Exploratory Test Design

Let’s Test, Runö, 7-May-2012

Rikard Edgren
Qamcom Karlstad
rikard.edgren@qamcom.se
“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.”

[Cem Kaner]

Why is this good?

*at start we don’t know everything important*

*we want to know more*

*we will design tests and learn as we go*
Agenda

1. Test Analysis
   - Sources for Test Ideas

2. Test Design
   - Quality Characteristics

3. Test Execution
   - Modeling
Part 1: Test Analysis

- You always have requirements of some sort.
- Understanding and questioning them is a good start, but no more than that.

- It’s about learning, and finding out what is important.
- Exploratory Test Design looks at many places.
Sources For Test Ideas

1. Capabilities – requirements, examples et.al.
2. Failure Modes – “what if...” - question everything
3. Models – many, if invisible models count
4. Data – exploit dependencies
5. Surroundings – environment / granularity
6. White Box – developer perspective + tester mindset
7. Product History – error catalogues
8. Rumors – kill them or prove them right
9. Actual Software – gulp your Pommac
10. Technologies – things that tend to go wrong
11. Competitors – also in-house, analogue solutions
12. Purpose – benevolent start
13. Business Objectives – product vision, value drivers
14. Product Image – what should/would users think?
15. Business Knowledge – learn, or co-operate
16. Legal Aspects – what must be avoided?

17. Creative Ideas – products worth building are unique
18. Internal Collections – product-specific quicktests
19. You – you are a user, you matter
Exercise: Your Information Sources

- Write down one or two actual sources of information that improved your testing (do not use requirements!)

- Example:

  - Interaction designers gave me their background material they collected from real customers. It helped me design more realistic tests, and also to understand which types of problems that are important.
20. Project Background – what happened last time?
21. Information Objectives – the purposes of testing
22. Project Risks – test risky areas early
23. Test Artifacts – other’s testing
24. Debt – test against shortcuts
25. Conversations – people talk and collaborate
26. Context Analysis – what should effect your testing?
27. Many Deliverables – test objects and/or inspiration
28. Tools – a starting point for exploration
29. **Quality Characteristics** – in the back of your head
30. **Product Fears** – capture stakeholder’s worries
31. **Usage Scenarios** – what people want to do
32. **Field Information** – environment, needs, feelings
33. **Users** – some we like, some we don’t like

34. **Public Collections** – Appendix, Cheat Sheet, Not Done
35. **Standards** – read, understand, use...
36. **References** – as oracle and inspiration
37. **Searching** – Altavista, Volunia et.al.
Test Analysis Questions

- Yes, we have all of these, but what should we do with them?
- **Understand**, and use as appropriate
  - Some become straightforward test ideas
  - Some need a lot of elaborations
  - Some make other tests richer

- *We don’t even have time to do the requirements-based tests; how should we have time for all of these?*
- **Judgment**, some of these give more important information
  - skip the existing tests someone (you?) already has run
  - try a few that looks promising
  - change the ways you test from time to time
Can we get new, relevant information by running the test?

- **YES**
  - Is it really fast to execute and evaluate?
    - **NO**
      - Don’t run
    - **YES**
      - Run

- **NO**
  - Is it important, to someone?
    - **NO**
      - Don’t run
    - **YES**
      - Run
Part 2: Test Design

- A process of synthesizing test ideas.

- This involves:
  - many information sources
  - judgment of what’s important
  - experience and creativity
  - taking advantage of what we learn
  - practical limitations/opportunities

- The number one heuristic: Diverse half-measures
One-Liner Test Ideas

- Continuously jot down your test ideas in short format, leaving details to execution.
- Don’t necessarily think in terms of Pass/Fail.
- Let other people review to get more ideas, better prioritization (and skip test ideas that don’t seem important.)

- Use the list for planning
  - HOW and WHEN to test
  - When to apply techniques, and which to run without flourishes
  - Things to keep in the back of your head all the time
<table>
<thead>
<tr>
<th>Quality Characteristics</th>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Capability</strong></td>
<td>Can the product perform valuable functions?</td>
</tr>
<tr>
<td><strong>Reliability</strong></td>
<td>Can you trust the product in many and difficult situations?</td>
</tr>
<tr>
<td><strong>Usability</strong></td>
<td>Is the product easy to use?</td>
</tr>
<tr>
<td><strong>Charisma</strong></td>
<td>Does the product have &quot;it&quot;?</td>
</tr>
<tr>
<td><strong>Compatibility</strong></td>
<td>How well does the product interact with software and environments?</td>
</tr>
<tr>
<td><strong>Supportability</strong></td>
<td>Can customers’ usage and problems be supported?</td>
</tr>
<tr>
<td><strong>Performance</strong></td>
<td>Is the product fast enough?</td>
</tr>
<tr>
<td><strong>Testability</strong></td>
<td>Is it easy to check and test the product?</td>
</tr>
<tr>
<td><strong>Maintainability</strong></td>
<td>Can the product be maintained and extended at low cost?</td>
</tr>
<tr>
<td><strong>Security</strong></td>
<td>Does the product protect against unwanted usage?</td>
</tr>
<tr>
<td><strong>Portability</strong></td>
<td>Is transferring of the product to different environments and languages enabled?</td>
</tr>
</tbody>
</table>
Examples: Usability

