *when* to ask your questions for optimal information usage.

Examples:

- Open book testing.
- Reviewing a requirements document or participating in a requirements gathering session.
- Using technical jargon when asking questions of developers, business jargon when asking questions of your customer, and no jargon when asking questions of your spouse.

Questioning

What it looks like when you do it poorly:

- You run out of questions quickly or fail to ask any all together.
- You ask too many questions or ask them too quickly, overwhelming the other party and making your session less effective then it could be.
- You ask questions without regard to the value of the information, failing to know which next piece of information will be the most important to the testing project.

How could you get better at it:

- Play the game 20 Questions and other free-form question based games.
- Review requirements documents and identify the assumptions that are not explicit in the specifications.
- Try open book testing.

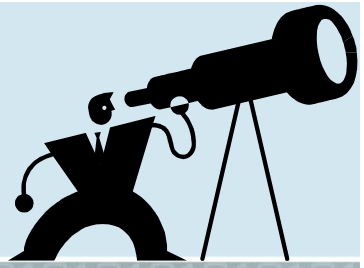
Observing

Gathering empirical data about the object of your study; collecting different kinds of data, or data about different aspects of the object; establishing procedures for rigorous observations.

Examples:

- Logging information during your testing (application logs, PerfMon/SysMon, code coverage, etc…).
- Using tools that allow you to "see" things you can't normally see (pixel comparison tools, spell checkers, conformance validation tools [WSDL, Section 508, etc…], Headspace, etc...).
- Counter example: inattentional blindness.

Observing



What it looks like when you do it poorly:

- Inattentional blindness: you don't notice what you are not looking for.
- You don't leverage tools to look for specific types of errors that they can find more easily then humans.
- Tunnel vision: You always collect the same type of data, find the same types of errors, and use the same types of lab procedures.

How could you get better at it:

- Practice using different tools that capture information about the application: runtime analysis tools, logs, validators, system monitoring tools, etc...
- Practice describing the dimensions and behaviors of the Workroom Productions black box test machines: http://www.workroom-productions.com/black_box_machines.html. Get to the point where someone could program a duplicate from your description.
- Pair with someone. As you are testing a product, describe your observations out loud (stream of consciousness) to the other person. When you grow silent, they should comment on the aspects that they noticed that you didn't. Or they can point out other aspects that you can look at that you haven't yet.

Manipulating

Making and managing contact with the object of your study; configuring and interacting with it; establishing procedures for better control of experimental conditions.

Examples:

- OFAT vs. MFAT
- Developing and using lab procedures.
- Using tools to extend your reach into the application you're testing.

Manipulating

What it looks like when you do it poorly:

- You don't automate repetitive tasks that you find yourself always doing for test setup.
- You always interact with the application in the same way. You don't have a rich library of methods to pull from for different methods of interaction.
- You introduce errors into your testing due to faulty lab procedures.

How could you get better at it:

- Practice using tools that extend your "reach" into the application: automation tools, imaging tools, virtual environments, etc...
- Practice alternating between OFAT and MFAT while testing. Use an egg timer to keep you switching.
- Practice using different methods of interaction during 5 to 15 minute test sessions: mouse only, keyboard only, API only, hotkeys, accessibility, etc...

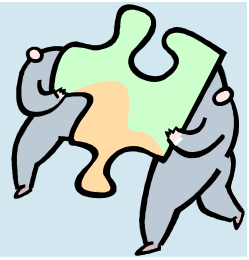
Collaboration

Working and thinking with others on the same problem; group problem-solving.

Examples:

- Pair programming.
- Riffing with someone to help test ideas and see if they "stick."
- Co-authoring.

Collaboration



What it looks like when you do it poorly:

- You are always working alone. There is not a second chair next to your desk, or it's often empty.
- You don't get regular feedback on your testing, management style, or deliverables.
- Your team meetings don't actually result in forward progress, they are "around the table" status meetings where little value is exchanged.

