copenhagen2017

michael@developsense.com

Transforming Trainees Into Testers

Michael Bolton
DevelopSense

EuroSTAR Software Testing Conference 25

The Goals

- Take 18 people who have never been testers and prepare them for testing jobs
- Focus on crisp, effective bug reporting
- Develop testing skills that will be useful in as many contexts as possible.

We develop skill and tacit knowledge in three ways.
Only one way is through explanations,
and that may be the weakest way.

People forget that even though we would prefer not to make mistakes, mistakes are normal when we're learning to do difficult things.

We learn far more, and more quickly, from our own practice, mistakes, and feedback. Many process models fail to recognize feedback loops.

We also learn from being immersed in a culture where people are doing the same kinds of things. Many process models fail to recognize this.

Teaching Principles

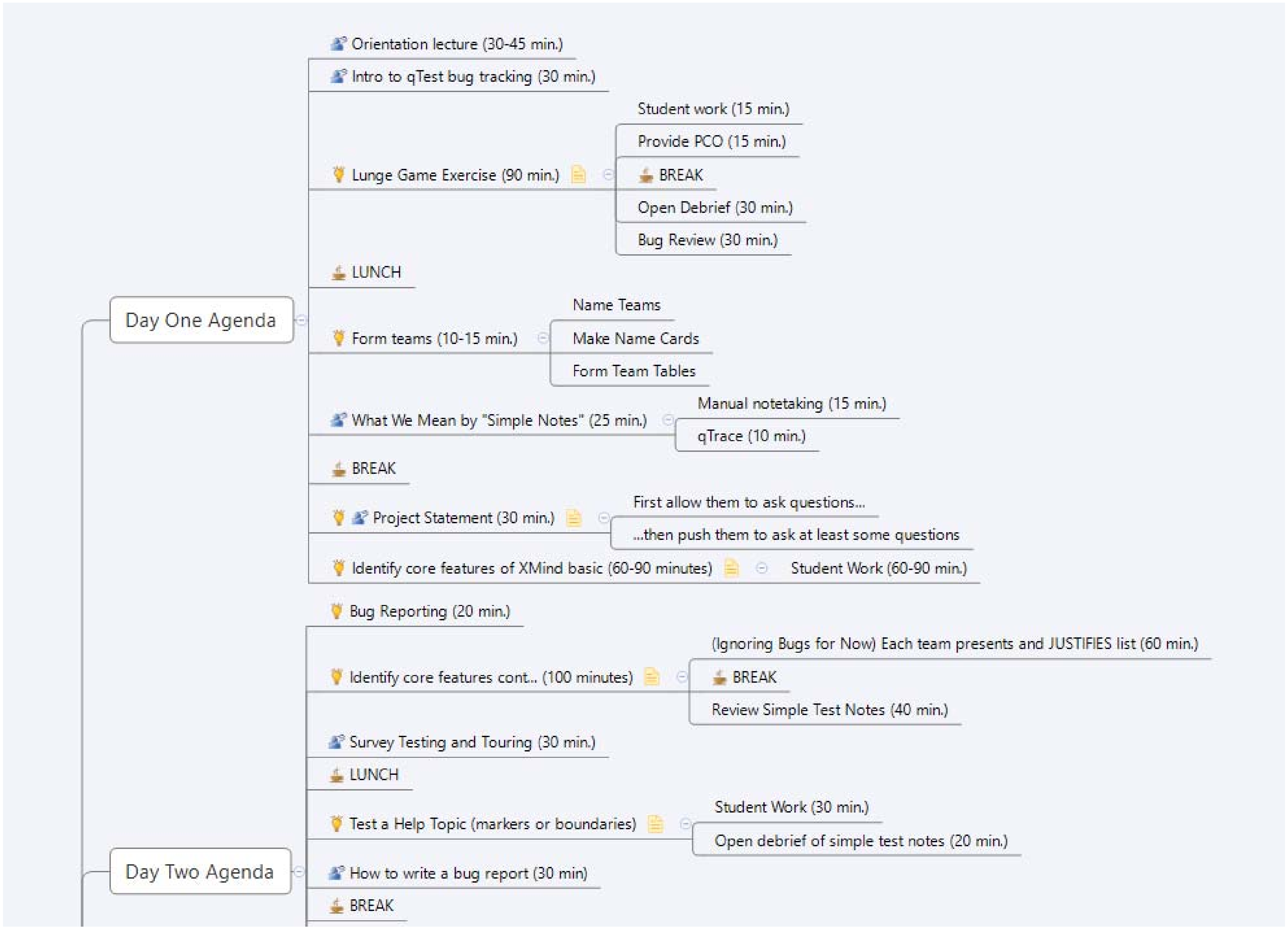
- Naturalistic experiential exercises
- Learn to test by *trying* to test
- Emphasize
  - Rapid learning
  - Recording and reporting
  - Developing models of products, testing, and risk
  - Developing test ideas and checklists
  - Collaboration
- Immediate and constant feedback
  - moving to peer review before instructor review
- Start easy, get harder quickly
- Support after the classes from our network
  - online coaching via Skype, email, etc.

Teaching Principles

- De-emphasize…
  - Paperwork
  - Specific techniques
  - Formal preparation
  - Following scripts and test cases
  - Solo work
  - Bureaucracy

