Lectures

• Lecture 1 (11.02) – Introduction to Software Testing
• Lecture 2 (18.02) – Basic Black-Box Testing Techniques: Boundary Value Analysis & Equivalence Class Partitioning
• Lecture 3 (25.02) – BBT advanced: Combinatorial Testing
• Lecture 4 (04.03) – Basic White-Box Testing Techniques: Control-Flow Coverage
• Lecture 5 (11.03) – BBT adv.: State-Transition, Metamorphic, Random Testing
• Lecture 6 (18.03) – Test Levels, Test Tools, Test Automation
• Lecture 7 (25.03) – BBT adv.: Exploratory Testing, Behaviour Testing
• Lecture 8 (01.04) – BBT adv.: GUI / Visual Testing, Usability Testing, A/B Testing
• Lecture 9 (08.04) – Data-Flow Testing / Test-Suite Effectiveness: Mutation Testing
• Lecture 10 (15.04) – WBT adv.: Symbolic Execution, Static Code Analysis, Review
• Lecture 11 (22.04) – Defect Estimation / Test Documentation, Organisation and Process Improvement (Test Maturity Model)
• Lectures 12+13 (29.04 + 06.05) – Industry Guest Lectures + Advanced Topics
• Lecture 14 (13.05) – Exam Preparation
Structure of Lecture 8

- GUI Testing
  - GUITAR
  - TESTAR
- Visual Testing
  - Sikuli IDE
  - Lab 7
- Security Testing
- Usability Testing
- A/B Testing
Structure of Lecture 8

- GUI Testing
  - GUITAR
  - TESTAR
- Visual Testing
  - Sikuli IDE
  - Lab 7
- Security Testing
- Usability Testing
- A/B Testing
GUITAR

Source: https://sourceforge.net/projects/guitar/
GUITAR – GUI Ripping

• During ripping, the GUI application is executed automatically; the application’s windows are opened in a depth first manner.

• The Ripper extracts all the widgets and their properties from the GUI.
  • Properties of widgets include basic attributes such as position, color, size, and enabled status.
  • Properties also include information about widgets' events, such as: whether a widget opens a modal or modeless window or a menu, whether a widget closes a window, and whether the widget is a button or an editable text-field.

• The Ripper extracts properties for widgets as well as their containing GUI windows and stores the information in the GUI Tree.
  • For each GUI window, the Ripper first extracts structural information of that window and then executes widgets that invoke other GUI windows.
  • The depth-first traversal terminates when all GUI windows are covered.
GUITAR – GUI Ripping

GUI of an app with two windows

Radio Button Demo
- Attributes:
  - Title=Radio Button Demo
  - Root=TRUE
  - Width=100
  - Height=50
  - Buttons=Create Reset Exit
  - Exit . Invokes=Exit Confirmation

Exit Confirmation
- Attributes:
  - Title=Exit Confirmation
  - Width=100
  - Height=30
  - Buttons=Yes No
  - Yes . Terminal=TRUE
  - No . Terminal=TRUE
GUITAR – Model Conversion

- EFG = Event Flow Graph
GUITAR – Test Case Generation

Applies traversal algorithms for the EFG:

Defined length $L$ of event sequences

Example ($L=5$):

$e_1 \rightarrow e_2 \rightarrow e_3 \rightarrow e_4 \rightarrow e_5$
# GUITAR – Test Case Execution & Evaluation

<table>
<thead>
<tr>
<th>Fault ID</th>
<th>Summary</th>
<th>Test case</th>
</tr>
</thead>
<tbody>
<tr>
<td>AU&lt;sub&gt;1&lt;/sub&gt;</td>
<td>FileNotFoundException with invalid input file name for Export Graphic</td>
<td>Expand ‘File’ menu → Click ‘Export Graphic’ submenu → Enter an invalid file name → Click ‘Save’</td>
</tr>
<tr>
<td>AU&lt;sub&gt;2&lt;/sub&gt;</td>
<td>FileNotFoundException with invalid input file name for Export All Graphic</td>
<td>Expand ‘File’ menu → Click ‘Export All Graphics’ submenu → Enter an invalid file name → Click ‘Save’</td>
</tr>
<tr>
<td>AU&lt;sub&gt;3&lt;/sub&gt;</td>
<td>An inappropriate exception trace printed out when deleting object with a blank document</td>
<td>Expand ‘Edit’ menu → Click ‘Delete from Model’ submenu</td>
</tr>
<tr>
<td>JR&lt;sub&gt;1&lt;/sub&gt;</td>
<td>FileNotFoundException with a non-existing Journal abbreviation file</td>
<td>Expand ‘Option’ menu → Click ‘Manage journal abbreviation’ submenu → Enter an invalid New file name → Click ‘OK’</td>
</tr>
<tr>
<td>JR&lt;sub&gt;2&lt;/sub&gt;</td>
<td>MalformedURLException with an invalid Journal abbreviation download URL</td>
<td>Expand the ‘Option’ menu → Open ‘Journal abbreviation’ windows → Click ‘Download’ button → Enter an invalid URL → Click ‘OK’</td>
</tr>
<tr>
<td>JR&lt;sub&gt;3&lt;/sub&gt;</td>
<td>NullPointerException with invalid import folder name</td>
<td>Expand ‘Option’ menu → Click ‘Manage custom imports’ submenu → Click ‘Add from folder’ → Enter a non-existing folder path → Click ‘Cancel’</td>
</tr>
<tr>
<td>JR&lt;sub&gt;4&lt;/sub&gt;</td>
<td>ZipException with invalid zip file name</td>
<td>Expand ‘Option’ menu → Click ‘Manage custom imports’ submenu → Click ‘Add from jar’ → Enter a non-existing zip file name → Click ‘Select a Zip-archiver’</td>
</tr>
</tbody>
</table>