How could you get better at it:

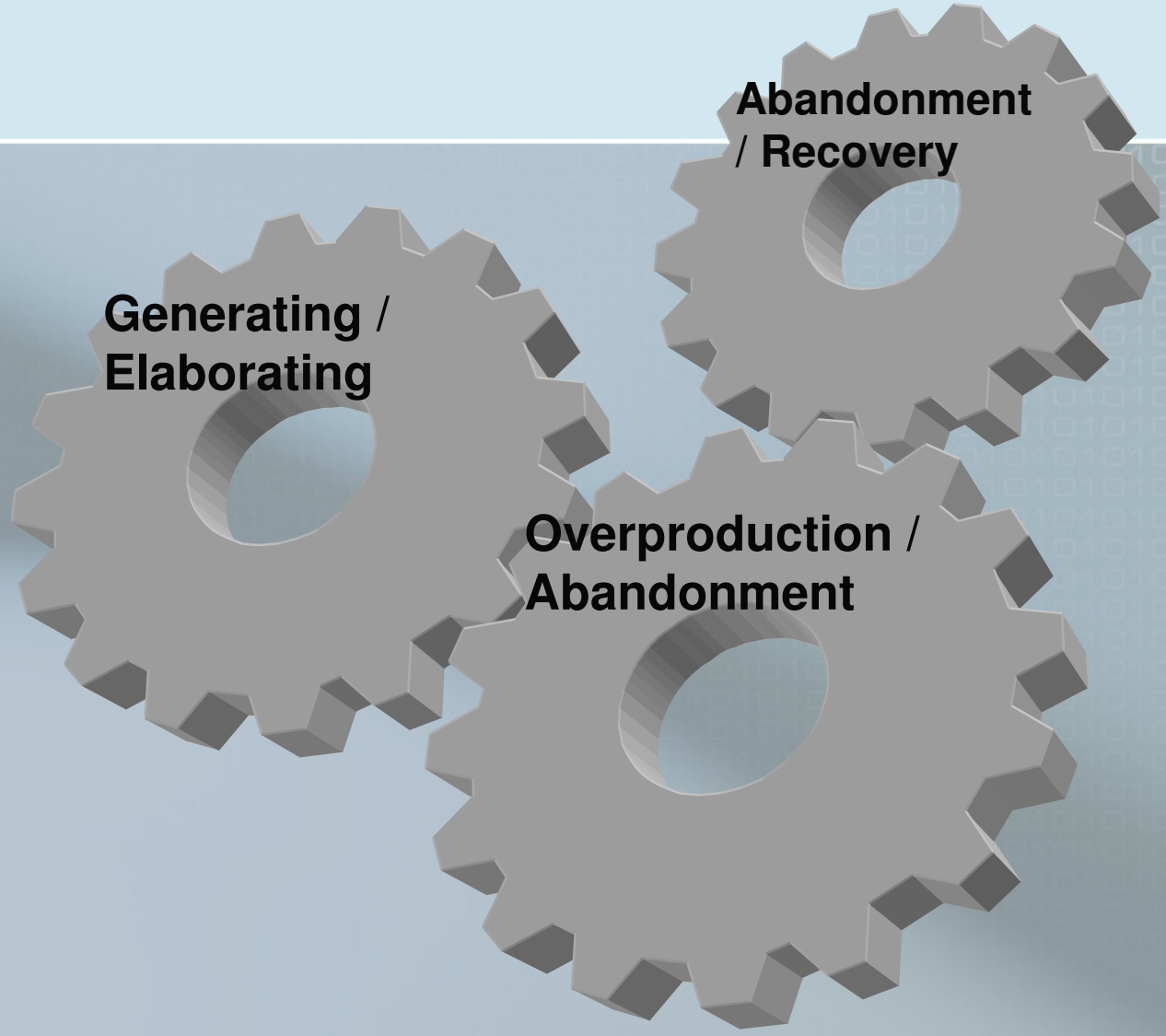
- Pair testing / pair programming.
- Set up regular interaction for feedback: peer reviews, session debriefs, pairing, and group problem solving.
- Deliver status one-on-one and use team meetings to discuss issues that affect the team.

Exercise: Greenland

These are the skills that comprise professional exploration of technology that can be observed, measured, and improved.

Questioning
Observing
Manipulating
Collaboration

Doing Testing


Abandonment
/ Recovery
Generating /
Elaborating
Overproduction /
Abandonment

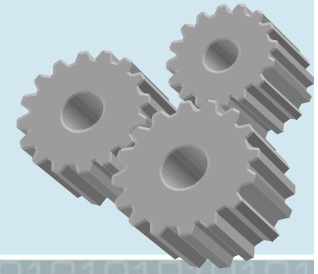
Generating/Elaborating

Working quickly in a manner good enough for the circumstances. Revisiting the solution later to extend, refine, refactor or correct it.

