Exploratory and Experience Based Testing

21.11.2011

Juha Itkonen
Aalto University School of Science
Department of Computer Science and Engineering
Contents

• Intelligent Manual Testing
  – Experience base testing
  – Exploratory testing

• Ways of Exploring
  – Session Based Test Management
  – Touring testing

• Intelligent Manual Testing Practices
  – Examples of empirically identified testing practices

• Benefits of Experience Based Testing
Manual Testing

- Testing that is performed by human testers
- Stereotype of manual testing
  - Executing detailed pre-designed test cases
  - Mechanically following the step-by-step instructions
  - Treated as work that anybody can do

Research has shown:
1. Individual differences in testing are high
2. Test case design techniques alone do not explain the results

In practice, it’s clear that some testers are better than others in manual testing and more effective at revealing defects...
Traditional emphasis on test documentation

• Test case design and documentation over emphasized
  – Both in textbooks and research
• Test cases make test designs tangible, reviewable, and easy to plan and track – i.e. manage and control
• In many contexts test cases and other test design documentation are needed

• The level and type of test documentation should vary based on context
Experience is invaluable in software testing

- **Domain experience**
  - Knowledge and skills gained in the application domain area
  - How the system is used in practice, and by whom
  - What are the goals of the users
  - How is the system related to the customer’s (business) processes

- **Technical system experience**
  - How the system was built
  - What are the typical problems and defects
  - How is the system used and all the details work
  - How things work together and interact

- **Testing experience**
  - Knowledge of testing methods and techniques
  - Testing skills grown in practice
Software testing…

- is creative and exploratory work
- requires skills and knowledge
  - application domain
  - users’ processes and objectives
  - some level of technical details and history of the application under test
- requires certain kind of attitude
Tester’s Attitude

**Destructive Attitude**

- People tend to see what they want or expect to see
  - If you want to show that software works correctly, you will miss defects
- Tester’s goal is to “break the software”
  - Reveal all relevant defects
  - Find out any problems real users would experience in practice
- Testing is all about exceptions, special cases, invalid inputs, error situations, and complicated unexpected combinations

Photo by Arvind Balaraman
Tester’s Goal

Provide information

- Explore, investigate, and measure
- Provide quality related information for other stakeholders in useful form
- Testers attitude is destructive towards the software under test, but highly constructive towards people
My viewpoint: Experience Based – Intelligent – Manual Testing

- Manual testing that builds on the tester’s experience
  - knowledge and skills
- Some aspects of testing rely on tester’s skills
  - during testing
  - e.g., input values, expected results, or interactions
- Testers are assumed to know what they are doing
  - Testing does not mean executing detailed scripts
- Focus on the actual testing work in practice
  - What happens during testing activities?
  - How are defects actually found?
  - Experience-based and exploratory aspects of software testing
Exploratory Testing is creative testing without predefined test cases

Based on knowledge and skills of the tester

1. **Tests are not defined in advance**
   - Exploring with a general mission
   - without specific step-by-step instructions on how to accomplish the mission

2. **Testing is guided by the results of previously performed tests** and the gained knowledge from them
   - Testers can apply deductive reasoning to the test outcomes

3. **The focus is on finding defects** by exploration
   - Instead of demonstrating systematic coverage

4. **Parallel learning** of the system under test, test design, and test execution

5. **Experience and skills of an individual tester** strongly affect effectiveness and results
Document driven vs. exploratory testing
Exploratory Testing is an approach

- Most of the testing techniques can be used in exploratory way
- Exploratory testing and (automated) scripted testing are the ends of a continuum
Lateral thinking

- Allowed to be distracted
- Find side paths and explore interesting areas
- Periodically check your status against your mission
Two views of agile testing

**eXtreme Testing**

- Automated unit testing
  - Developers write tests
  - Test first development
  - Daily builds with unit tests always 100% pass
- Functional (acceptance) testing
  - Customer-owned
  - Comprehensive
  - Repeatable
  - Automatic
  - Timely
  - Public

Focus on automated verification — enabling agile software development

**Exploratory Testing**

- Utilizes professional testers’ skills and experience
- Optimized to find bugs
- Minimizing time spent on documentation
- Continually adjusting plans, re-focusing on the most promising risk areas
- Following hunches
- Freedom, flexibility and fun for testers