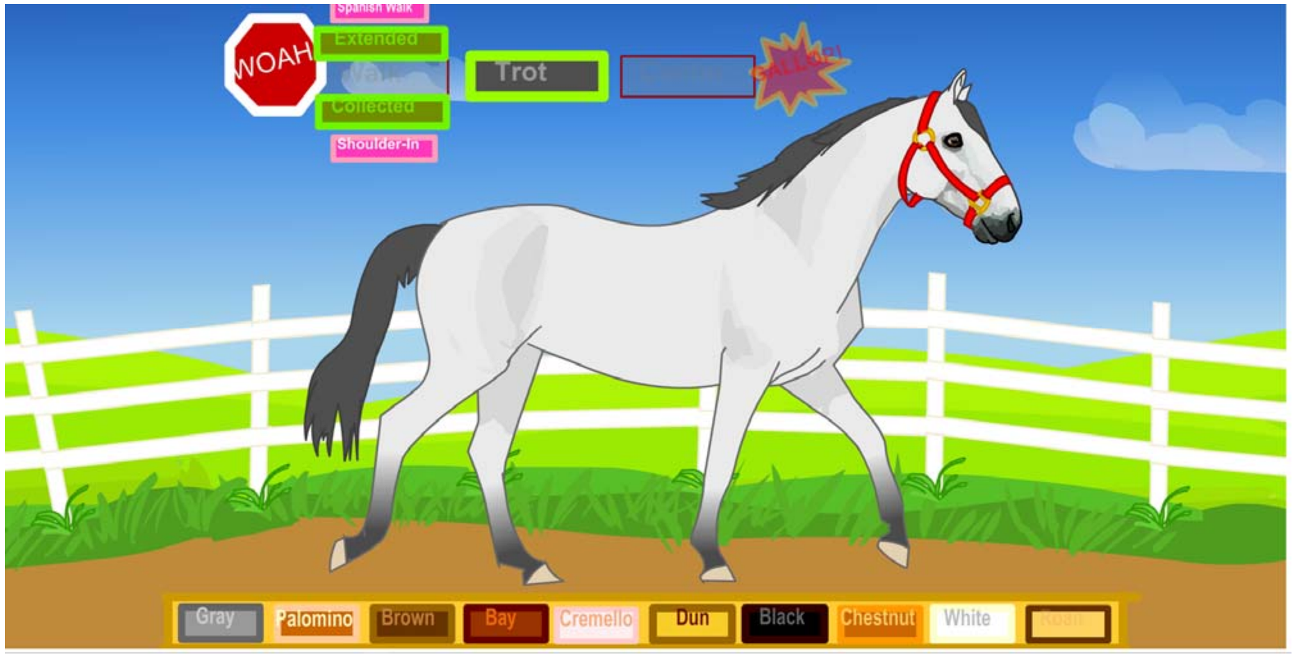
The General Premise for RST

- You have something to test.
- You have had no opportunity to prepare.
- If there are bad bugs, your clients need to know, and time is limited. But, except for you, nobody knows how to test.
- You must prepare starting now, just enough, using any available help, to learn about the product quickly, then apply useful tools to get into to deep testing.
- You must do all that while looking like you know what you are doing, and ACTUALLY knowing what you are doing.



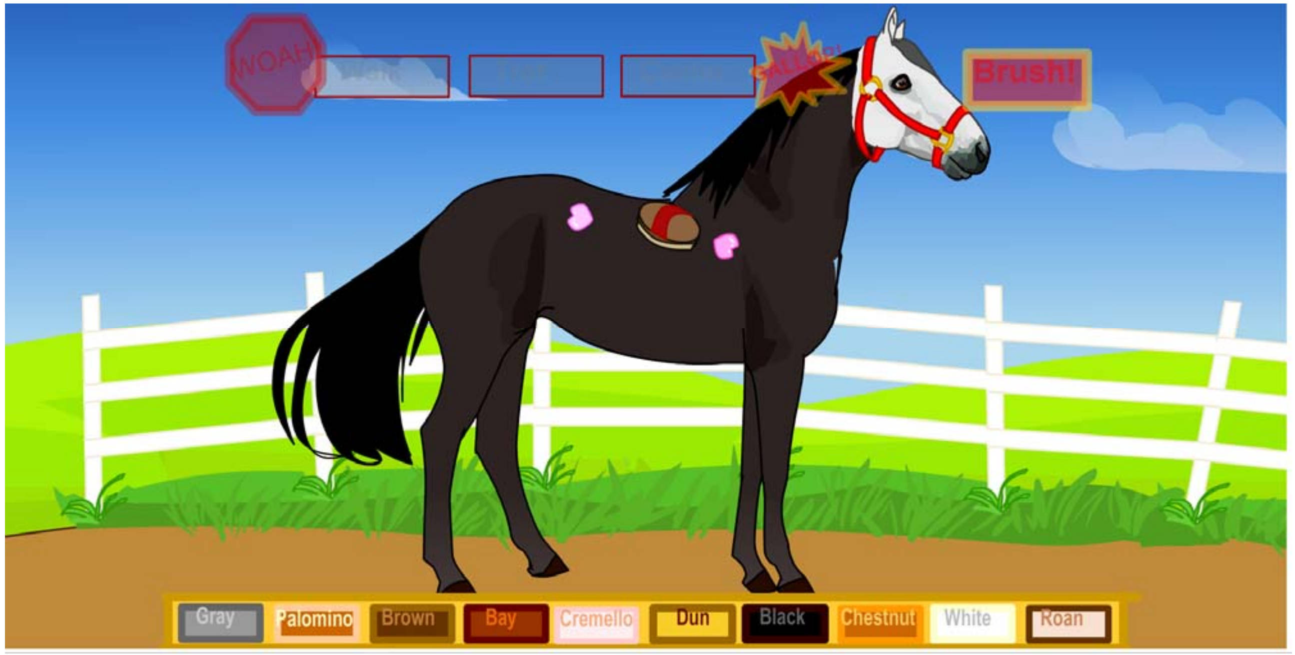
- Day One Agenda
  - Orientation lecture (30-45 min.)
  - Intro to qTest bug tracking (30 min.)
  - Lunge Game Exercise (90 min.)
    - Student work (15 min.)
    - Provide PCO (15 min.)
    - BREAK
    - Open Debrief (30 min.)
    - Bug Review (30 min.)
  - LUNCH
  - Form teams (10-15 min.)
    - Name Teams
    - Make Name Cards
    - Form Team Tables
  - What We Mean by "Simple Notes" (25 min.)
    - Manual notetaking (15 min.)
    - qTrace (10 min.)
  - BREAK
  - Project Statement (30 min.)
    - First allow them to ask questions...
    - ...then push them to ask at least some questions
  - Identify core features of XMind basic (60-90 minutes)
    - Student Work (60-90 min.)
- Day Two Agenda
  - Bug Reporting (20 min.)
  - Identify core features cont... (100 minutes)
    - (Ignoring Bugs for Now) Each team presents and JUSTIFIES list (60 min.)
    - BREAK
    - Review Simple Test Notes (40 min.)
  - Survey Testing and Touring (30 min.)
  - LUNCH
  - Test a Help Topic (markers or boundaries)
    - Student Work (30 min.)
    - Open debrief of simple test notes (20 min.)
  - How to write a bug report (30 min)
  - BREAK

First Exercise: The Horse Lunge Game



“As you see, this is a horse game for kids. The author wants this game for her portfolio, to showcase her talents as an animator for kids’ games and educational software. She knows that there are a few glitches in the game and she wants to clean them up. Find and report every bug you can to help her do that.”

First Exercise: The Horse Lunge Game



- Lots of easy-to-find bugs, so lots of reports
  - Little *systematic* bug reporting
  - Lots of weak bug reports
  - We were able to tell them *how* the reports were weak

Generating Feedback

- Debrief questions
  - How did that feel?
  - Do you feel you tested the program?
  - Do you think you tested it enough?
  - What more should you do?
  - What are your best bugs?
- Tuning up
  - Emphasis on note-taking
  - Description of bug and issue
- Key Lesson
  - Shallow testing of shallow programs with shallow bugs isn’t too hard.

We want PROOF!
Expect to discuss these things during a debrief

- **P**ast
  - What happened during the session?
- **R**esults
  - What was achieved?
- **O**bstacles
  - What got in the way or slowed things down?
- **O**utlook
  - What's next? What remains to be done?
- **F**eelings
  - How does the tester feel about all this?

Our approach was to offer the testers freedom AND responsibility to learn about testing BY testing. In our view, people don't learn testing very well from scripts or canned examples. They learn by trying to test, not doing very well, getting AND giving feedback, and improving via practice.