Examples:

- Developing an initial list of charters, and then add and remove charters over time as the project unfolds.
- Writing code, and then refactoring that code over time as you actually start using it.
- Modeling.

Generating/Elaborating



What it looks like when you do it poorly:

- Your ideas don't evolve over time. You are doing something today the exact same way you were doing it a year ago.
- You can't generate new ideas and alternatives. You rely on recovery for ideas or you rely on someone/something else for your ideas.
- You continue to change something that's working past the point of good enough. You change for the sake of change. No new value is added with the change.

How could you get better at it:

- Develop and practice with heuristics to trigger your thinking when you need to generate ideas quickly.
- Practice using bug taxonomies to refine your planning and strategy products.
- Develop a personal syllabus for self education, complete with tasks and resources. Each week, revisit that syllabus and refine it to match your new needs and goals.

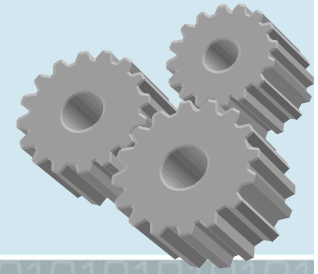
Overproduction/Abandonment

Producing many different speculative ideas and making speculative experiments, more than you probably need, then abandoning what doesn't work.

Examples:

- Brainstorming, trial and error, "bracketing" in photography, genetic algorithms, free market dynamics.
- Random tests: acknowledging that we may be systematically doing bad testing and running random tests to ensure that we didn't miss something with our other tests.
- Monitoring and logging during a performance test: collecting large amounts of data from servers, log files, performance monitors, and performance tools in order to aggregate the data later and only use a portion of it in the final analysis or report.

Overproduction/ Abandonment



What it looks like when you do it poorly:

- Excessive or superfluous amounts of data or ideas.
- Not abandoning data or ideas.
- Analysis/Paralysis.

How could you get better at it:

- Following a thread for five more minutes.
- Identify the things you currently maintain that don't add value, try not maintaining them.
- Develop heuristics for idea production (we'll look at some later…).

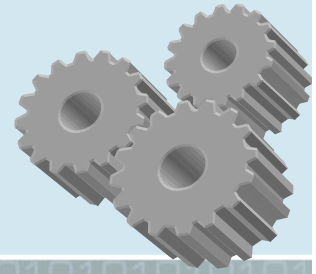
Abandonment/Recovery

Abandoning ideas and materials in such a way as to facilitate their recovery, should they need to be revisited. Maintaining a "boneyard" of old ideas. Repurposing artifacts or ideas.

Examples:

- Junkyard wars.
- The MacGyver principle: saturate yourself with ideas about what things are and how they might be used so that when you're in a situation and you need to solve a problem you can put together a solution from parts that other people might not see as available.
- Going back to a past release and re-using a pile of test cases for a feature that you know little about.

Abandonment/Recovery



What it looks like when you do it poorly:

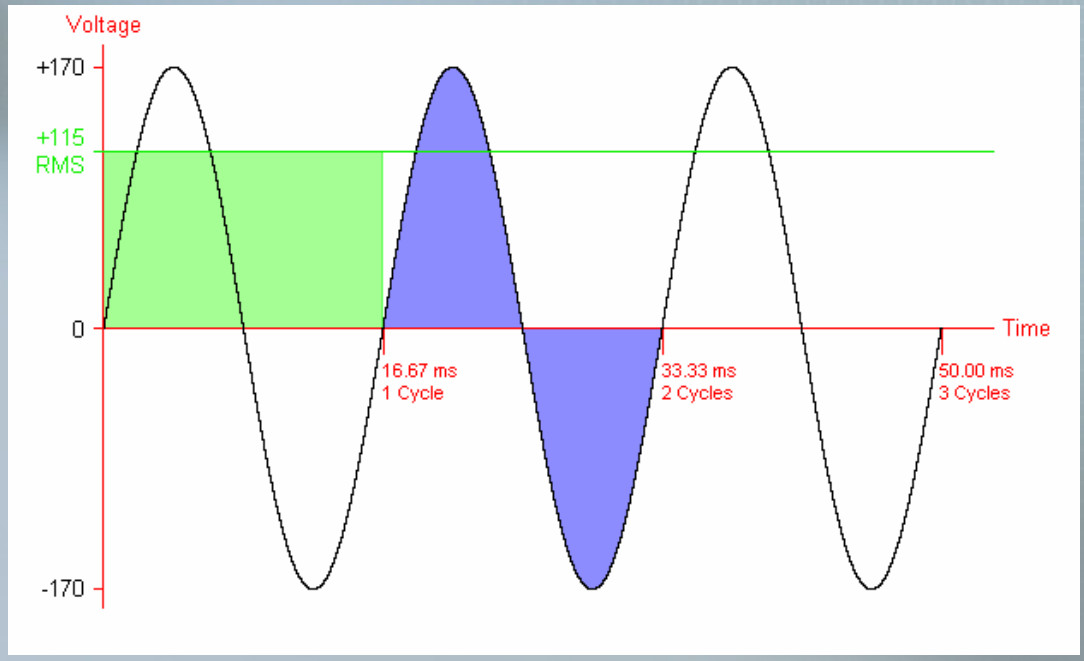
- You fail to abandon in a way that facilitates recovery.
- You don't know how to "partial recover" something. It's all or nothing.
- You recover too often. You rely on the past when what you really need to do is recreate the wheel.