Focus on manual validation — making testing activities agile
Contents

• Intelligent Manual Testing
  – Experience base testing
  – Exploratory testing

• Ways of Exploring
  – Session Based Test Management
  – Touring testing

• Intelligent Manual Testing Practices
  – Examples of empirically identified testing practices

• Benefits of Experience Based Testing
Some ways of exploring in practice

- Freestyle exploratory testing
  - Unmanaged ET
- Functional testing of individual features
- Exploring high level test cases
- Exploratory regression testing
  - by verifying fixes or changes

- Session-based exploratory testing
- Exploring like a tourist

- Outsourced exploratory testing
  - Advanced users, strong domain knowledge
  - Beta testing
Session Based Test Management (SBTM)


- Charter
- Time Box
- Reviewable Result
- Debriefing
Session-Based Testing
– a way to manage ET

- Enables planning and tracking exploratory testing
  - Without detailed test (case) designs
  - Dividing testing work in small chunks
  - Tracking testing work in time-boxed sessions
- Efficient – no unnecessary documentation
- Agile – it’s easy to focus testing to most important areas based on the test results and other information
  - Changes in requirements, increasing understanding, revealed problems, identified risks, …
- Explicit, scheduled sessions can help getting testing done
  - when resources are scarce
  - When testers are not full-time testers…

Juha Itkonen, 2011 SoberIT
Exploring like a tourist – a way to guide ET sessions

- Touring tests use tourist metaphor to guide testers’ actions
- Focus to intent rather than separate features
  - This intent is communicated as tours in different districts of the software

Districts and Tours

- Business district
  - Guidebook tour
  - Money tour
  - Landmark tour
  - Intellectual tour
  - FedEx tour
  - After-hours tour
  - Garbage collector’s tour
- Historical district
  - Bad-Neighborhood tour
  - Museum tour
  - Prior version tour
- Entertainment district
  - Supporting actor tour
  - Back alley tour
  - All-nighter tour
- Tourist district
  - Collector’s tour
  - Lonely businessman tour
  - Supermodel tour
  - TOGOF tour
  - Scottish pub tour
- Hotel district
  - Rained-out tour
  - Coach potato tour
- Seedy district
  - Saboteur tour
  - Antisocial tour
  - Obsessive-compulsive tour

Examples of exploratory testing tours

The Guidebook Tour
• Use user manual or other documentation as a guide
• Test rigorously by the guide
• Tests the details of important features
• Tests also the guide itself
• Variations
  – Blogger’s tour, use third party advice as guide
  – Pundit’s tour, use product reviews as guide
  – Competitor’s tour

The Garbage Collector’s Tour
• Choosing a goal and then visiting each item by shortest path
• Screen-by-screen, dialog-by-dialog, feature-by-feature, …
• Test every corner of the software, but not very deeply in the details

The All-Nighter Tour
• Never close the app, use the features continuously
  – keep software running
  – keep files open
  – connect and don’t disconnect
  – don’t save
  – move data around, add and remove
  – sleep and hibernation modes …
Contents

- Intelligent Manual Testing
  - Experience base testing
  - Exploratory testing

- Ways of Exploring
  - Session Based Test Management
  - Touring testing

- Intelligent Manual Testing Practices
  - Examples of empirically identified testing practices

- Benefits of Experience Based Testing
Empirically observed practices from industry
Testing, not test case pre-design
Practices work on different levels of abstraction
  – Many practices are similar to traditional test case design techniques
  – Many practices are similar to more general testing strategies, heuristics, or rules of thumb
IMT Practices

Overall strategies

• Structuring testing work
• Guiding a tester through features

Detailed techniques

• Low level test design
• Defect hypotheses
• Checking the test results
Overall strategies

- Exploratory
  - Exploring weak areas
  - Aspect oriented testing
  - User interface exploring
  - Top-down functional exploring
  - Simulating a real usage scenario
  - Smoke testing by intuition and experience

- Documentation based
  - Data as test cases
  - Exploring high-level test cases
  - Checking new and changed features
Detailed techniques

- Exploratory
  - Testing alternative ways
  - Exploring against old functionality
  - Simulating abnormal and extreme situations
  - Persistence testing
  - Feature interaction testing
  - Defect based exploring

- Input
  - Testing input alternatives
  - Testing boundaries and restrictions
  - Covering input combinations
  - Testing to-and-from the feature

- Comparison
  - Comparing with another application or version
  - Comparing within the software
  - Checking all the effects
  - End-to-end data check

Juha Itkonen, 2011 SoberIT
27
Basic Objectives in Testing Activities

- **Exploring**: Guiding tester through the functionality
- **Coverage**: Selecting what gets tested – and what not
- **Oracle**: Deciding if the results are correct
- **Risks**: Detecting specific types of defects
- **Prioritization**: Selecting what to test first
<exploratory strategy>

Exploring weak areas

- **Description:** Exploring areas of the software that are weak or risky based on the experience and knowledge of the tester.