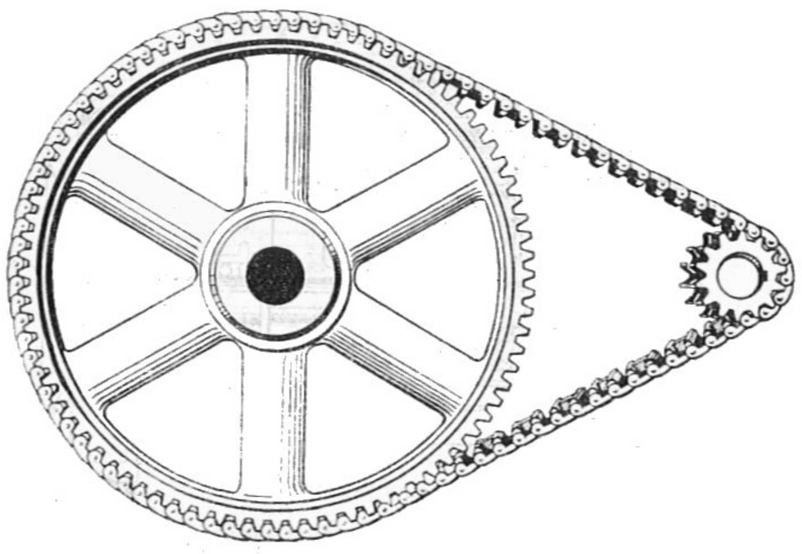
Why Do Freedom and Responsibility Matter?

Engagement and Judgment

The approach you use governs how your mind drives action– like a bicycle gear.

Match the rhythms of your mind...

(YOUR APPROACH)



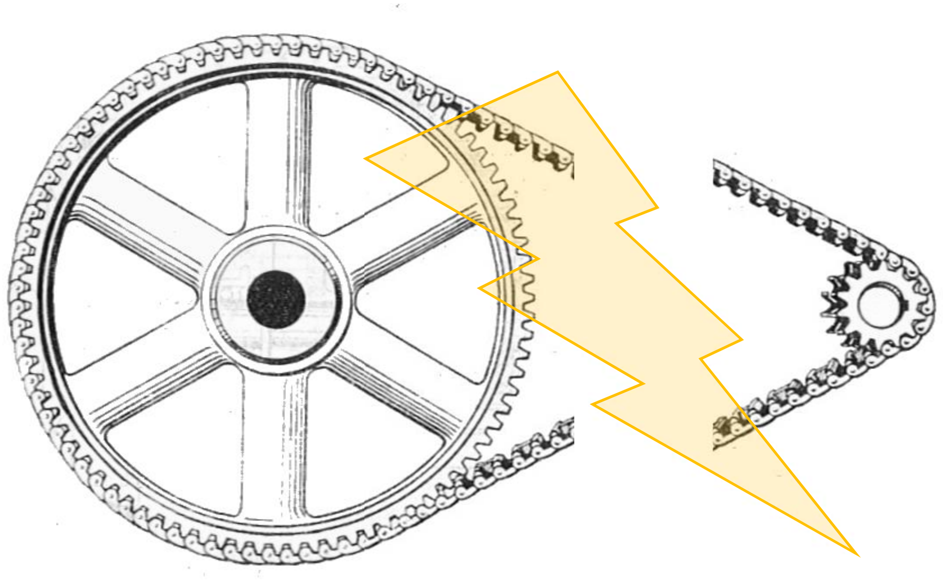
...to the structure of the testing.

(YOUR STRATEGY)

Why Do Freedom and Responsibility Matter?

Engagement and Judgment

When your mind is in the "wrong gear"…



You "stall." You "bounce off the problem."
or you "over-engineer the solution"

Why Do Freedom and Responsibility Matter?

Engagement and Judgment

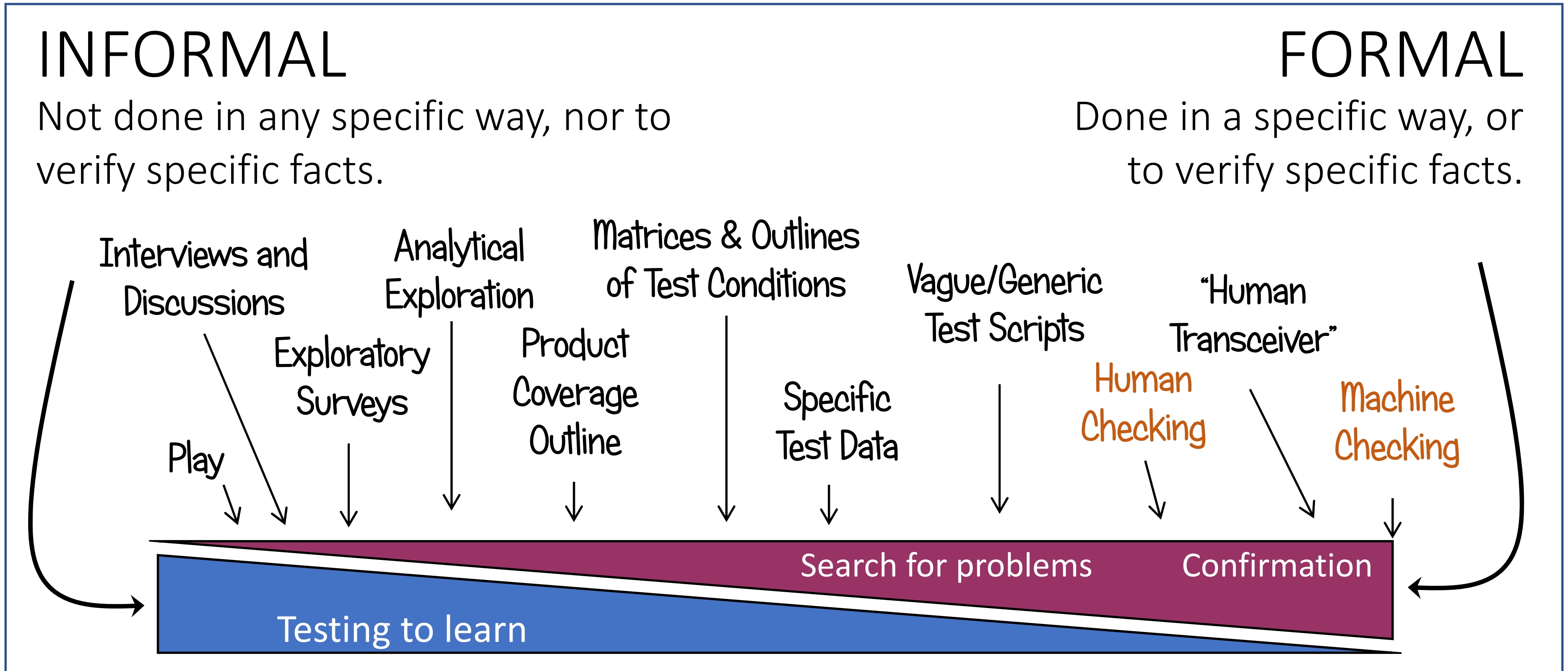
When your mind is in the "wrong gear"…

Alternate between deliberative and spontaneous testing styles.