How could you get better at it:

- Build your "boneyard." Create a place for ideas (both personal and professional). Make sure it's accessible and searchable. (Moleskin, email, blog posts and articles, thumb drives)
- Scan past artifacts on a regular basis. If you don't keep your mental index up to date, the boneyard is worthless.
- Find a template you use (a kind of institutionalized recovery), and refactor it. Challenge each element and brainstorm missing elements. Close it. Now reinvent the template from scratch (*no copy and paste*).

Alternating

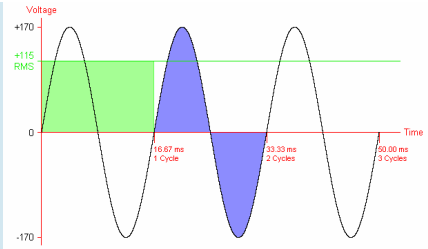
Switching among different activities or perspectives to create or relieve productive tension and make faster progress.



Examples of Alternating

- Warming up vs. cruising vs. cooling down
- Doing vs. describing
- Doing vs. thinking
- Careful vs. quick
- Data gathering vs. data analysis
- Working with the product vs. reading about the product
- Working with the product vs. working with the developer
- Product vs. project
- Solo work vs. team effort
- Your ideas vs. other peoples' ideas
- Lab conditions vs. field conditions
- Current version vs. old versions
- Feature vs. feature
- Requirement vs. requirement
- Test design vs. execution
- Coverage vs. oracles
- Testing vs. touring
- Testing vs. resting
- Individual tests vs. general lab procedures and infrastructure
- Reaction vs. non-reaction
- Strategic vs. tactical

Alternating



What it looks like when you do it poorly:

- You struggle with the intrinsic exploratory nature of exploratory testing. All other skills and tactics suffer. Your test execution is one dimensional.

How could you get better at it:

- Write the polarities into your charters, focusing on a specific polarity for a complete test session. This does not exclude other polarities, just focuses on one. Use an egg timer to signal a time to change polarities.
- Vary the context of your testing. You can do this by pairing with people you don't normally pair with (programmers, project managers, tech writers, that performance tester no body talks to, etc…), by testing applications you don't normally test (web apps, web services, desktop apps, embedded systems, utility apps, plug-ins, graphics applications, etc…), and by changing your context (startup, regulated, IT, software, maintenance, new development, etc…).

Refocusing

Managing the scope and depth of your attention. Looking at different things, looking for different things, in different ways.

Examples:

- Look at one feature, then switch your attention to another feature.
- Seeing a behavior that interests you in a new way, making a note of it, and then continuing in your current thread of testing.
- Seeing a behavior that interests you in a new way and dropping your current thread of testing to follow up on that new behavior.

Refocusing

What it looks like when you do it poorly:

- You don't know what you should be working on (chartering makes it relevant and organized, refocusing has to do with the very low level idea of thinking of this looking at that).
- Tunnel vision: only capable of looking at something in one and only one way.
- Compulsive refocusing: where you can't hold your attention on any one thing – you can't complete a thought or idea.