- **Goal:** Reveal defects in areas that are somehow known to be risky. Focus testing on risky areas.
  - complicated
  - coded in a hurry
  - lots of changes
  - coders' opinion
  - testers' opinion
  - based on who implemented
  - a hunch...
<exploratory strategy>

Top-down functional exploring

- **Description:** Proceeding in testing by first going through typical cases and simple checks. Proceed gradually deeper in the details of the tested functionality and applying more complicated tests.

- **Goal:** To get first high level understanding of the function and then deeper confidence on its quality set-by-step.
  - Is this function implemented?
  - Does it do the right thing?
  - Is there missing functionality?
  - Does it handle the exceptions and special cases?
  - Does it work together with the rest of the system?
  - How about error handling and recovery
  - …
Using data as test cases

- **Description:** Pre-defined test data set includes all relevant cases and combinations of different data and situations. Covering all cases in a pre-defined test data set provides the required coverage.
  - Testing is exploratory, but the pre-defined data set is used to achieve systematic coverage.
  - Suitable for situations where data is complex, but operations simple. Or when creating the data requires much effort.

- **Goal:** To manage exploratory testing based on pre-defined test data. Achieve and measure coverage in exploratory testing.

- **Example:** Different types of customers in a CRM system.
  - User privileges
  - Situation, services, relationships
  - History, data
Comparing within the software

• **Description:** Comparing similar features in different places of the same system and testing their consistency.

• **Goal:** Investigating and revealing problems in the consistency of functionality inside a software; help decide if a feature works correctly or not.
Testing to-and-from the feature

• **Description:**
  - Test all things that affect to the feature
  - Test all things that get effects from the feature

• **Goal:** Systematically cover the feature’s interactions. Reveal defects that are caused by a not-the-most-obvious relationship between the tested feature and other features or environment.
Ways of utilizing IMT Practices

- Training testers
- Guiding test execution
- Test documentation and tracking
- Test patterns for different situations
Contents

- Intelligent Manual Testing
  - Experience base testing
  - Exploratory testing

- Ways of Exploring
  - Session Based Test Management
  - Touring testing

- Intelligent Manual Testing Practices
  - Examples of empirically identified testing practices

- Benefits of Experience Based Testing
Strengths of experience based testing

Testers’ skills

- Utilizing the skills and experience of the tester
  - Testers know how the software is used and for what purpose
  - Testers know what functionality and features are critical
  - Testers know what problems are relevant
  - Testers know how the software was built
    - Risks, tacit knowledge
- Enables creative exploring
- Enables fast learning and improving testing
  - Investigating, searching, finding, combining, reasoning, deducting, ...
- Testing intangible properties
  - “Look and feel” and other user perceptions
Strengths of experience based testing

Process

• Agility and flexibility
  – Easy and fast to focus on critical areas
  – Fast reaction to changes
  – Ability to work with missing or weak documentation

• Effectiveness
  – Reveals large number of relevant defects

• Efficiency
  – Low documentation overhead
  – Fast feedback
Challenges of experience based testing

- Planning and tracking
  - How much testing is needed, how long does it take?
  - What is the status of testing?
  - How to share testing work between testers?

- Managing test coverage
  - What has been tested?
  - When are we done?