Deliberative testing moves mountains.
Spontaneous testing jumps over them.

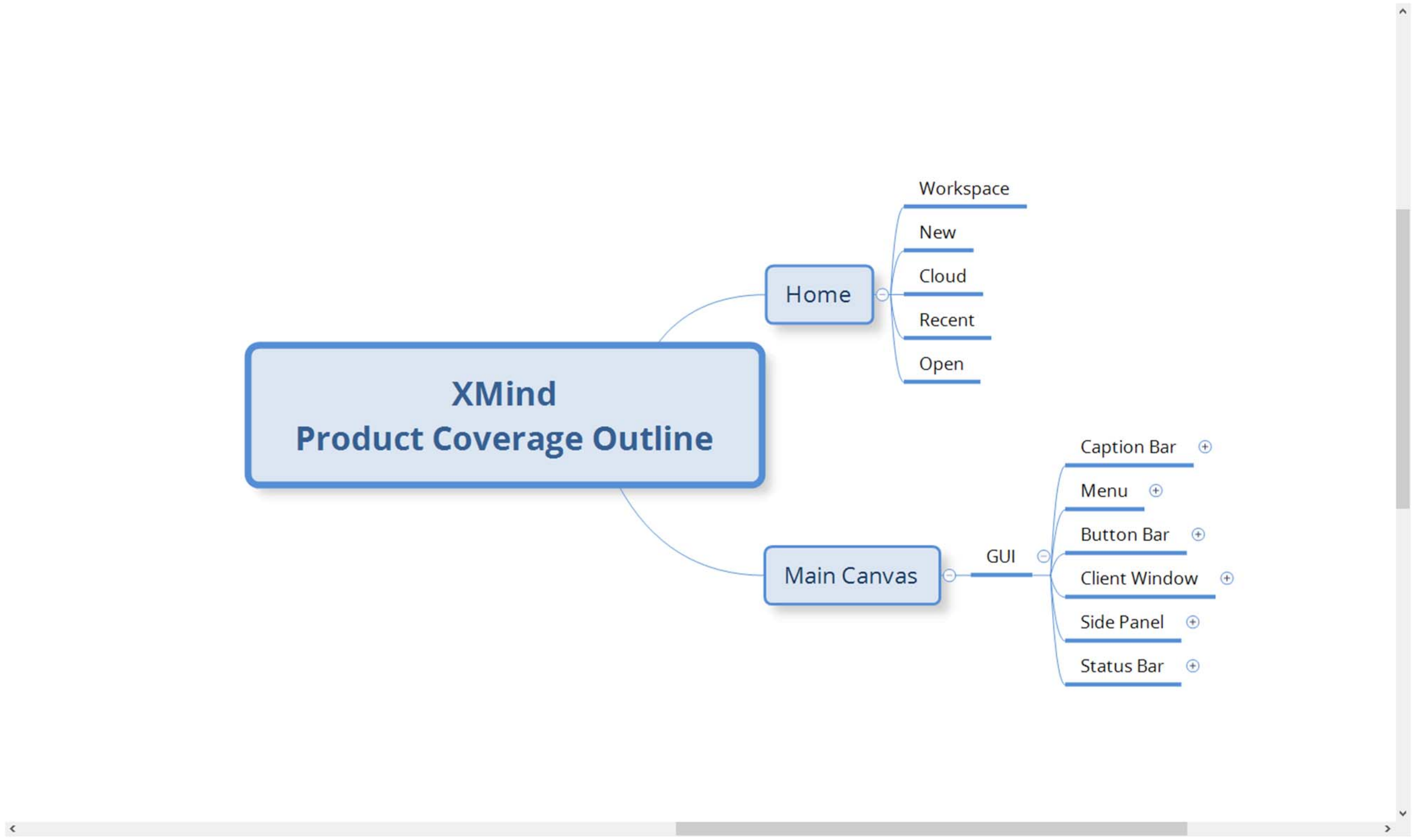
or you "over-engineer the solution"