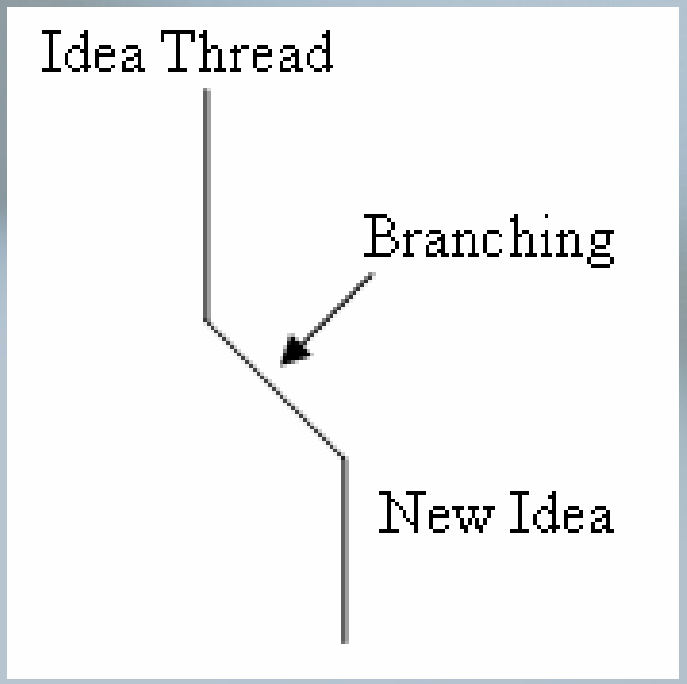
How could you get better at it:

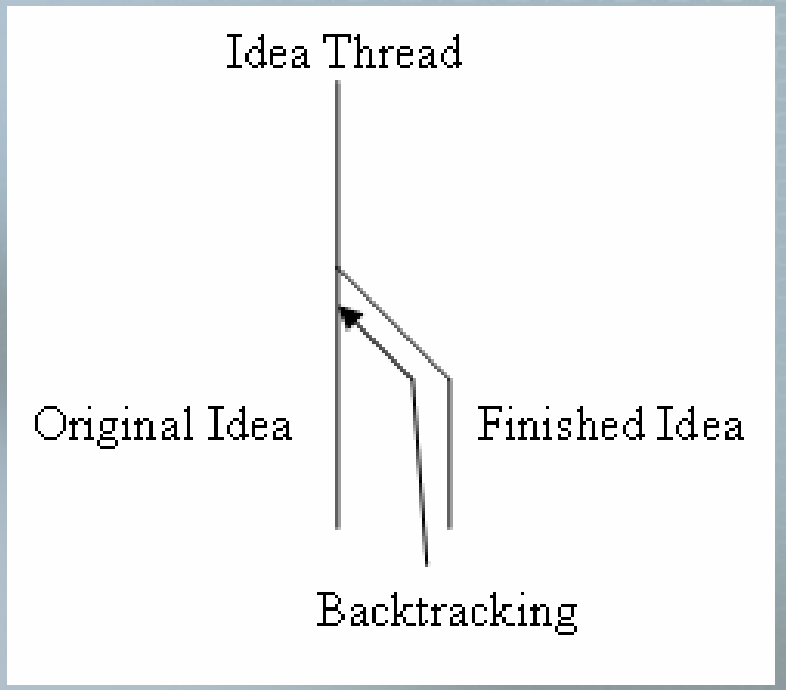
- Practice in situations that require you to switch focus (lots of complexity, lots of things going on, lots of features).
- Practice classic areas in testing where refocusing is necessary (like Blink testing…).
- Speeding up (put yourself under time pressure) – let go of a process and create a new process that can be done in the new timeframe.
- Do exercises in stating what the thread is: thread integrity.

Branching/Backtracking

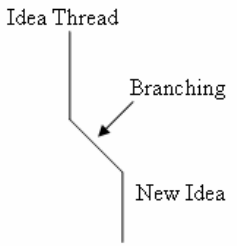
Allowing yourself to be productively distracted from one course of action in order to explore an unanticipated new idea. Identifying opportunities and pursuing them without losing track of the process.