- Logging and reporting
  - Visibility outside testing team
    - or outside individual testing sessions

- Quality of testing
  - How to assure the quality of tester’s work
    - Detailed test cases can be reviewed, at least
Reasons for documenting test cases

- Optimizing
  - Selecting optimal test set
  - Avoiding redundancy
- Organization
  - Organized so that tests can be reviewed and used effectively
  - Selecting and prioritizing
- Repeatability
  - Know what test cases were run and how; so that you can repeat the same tests
- Tracking
  - What requirements, features, or components are tested
  - What is the coverage of testing
  - How testing proceeds? Are we going to make the deadline?
- Proof of testing
  - Evaluating the level of confidence
  - How do we know what has been tested?
Detail level of test cases

- Experienced testers need less detailed test cases
  - More experienced as testers
  - More familiar with the software and application domain
- Input conditions
  - Depends on the testing technique and goals of testing
  - E.g. if goal is to cover all pairs of certain input conditions, the test cases have to be more detailed than in scenario testing
- Expected results
  - More detail is required if the result is not obvious, requires complicated comparison, etc.
  - Inexperienced tester needs more guidance on what to pay attention to
Why should we document the expected outcome?

• The expected values explicitly define what is the correct result
  – Important if the correct result is not obvious

On the other hand...
If provided with detailed expected results, the tester just looks for the exact details pointed out and ignores everything else – many unexpected defects are missed.

Looks OK to me!
Experimental Comparison of ET and Test Case Based Testing (TCBT)


- Effectiveness in terms of revealed defects
- Test execution time was fixed

- **No difference in effectiveness**
  - ET revealed more defects, but no statistical difference

- **TCBT required much more effort**
  - Test case design before the test execution

- **TCBT produced twice as many false reports than ET**
Who tested my software?

- **Testing is not an action that is solely performed by specialists.** In all our cases, people in roles that vary from sales to software development found a substantial number of defects.
- **Validation from the viewpoint of end-users was found more valuable than verification aiming at zero defect software.**

### Table 7 Distribution of defect reports between reporter groups

<table>
<thead>
<tr>
<th>Reporter Group</th>
<th>Case A</th>
<th>Case B</th>
<th>Case C</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sales &amp; Consulting</td>
<td>145</td>
<td>117</td>
<td>136</td>
<td>398</td>
</tr>
<tr>
<td>Support</td>
<td>111</td>
<td>79</td>
<td>239</td>
<td>429</td>
</tr>
<tr>
<td>Manager</td>
<td>108</td>
<td>247</td>
<td>476</td>
<td>831</td>
</tr>
<tr>
<td>Internal Misc</td>
<td>348</td>
<td>89</td>
<td>620</td>
<td>1057</td>
</tr>
<tr>
<td>Specialized Tester</td>
<td>367</td>
<td>-</td>
<td>117</td>
<td>484</td>
</tr>
<tr>
<td>Developer</td>
<td>134</td>
<td>419</td>
<td>282</td>
<td>835</td>
</tr>
<tr>
<td>Customer (ext)</td>
<td>458</td>
<td>431</td>
<td>N/A</td>
<td>889</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>1671</td>
<td>1382</td>
<td>1870</td>
<td>4923</td>
</tr>
</tbody>
</table>
Who tested my software?

- *Developers’ defects had the highest fix rate and specialized testers’ defects had the lowest fix rate.*
- *People with a personal stake in the product (e.g., sales and consulting personnel) tend to place more importance on their defects, but it does not seem to improve their fix ratios.*
The role of knowledge in failure detection

- Detailed analysis of 91 defect detection incidents from video recorded exploratory testing sessions
- Analysed what type of knowledge is required for detecting failures?
- Analysed failure detection difficulty
The role of knowledge in failure detection – findings

• Knowledge utilization
  • Testers are able to utilize their personal knowledge of the application domain, the users’ needs, and the tested system for defect detection.

• Side effect bugs
  • In exploratory testing, testers frequently recognize relevant failures in a wider set of features than the actual target features of the testing activity.

• Obvious failures
  • A large number of the failures in software applications and systems can be detected without detailed test design or descriptions.

• Application domain related failures are simple to reveal
  • The majority of failures detected by domain knowledge are straightforward to recognize.
The role of knowledge in failure detection – conclusions

• The ET approach could be effective even when less experienced testers are used.

• All testing does not need to be scripted or rigorously (pre)designed.
  – A lot of benefits can be achieved by efficient exploring

• The ET approach is an effective way of involving the knowledge of domain experts in testing activities
  – who do not have testing expertise
Questions and more discussion

Contact information
Juha Itkonen
juha.itkonen@aalto.fi
+358 50 577 1688
http://www.soberit.hut.fi/jitkonen
References (primary)


References (secondary)