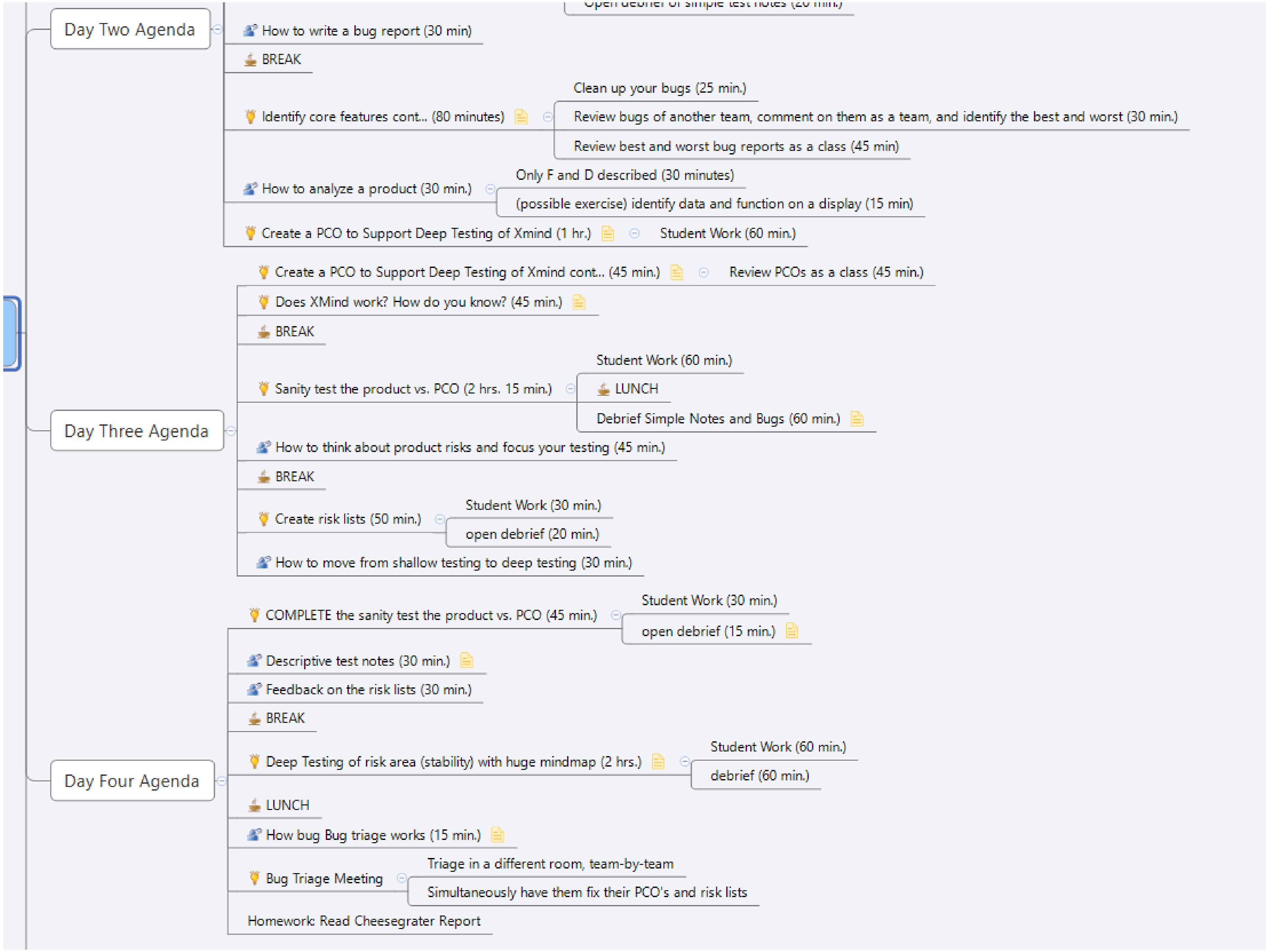
The Formality Continuum


INFORMAL
Not done in any specific way, nor to verify specific facts.
FORMAL
Done in a specific way, or to verify specific facts.
Interviews and Discussions
Analytical Exploration
Matrices & Outlines of Test Conditions
Vague/Generic Test Scripts
"Human Transceiver"
Exploratory Surveys
Product Coverage Outline
Specific Test Data
Human Checking
Machine Checking
Play
Search for problems
Confirmation
Testing to learn

Loops of testing start with informal, exploratory work. If you want to do excellent formal testing (like automated checking), it must begin with excellent informal work.

A Real Product: XMind





- Open debrief of simple test notes (20 min.)
- Day Two Agenda
  - How to write a bug report (30 min)
  - BREAK
  - Identify core features cont... (80 minutes)
    - Clean up your bugs (25 min.)
    - Review bugs of another team, comment on them as a team, and identify the best and worst (30 min.)
    - Review best and worst bug reports as a class (45 min)
  - How to analyze a product (30 min.)
    - Only F and D described (30 minutes)
    - (possible exercise) identify data and function on a display (15 min)
  - Create a PCO to Support Deep Testing of Xmind (1 hr.)
    - Student Work (60 min.)
- Day Three Agenda
  - Create a PCO to Support Deep Testing of Xmind cont... (45 min.)
    - Review PCOs as a class (45 min.)
  - Does XMind work? How do you know? (45 min.)
  - BREAK
  - Sanity test the product vs. PCO (2 hrs. 15 min.)
    - Student Work (60 min.)
    - LUNCH
    - Debrief Simple Notes and Bugs (60 min.)
  - How to think about product risks and focus your testing (45 min.)
  - BREAK
  - Create risk lists (50 min.)
    - Student Work (30 min.)
    - open debrief (20 min.)
  - How to move from shallow testing to deep testing (30 min.)
- Day Four Agenda
  - COMPLETE the sanity test the product vs. PCO (45 min.)
    - Student Work (30 min.)
    - open debrief (15 min.)
  - Descriptive test notes (30 min.)
  - Feedback on the risk lists (30 min.)
  - BREAK
  - Deep Testing of risk area (stability) with huge mindmap (2 hrs.)
    - Student Work (60 min.)
    - debrief (60 min.)
  - LUNCH
  - How bug Bug triage works (15 min.)
  - Bug Triage Meeting
    - Triage in a different room, team-by-team
    - Simultaneously have them fix their PCO's and risk lists
  - Homework: Read Cheesegrater Report

Second Exercise: XMind Survey

- *Our mission is to discover the quality status of the basic version of XMind.* (a real, popular mind mapping tool)
- *We must find critical bugs in XMind before this version is deployed in one week. We also must advise if there is any good reason to consider postponing the deployment of this product. We are also interested in non-critical bugs, but your preference should be to find critical problems QUICKLY and get them reported SOON.*
- *Remember, XMind doesn’t make money on the basic version. They make money when people upgrade to the Pro version.*
- *The “properties” functionality was rewritten in the latest release. But, there shouldn't be any problems because the functionality has not changed. The “markers” feature has just been added, though.*

Lessons

- Summarizing a Session
  - Your name, date/time, length of test session
  - Product version/environment
  - **Charter** statement for your test session (one or two sentences)
  - Any way in which you have **not** fulfilled your charter
  - Any bugs you found (in this case put them in qTest, but include the bug titles and IDs in your notes)
  - Any questions or issues that came up for you
- Bugs and Issues
  - A bug is anything that threatens the value of the product
    - informally, a bug is something that bugs somebody who matters
  - An issue is anything that threatens the value of our testing (or of the project, or of the busines)
    - most of the time, an issue is something that makes testing harder or slower

XMind Survey

- Trap
  - Getting started without questioning the mission
- Outcome
  - Shallow notes and mind maps
  - Shallow analysis
  - Still poor bug reports
  - Teams did some review of their own work, but with little grounding for it
- Feedback
  - Sharper feedback from instructors
- Lessons
  - Testing will be shallow if you don't know the product.
  - Deeper testing of deeper programs with deeper bugs is harder.

Mapping the Product

- Mind maps helped to illustrate students' developing skills and models
- Peer review seemed to greatly sharpen the quality of the work and of the artifacts; two-stage peer review even more so.

You Found a Bug!

- Before you report it, RIMGEA!
- Replicate it
  - try to reproduce it; if you can't, gather and record clues
- Isolate it
  - minimize the steps to reproduce it in the easiest possible way
- Maximize
  - see if you can make it trigger a worse failure
  - vary behaviour, program settings, inputs, or platforms
- Generalize it
  - "uncorner the corner cases"
- Externalize it
  - Consider how it might impact other stakeholders
- And then…
  - report it clearly and dispassionately

See the Black Box Software Testing Bug Advocacy material at
http://www.testingeducation.org/BBST/bugadvocacy/

Bug Title (Summary) Guidelines

- State the problem first, then conditions
  - What Bad Thing happened? or
  - What Good Thing failed to happen?
  - When and where did it happen/not happen?
- Be concise
- Be specific, not generic
- Identify the business risk
  - a bug doesn't matter if it doesn't threaten value to some person who matters
- Express yourself dispassionately

Bug Description Guidelines

- State the problem.
- State an example of the problem UNLESS an example is obvious from the problem statement
- Say specifically how to make it happen UNLESS a description is obvious
- Put the steps in a numbered list UNLESS there are only one or two steps.
- State your oracle (WHY you think it's a bug).
- If there is a workaround or an exception to the problem, say so.