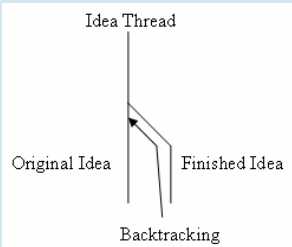
Branching/Backtracking Illustrated





Branching/ Backtracking





What it looks like when you do it poorly:

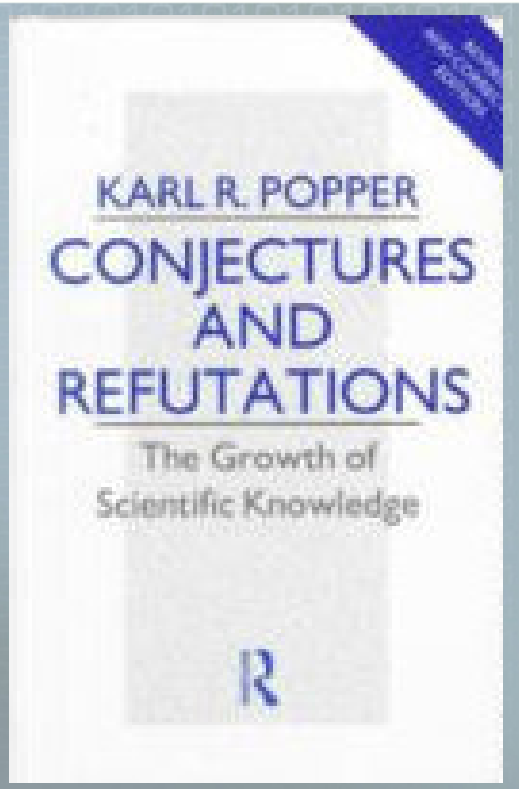
- At the end of your session, you /accidentally/ didn't fulfill your original mission. You either got lost or distracted. You didn't backtrack.
- During a debriefing session, you can't think of any new charters that you might want to run based on the testing you just did. No ideas occurred to you. You didn't get distracted at all. You didn't branch.
- While you are testing, the same idea keeps distracting you, you explore it a little, then you backtrack, only to get distracted by it again later. This continues for the entire test session. You were thrashing, over backtracking and over branching.

How could you get better at it:

- Condense the mission of your testing down to a post-it note and place that on the monitor, over a portion of the screen. It should distract you enough to keep you looking at it. Focus on THAT mission.
- Give yourself test sessions with no mission. Encourage distraction. Swab the application in your intuition. I call these learning tours.
- Pair with someone else. Talk to them about your thread. Let them challenge it. Let them force you to branch or backtrack when for you it doesn't seem natural to do so.

Conjecturing

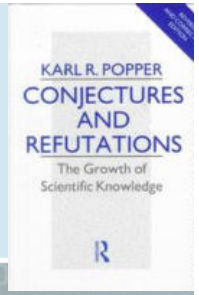
Considering possibilities and probabilities. Considering multiple, incompatible explanations that account for the same facts.



Examples:

- "Classic" conjecture and refutation. "This system is secure. Now prove that it's not."
- Prioritizing a risk list.
- Seeing a defect (a null pointer exception appears on the server log) and identifying five different ways it could have gotten there.

Conjecturing



What it looks like when you do it poorly:

- You can't imagine multiple possibilities for a behavior: “When does 2+2 not equal 4?”
- You fall victim to assimilation bias: The tendency to resolve discrepancies between your conjecture and new information by assimilating the information to fit your conjecture.
- You fall victim to confirmation bias: The tendency to selectively search for and gather evidence that is consistent with your conjecture. You ignore refutation.

How could you get better at it:

- Practice verbalizing your conjectures, then try to refute them. (Use classic games like twenty questions or card games like "Art Show" or explore magic tricks.)
- Read about philosophy and economics. For example, Karl Popper and Freakonomics.
- Whenever you find a bug, imagine three reasons for why it might *not* be a bug and three different failures that could have manifested themselves in that way.

Exercise: Greenland

These are the skills that comprise professional exploration of technology that can be observed, measured, and improved.