Expand the ‘Option’ menu –> Open ‘Journal abbreviation’ window –> Click ‘Download’ button –> enter an invalid URL –> Click ‘OK’
MalformedURLException with an invalid Journal abbreviation download URL: “CRASH ME”
The text “CRASH ME” was provided by the testers – but the app should not have been crashed; instead it should have handled the incorrect input properly.
Structure of Lecture 8

- GUI Testing
  - GUITAR
  - TESTAR
- Visual Testing
  - SikuliX
  - Lab 7
- Security Testing
- Usability Testing
- A/B Testing
‘Scriptless’ testing tool Testar

- Scripts are generated automatically (on-the-fly)
- No maintenance of scripts needed
- Main purpose: Robustness testing
  - Robustness is the ability of a computer system to cope with errors during execution and cope with erroneous input

[Link to testar.org]
Testar tool (1)

- GUI contains graphical objects called widgets
- Widgets form a hierarchy called widget tree
- Widgets have properties (title, etc.)
- Widget tree and properties of each widget form a GUI state
- GUI state changes, if user executes an action (click, write, etc.)
Testar tool (2)

- Example widget of Calculator program
Testar tool (3)

- Widget tree of Calculator program
Testar tool (4) – What it can do ...

Invalid input to SUT:
1) error message (specified) \(\rightarrow\) NOT A FAILURE \(\rightarrow\) can be ignored (if necessary apply Filter to avoid it’s activation)
2) error message (not specified) \(\rightarrow\) FAILURE \(\rightarrow\) define Oracle to detect it
3) no error message \(\rightarrow\) FAILURE (BUT: Testar cannot detect it)

Valid input to SUT:
1) no error message \(\rightarrow\) NOT A FAILURE, i.e., common behavior of the SUT and we don't care about it \(\rightarrow\) apply filters to avoid unwanted behavior (e.g., ‘close’ button)
2) error message (not specified) \(\rightarrow\) FAILURE --> define Oracle to detect it
3) error message (specified for invalid input) \(\rightarrow\) FAILURE (BUT: Testar will treat it like NOT A FAILURE \(\rightarrow\) see “Invalid input” section on the left)
Testar tool (4) – What it can do ...

Invalid input to SUT:

1) error message (specified) \(\rightarrow\) NOT A FAILURE \(\rightarrow\) can be ignored (if necessary apply Filter to avoid it’s activation)
2) error message (not specified) \(\rightarrow\) FAILURE \(\rightarrow\) define Oracle to detect it
3) no error message \(\rightarrow\) FAILURE (BUT: Testar cannot detect it)

Valid input to SUT:

1) no error message \(\rightarrow\) NOT A FAILURE, i.e., common behavior of the SUT and we don't care about it \(\rightarrow\) apply filters to avoid unwanted behavior (e.g., ‘close’ button)
2) error message (not specified) \(\rightarrow\) FAILURE --\(\rightarrow\) define Oracle to detect it
3) error message (specified for invalid input) \(\rightarrow\) FAILURE (BUT: Testar will treat it like NOT A FAILURE \(\rightarrow\) see “Invalid input” section on the left)

Note: Theoretically, one could think of defining messages for incorrect business logic, but there is no way how such messages could be triggered when adding valid input – only if the production code had built-in asserts, then this was a possibility.
Testar tool (5) – What it can do ...