A Mnemonic for Rapid Bug Reporting: PEOPLE WORKing

- A **P**roblem that you've observed.
- An **E**xample to illustrate the problem
  - include steps, data, circumstances, or platform (only) if needed
- An **O**racle
  - An oracle is "a way to recognize a problem" (or to describe one, after you've recognized it)
  - Oracles are often linked to a risk or to a quality criterion that is threatened.
- **P**olite
  - Nobody really likes bad news. Egos are on the line.
- **L**iterate
  - A bug report is a story about an interaction between a person and a product or feature. Tell that story.
- **E**xtrapolation
  - How could this problem be more general, and/or more extreme?
- A **Work**around
  - Maybe there is one that reduces the severity or impact of the problem.

A Checklist to Sharpen Bug Reporting PEOPLE WORKing

- A **P**roblem that you’ve observed.
- An **E**xample to illustrate the problem
- An **O**racle
- **P**olite
- **L**iterate
- **E**xtrapolation
- A **Work**around

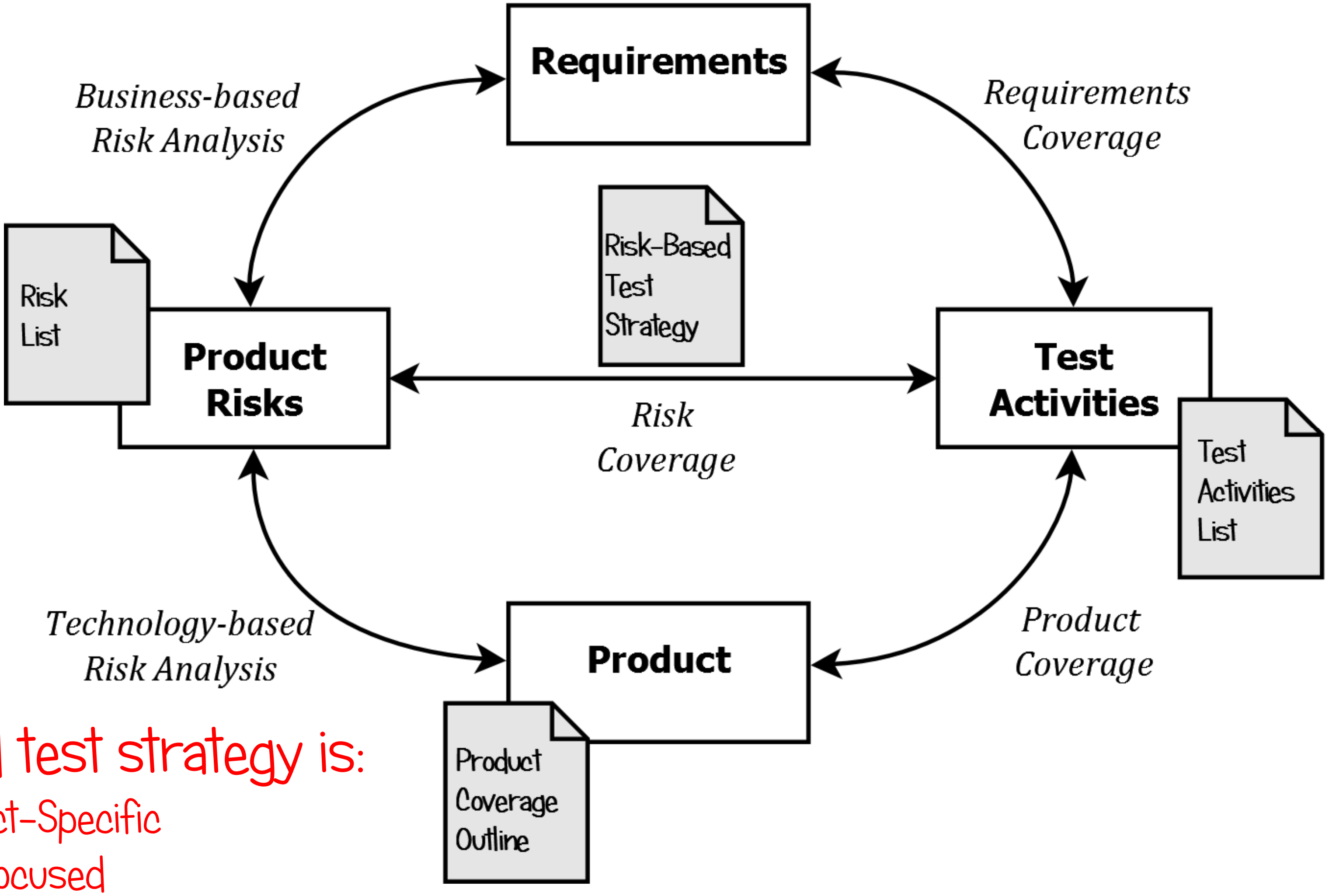
Your bug report doesn’t have to follow this pattern (although it should almost certainly include the first three, and a workaround if there is one). But you might find it helpful to consider these points when you’re writing or evaluating a bug report.

The Four-Part Risk Story

- There is a *potential* that…
- some **VICTIM…**
  (a problem cannot exist without a person to experience it)
- will experience a **PROBLEM…**
  (such as loss, harm, annoyance, or diminished value)
- because of a **VULNERABILITY** in the product...
  (such as a bug, a missing feature, or an issue)
- that is triggered by some **THREAT.**
  (even if the vulnerability is there, if it’s never triggered, then the problem won’t be experienced by a person)

All four factors must be present.

Risk Is A Key Driver in Your Test Strategy



A good test strategy is:

- Product-Specific
- Risk-focused
- Diversified
- Practical

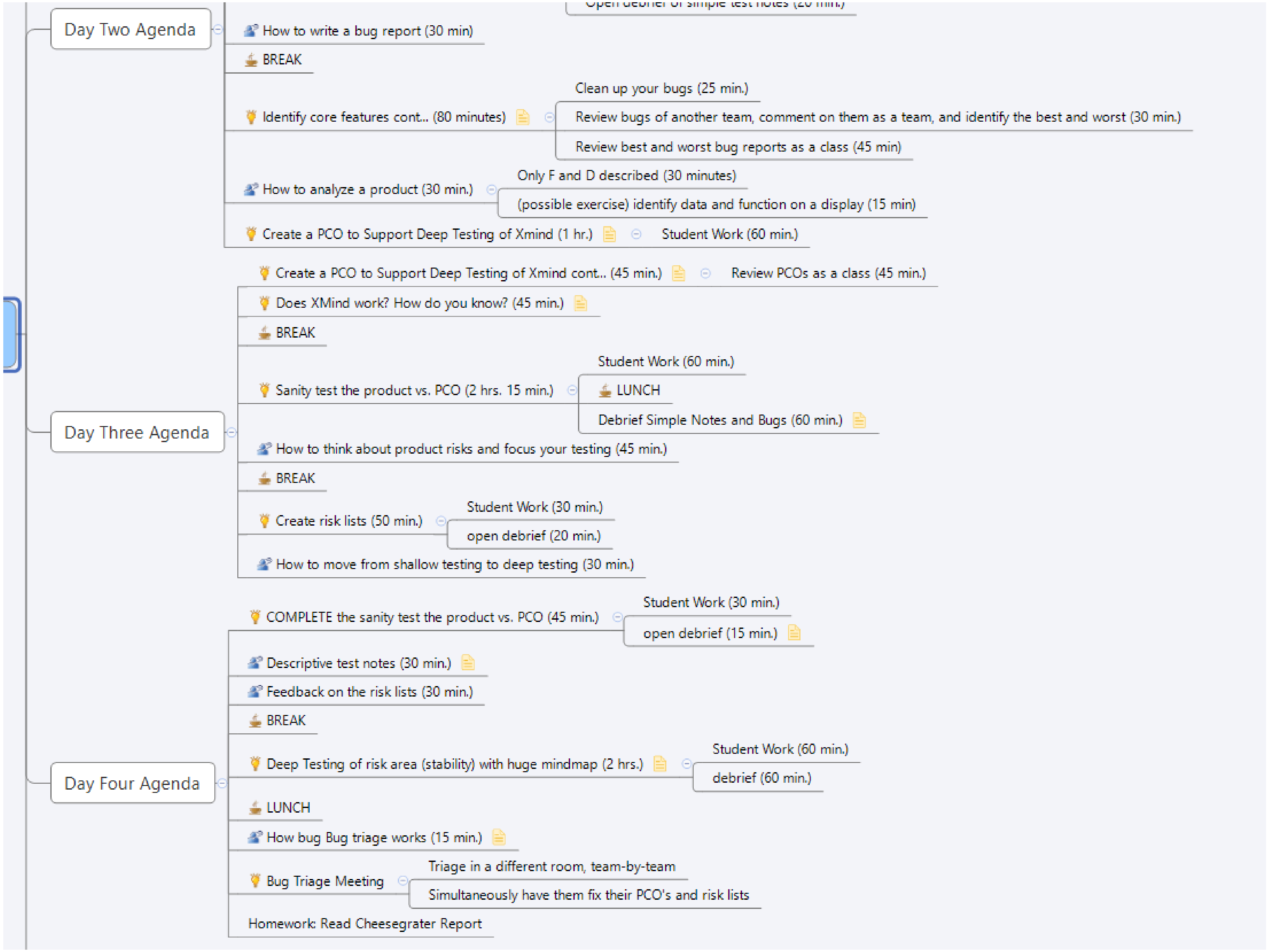
Assignment: Deep Testing

We have determined that there is some evidence pointing to instability in XMind. Therefore…

Perform stress testing on XMind. Cover a wide variety of features and functions. You may take advantage of the random1.xmind and random2.xmind files that have been generated for testing purposes.

As usual, each team must produce:

- *simple test notes*
- *progressive improvement on the depth and breadth of your PCO*
- *bug reports*



- Open debrief of simple test notes (20 min.)
- Day Two Agenda
  - How to write a bug report (30 min)
  - BREAK
  - Identify core features cont... (80 minutes)
    - Clean up your bugs (25 min.)
    - Review bugs of another team, comment on them as a team, and identify the best and worst (30 min.)
    - Review best and worst bug reports as a class (45 min)
  - How to analyze a product (30 min.)
    - Only F and D described (30 minutes)
    - (possible exercise) identify data and function on a display (15 min)
  - Create a PCO to Support Deep Testing of Xmind (1 hr.)
    - Student Work (60 min.)
- Day Three Agenda
  - Create a PCO to Support Deep Testing of Xmind cont... (45 min.)
    - Review PCOs as a class (45 min.)
  - Does XMind work? How do you know? (45 min.)
  - BREAK
  - Sanity test the product vs. PCO (2 hrs. 15 min.)
    - Student Work (60 min.)
    - LUNCH
    - Debrief Simple Notes and Bugs (60 min.)
  - How to think about product risks and focus your testing (45 min.)
  - BREAK
  - Create risk lists (50 min.)
    - Student Work (30 min.)
    - open debrief (20 min.)
  - How to move from shallow testing to deep testing (30 min.)
- Day Four Agenda
  - COMPLETE the sanity test the product vs. PCO (45 min.)
    - Student Work (30 min.)
    - open debrief (15 min.)
  - Descriptive test notes (30 min.)
  - Feedback on the risk lists (30 min.)
  - BREAK
  - Deep Testing of risk area (stability) with huge mindmap (2 hrs.)
    - Student Work (60 min.)
    - debrief (60 min.)
  - LUNCH
  - How bug Bug triage works (15 min.)
  - Bug Triage Meeting
    - Triage in a different room, team-by-team
    - Simultaneously have them fix their PCO's and risk lists
  - Homework: Read Cheesegrater Report

Last Assignment (Day 5)

- We are working on Microsoft Word. The program manager is concerned today about possible problems in the Insert/Page Number/Format Page Numbers dialog.
- In one hour, examine that dialog with a partner.
- We want the highest-quality bug report about the single most important bug or issue that you can find in the dialog.
- You should also mail test notes to the product owner.

Outcomes

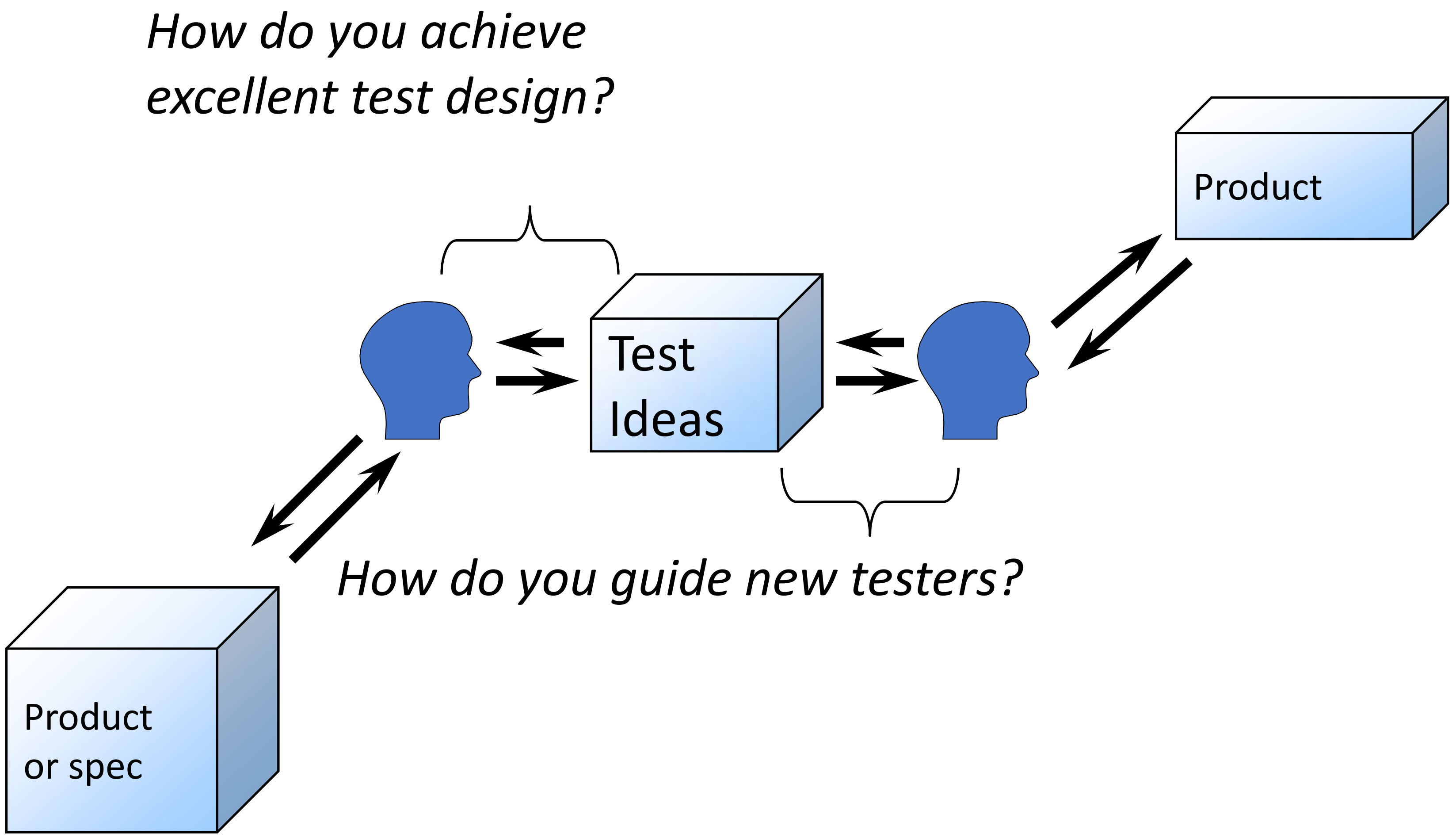
- The students were still struggling with creating a coverage model
- The bug reports were getting MUCH better
  - As I group, we produced 10 bug reports. Mine was fairly judged to be the *second* best of the lot.
- *The students were beginning to exercise skill and judgment for self-direction* which, when supported by an experienced test manager, gave them a running start.