Thinking Testing

Generating/Elaborating

Overproduction/Abandonment

Abandonment/Recovery

Refocusing

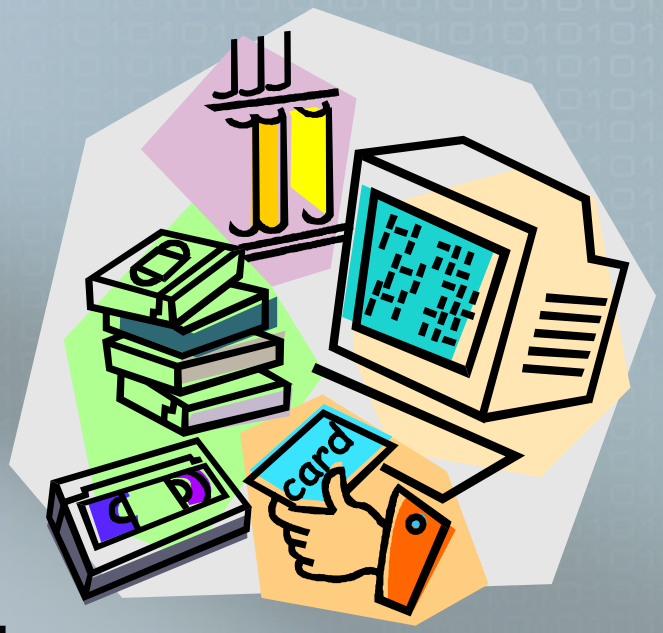
Alternating

Branching/Backtracking

Conjecturing

Recording

Preserving information about your process, progress, and findings.
Taking notes.



Examples:

- Using a capture tool like BB TestAssistant.
- Keeping notes in note-pad while you're testing.
- Carrying a moleskin around with you.

Recording

What it looks like when you do it poorly:

- Someone reading your reports can't tell a compelling story of your testing.
- Your work is not auditable to the degree required for the context of the project.
- It's difficult for someone to scan your records quickly to find the information then want.

How could you get better at it:

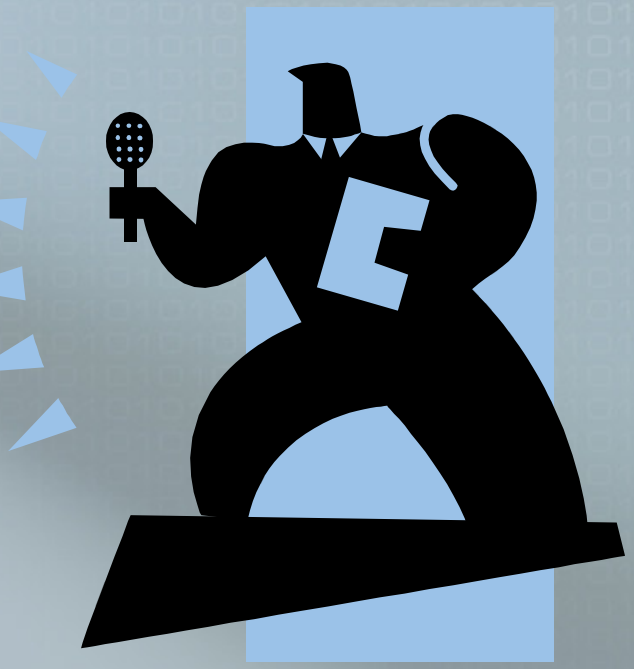
- Practice using tools like Spector Pro, BB TestAssistant, and SnagIt for test recording.
- Immediately after your testing, take your notes and answer the following questions using only your notes:
  1. What was I testing?
  2. Why was I testing?
  3. How did I test it?
  4. What data did I use?
  5. Where did I test it?
  6. What did I find?
  7. What did I not test?
  8. What, if I had it, would allow me to do more or better testing?
- Experiment taking your notes with different mediums (audio, video, paper, digital) with different formats (text only, pictures, mind maps, models, spreadsheets, templates, etc...).

Reporting

Making a credible, professional report of your work to your clients in oral and written form.

Examples:

- Telling the story of your progress so far.
- Writing a test summary report.
- The hallway conversation status report.



Reporting

What it looks like when you do it poorly:

- Your report is not relevant to your client.
- Your report doesn't go into enough detail, or goes into too much detail.
- Your report is not actionable.

How could you get better at it:

- Practice using MCOASTER (or MOST CARE) to frame your report (or develop a better model for your context).
- Partner with someone you work with (ideally several people) and at random points throughout the day ask each other for a quick one-minute or five-minute status report.
- Write up a status report at the end of each day and send it to your team members for review and feedback.
  - Can they understand what you did and didn't do?
  - Do they know why you did it?
  - Can they tell what techniques you used?

Exercise: Greenland

These are the skills that comprise professional exploration of technology that can be observed, measured, and improved.

Communicating Testing

Recording

Reporting

Exercise: Bullets and Numbering

These are the skills that comprise professional exploration of technology that can be observed, measured, and improved.