- Is the product easy to use?
  - **Intuitiveness**: it is easy to understand and explain what the product can do.
  - **Learnability**: it is fast and easy to learn and remember how to use the product.
  - **Operability**: an experienced user can perform common actions very fast.
  - **Clarity**: is everything stated explicitly and in detail, with a language that can be understood, leaving no room for doubt?
  - **Errors**: there are informative error messages, difficult to make mistakes and easy to repair after making them.
  - **Control**: the user should feel in control over the proceedings of the software.
  - **Tailorability**: default settings and behavior can be specified for flexibility.
  - **Accessibility**: the product is possible to use for as many people as possible, and meets applicable accessibility standards.
  - **Documentation**: there is a Help that helps, and matches the functionality.
Examples: Reliability

- Can you trust the product in many and difficult situations?
  - **Stability**: Run the product for a long time, without restarts.
  - **Robustness**: Provoke error messages; hit hard, hit many times.
  - **Stress handling**: Exceed various limits.
  - **Recoverability**: Turn off the power; restart and look at behavior.
  - **Data Integrity**: Use all types of data, in different sizes on different systems through all parts.
  - **Safety**: Thorough brainstorming around areas where people can get hurt.
  - **Disaster Recovery**: Test that the back-up actually can restore everything.
  - **Trustworthiness**: Note moments when you are unsure what the product is up to.
Exercise: Important Characteristics

- This is an ongoing exercise for the rest of Let’s Test:
- Which characteristics are important for you?
- How do you test these?
Part 3: Test Execution

The Test Eye

- want to see problems
- see a lot of things
- look at many places
- look often
- focus on what’s important
- look for others
Variation & Complexity

- Running the same tests over and over again is good to spot things that break (regression testing)
- To find new information, you should perform new tests, or do variations on your old tests.

- Adding more complexity gives richer tests with bigger chance to find new, important information
  - Data
  - Environment
  - Sequences
  - Users
Test Execution Heuristics

- Boundary value analysis
- Error-Prone Machine
- Be ready for serendipity
- Do one more thing
- Fresh eyes find failure
- Follow the scent
Modeling

- It might not suffice to keep all the information in your head.
- A great framework for getting structure is to use SFDPOT from James Bach’s [Heuristic Test Strategy Model](#).

- **Structure** – what the product is
- **Functions** – what the product does
- **Data** – what the product operates on
- **Platform** – the environment the product depends on
- **Operations** – what the users want to accomplish
- **Time** – relations between the product and time
Modeling

- People have different ways of modeling, and that’s a strength.
- We need several models in order to capture what’s important to different users.

- Models can come from
  - developers
  - data
  - surroundings
  - usage
  - quality characteristics
  - your detailed knowledge about the product
Invisible Models

- Skilled testers make many models all the time.
- Your understanding (regardless of origin) is a sort of model.

- The diversity in models bring more, and richer test ideas.
- These don’t need to be documented, but it’s a good exercise.
Exercise: Your first models

- Have a look at the software I will show.
- How many models do you get?
Perfect Age Models

Long Time Ago → Modern Time → Now → Future

Detailed Timeline

- Beginning of time
  - Common start dates on computers
    - 0100-01-01
    - 1899-12-31

- Leap Years
  - 29 Feb every 4 years
  - Except divisible with 100
    - Except also divisible with 400
    - 1600
    - 1900
    - 2000
    - 2012

- Now
  - This day
  - This month
  - This year

- 9999-12-31
  - Common computer end of time
Perfect Age Models
Perfect Age Models

Perfect Age Calculator Quality Model

- Capabilities
  - Accuracy
  - Data agnosticism
- Reliability
  - Robustness
  - Data Integrity
- Usability
  - Intuitiveness
  - Operability
- Charisma
  - Professionalism
  - Attractiveness
- Security
  - Virus-free
- Performance
  - Resource utilization
- IT-ility
  - Installability
  - Uninstallation
- Compatibility
  - OS Compatibility
  - Standards Conformance
Testers’ Trait – Many Perspectives

- Testers bring new perspectives, that’s why we find important problems others don’t see.

- If you train your ability to think in many different ways, this doesn’t cost a lot of time...
  ...as long as you can judge what is important.
Find Five Faults

...but you don’t have a key, and don’t know how many problems there are
...and there might be other things that are much more important
Finale

- You are this generation’s thinking testers
- You have to find YOUR best ways
- Do your best, collaborate, learn to understand **what is important**
Questions

Further reading:
- [Exploratory Testing Dynamics](http://example.com) (Bach, Bach, Bolton)
- [BBST Test Design](http://example.com) (Kaner, Fiedler)
- [The Little Black Book on Test Design](http://example.com) (Edgren)

www.thetesteye.com  rikard.edgren@qamcom.se