Outcomes

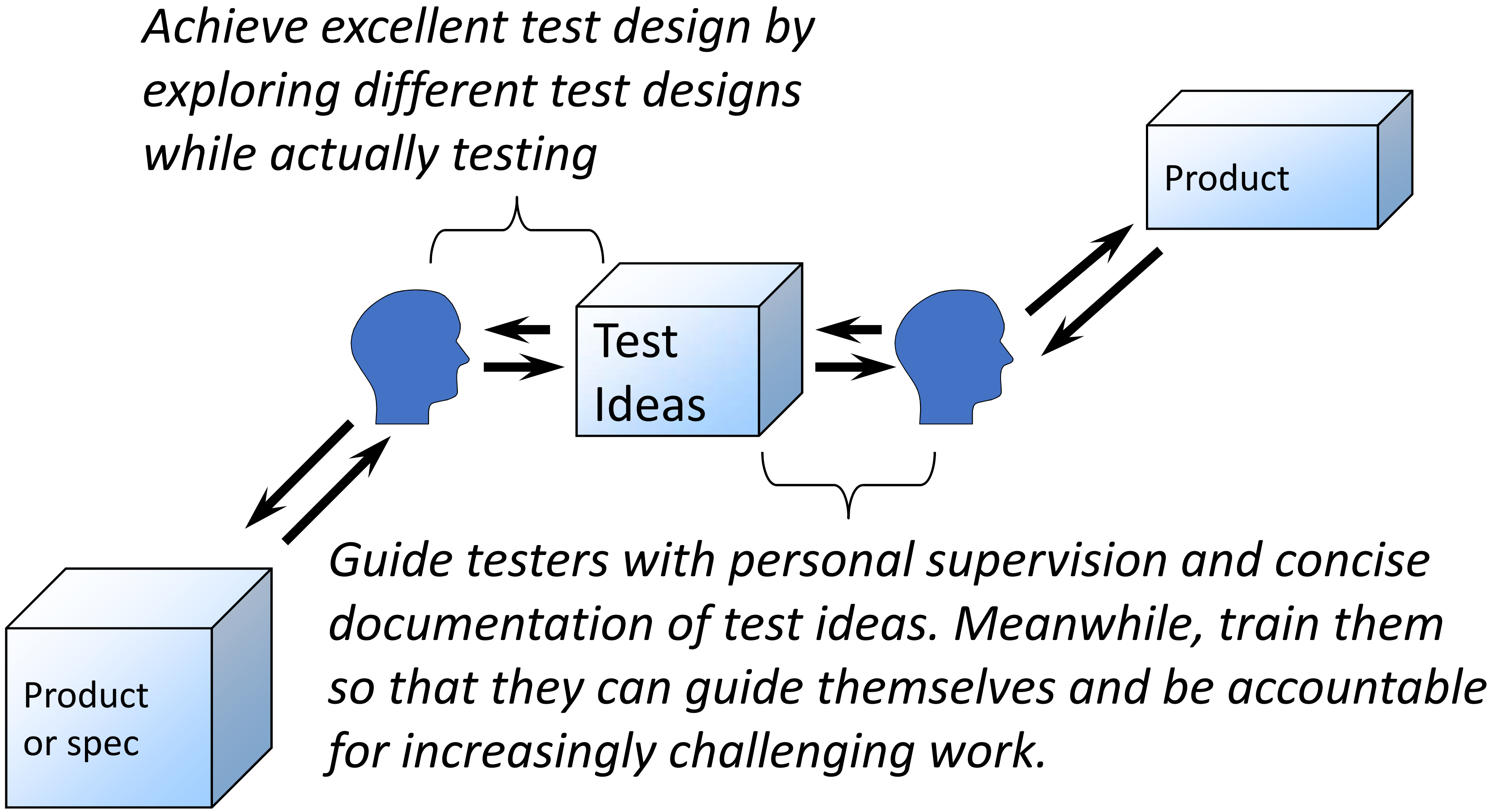
- Approximately 70% of the participants in the overall program are still in IT.
- According to some people who placed them, the trainees were actually *over*prepared compared to the rest of the market
  - in particular, they interviewed well.
- Because curriculum change was easy, adaptation to new ideas and tools was easy.
- Access to our professional network provided ready access to a lot of people
  - via Skype channels, coaching, aftercare, which trainees took advantage of

- Non-enterprise people didn’t adapt well to corporate drudgery.
  - The training program was too exciting!
  - Also, people would job-hop and quit more quickly because their experience didn't point them to long-term work.
- Some trainees didn't have the drive or determination to continue the learning, and dropped out.

Test Design and Execution


How do you achieve excellent test design?
Product
Test Ideas
How do you guide new testers?
Product or spec

Test Design and Execution


Achieve excellent test design by exploring different test designs while actually testing
Product
Test Ideas
Product or spec
Guide testers with personal supervision and concise documentation of test ideas. Meanwhile, train them so that they can guide themselves and be accountable for increasingly challenging work.