- **Modeling**
- **Resourcing**
- **Chartering**
- **Questioning**
- **Observing**
- **Manipulating**
- **Collaboration**
- **Generating/Elaborating**
- **Overproduction/Abandonment**
- **Abandonment/Recovery**
- **Refocusing**
- **Alternating**
- **Branching/Backtracking**
- **Conjecturing**
- **Recording**
- **Reporting**

Part 2

Evolving Work Products

Evolving Work Products

Exploratory testing spirals upward toward a complete and professional set of test artifacts. Look for any of the following to be created or refined during an exploratory test session.

- **Test Ideas:** Tests, test cases, test procedures, or fragments thereof.
- **Testability Ideas:** How can the product be made easier to test?
- **Bugs:** Anything about the product that threatens its value.
- **Risks:** Any potential areas of bugginess or types of bug.

Evolving Work Products (cont)

- **Issues:** Any questions regarding the test project, or matters to be escalated.
- **Test Coverage Outline:** Aspects of the product we might want to test.
- **Test Data:** Any data developed for use in tests.
- **Test Tools:** Any tools acquired or developed to aid testing.
- **Test Strategy:** The set of ideas that guide our test design.
- **Test Infrastructure and Lab Procedures:** General practices or systems that provide a basis for excellent testing.

Evolving Work Products (cont)

- **Test Estimation:** Ideas about what we need and how much time we need.
- **Test Process Assessment:** Our own assessment of the quality of our test process.
- **Testing Narrative:** The story of our testing so far.
- **Tester:** The tester evolves over the course of the project.
- **Test Team:** The test team gets better, too.
- **Developer Relations:** As you test, you also get to know the developer.

What work products did we see in the boundary testing example?

- ❑ Test Ideas
- ❑ Testability Ideas
- ❑ Bugs
- ❑ Risks
- ❑ Issues
- ❑ Test Coverage Outline
- ❑ Test Data
- ❑ Test Tools
- ❑ Test Strategy
- ❑ Test Infrastructure and Lab Procedures
- ❑ Test Estimation
- ❑ Test Process Assessment
- ❑ Testing Narrative
- ❑ Tester
- ❑ Test Team
- ❑ Developer Relations

Part 3

Testing Considerations

Disclaimer

Oh yea…

I know, I just put this section in here for completeness…

- However, these are useful to help you test robustly or evaluate someone else's testing.
- These heuristics (and heuristics like them) are central to your ability to perform the skills and tactics *quickly*. How fast do ideas occur to you?
- This is a compressed version of the Satisfice Heuristic Test Strategy model.

Project Environment

- **Customers:** Anyone who is a client of the test project.
- **Information:** Information about the product or project that is needed for testing.
- **Developer Relations:** How you get along with the programmers.
- **Test Team:** Anyone who will perform or support testing.
- **Equipment & Tools:** Hardware, software, or documents required to administer testing.
- **Schedules:** The sequence, duration, and synchronization of project events.
- **Test Items:** The product to be tested.
- **Deliverables:** The observable products of the test project.

Product Elements

- **Structure:** Everything that comprises the physical product.
- **Functions:** Everything that the product does.
- **Data:** Everything that the product processes.
- **Platform:** Everything on which the product depends (and that is outside your project).
- **Operations:** How the product will be used.
- **Time:** Any relationship between the product and time.

Quality Criteria Categories

- **Capability:** Can it perform the required functions?
- **Reliability:** Will it work well and resist failure in all required situations?
- **Usability:** How easy is it for a real user to use the product?
- **Security:** How well is the product protected against unauthorized use or intrusion?
- **Scalability:** How well does the deployment of the product scale up or down?
- **Performance:** How speedy and responsive is it?
- **Installability:** How easily can it be installed onto it target platform?
- **Compatibility:** How well does it work with external components & configurations?
- **Supportability:** How economical will it be to provide support to users of the product?
- **Testability:** How effectively can the product be tested?
- **Maintainability:** How economical is it to build, fix or enhance the product?
- **Portability:** How economical will it be to port or reuse the technology elsewhere?
- **Localizability:** How economical will it be to publish the product in another language?

General Test Techniques

- **Function Testing:** Test what it can do.
- **Domain Testing:** Divide and conquer the data.
- **Stress Testing:** Overwhelm the product.
- **Flow Testing:** Do one thing after another.
- **Scenario Testing:** Test to a compelling story.
- **Claims Testing:** Verify every claim.
- **User Testing:** Involve the users.
- **Risk Testing:** Imagine a problem, then find it.
- **Automatic Testing:** Write a program to generate and run a zillion tests.

Thank You

Are there any questions on anything we've covered?

Mike@MichaelDKelly.com