Testar always detects:
• … when SUT crashes or hangs (no Oracle needs to be defined)

Testar cannot detect:
• … wrong business logic implemented (e.g., 2 + 2 \(\Rightarrow\) result == 5)
• … when specified error messages are shown although the input to a function was valid
• … when invalid input is not handled properly (i.e., no specified error message is presented) AND there is no (unspecified) exception thrown or a program crash/hanging
Testar tool (6)

1. Start SUT
2. Scan GUI state
3. Select action
4. Execute action
5. Check oracles*
   - Failure detected → Sequence leading to failure
   - No failure detected → Check stop criteria
4a. Check stop criteria
   - Yes → Stop SUT
   - No → Derive (+filter) actions
5a. Derive (+filter) actions

* or crash
Testar tool (6)

Some parameters have to be set before running Testar (-> stop criteria)

- Start SUT
- Scan GUI state
- Check stop criteria
- Derive (+filter) actions
- Execute action
- Select action
- Check oracles*
- Sequence leading to failure
- Stop SUT

Failure detected
No failure detected

Yes
- No

Use Regular Expressions to define
- Oracles
- Filters

* or crash

Some parameters have to be set before running Testar (-> stop criteria)
Testar tool (7) – Stop criteria

• Number of Sequences (=Iterations)
• Number of Actions per Sequence
  – Identify when there have enough actions been performed for one test

\[
\begin{align*}
1 & \div 2 \div 1 + 4 4 4 \times 3 = 10 \text{ actions} \\
+ & - 3 \div 4 / 0 4 = 2 \\
\ldots & \text{Sequence 1: 10 actions} \\
\ldots & \text{Sequence 2: 10 actions}
\end{align*}
\]
Testar tool (8) – Regular expressions

• Example:

\b[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}\b

???
Testar tool (8) – Regular expressions

• Example:

\b[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}\b

dietmar@acm.org
Testar tool (9) – Oracles

- Testar can detect two types of failures:
  - When SUT crashes or hangs (no oracles needed)
  - When SUT’s widget title matches a specified oracle (regex)

Example: Test engineer specifies „fail“ as an oracle
- If the word „fail“ appears in some widget title, then Testar reports it as a failure (test = fail)
- If the word “fail” doesn’t appear in a sequence, then by default the test passes (test = pass)
Testar tool (10) - Filters

- Default: Testar executes actions on all the widgets it finds
  - Including buttons on the title bar (e.g., close, minimize, maximize, system)

- User can define that specific widgets shall not be executed
  - For example, forbid clicking on ‘close’ or ‘minimize’ buttons
  - To do this, an action filter can be configured
    - Filtering is done by matching the widget title to defined filters (using regex)
    - For example, user can specify „.*close.*“ as a filter and Testar will not execute actions on widgets, where title contains „close“
Testar tool (11) – Filter example

• Example:

  .*Backspace.*|.*CE.*|.*View.*

  This expression could be used to filter out (=ignore) all clicks to control elements whose title contain one of the given strings.
Testar output → folder structure

- **Testar**
  - **Bin**
    - **Testar.bat**
      - File to run Testar
    - **Output**
      - Sequences that matched oracles
      - Sequences where the SUT closed unexpectedly
    - **Graphs**
      - Contains raw graph files and a converter

- **SUT**
  - **Calculator**
    - Exercise in lab 2019
  - **ATM**
    - Homework task 2019

Note: This slide only shows the most important folders
Interpreting output (1)

• Tester must determine if the output reports a …
  – true positive -> an actual failure or …
  – false positive -> reported as failure, although it is not
  – That’s why the folders are called „unexpected…“ and „suspicious…“

• Tester must look through all the sequences
• Beware that there are many failure duplicates, and there is no (easy) way to automatically analyse the output files
Interpreting output (2)

- Testar generates files that can be converted into graphs
  - "offline_graph_conversion.bat" is automatically provided in each graph folder
- All the graphs are converted automatically using the .bat file
  - you use the screenshotted graph to understand what happened in the sequence
- Looking through the screenshotted graphs is simple but ...
  - it can happen that there are some layout issues and not all GUI states in a screenshot are connected
• Like in this example, where you can see only one state before failure

Which was previous state?
• To find the previous state, use action number in brackets
• Search for lower number to find previous state
• Ctrl + f
• It will show you the previous action
• This way you can move through the graph if some parts are not connected
Hints (1)

• Use a reasonable number of sequences/iterations (e.g., something between 20 and 100)
  • If the number is too small, you might not trigger a (potential) failure at all
  • If the number is too high, the execution might take too long and the (manual) analysis of the output files might take too much time

• Use a reasonable number of actions in each sequence/iteration
  • It will keep graphs smaller and therefore easier to read
  • A maximum of 20 actions should be fine but what to choose depends also on how you use the action filters
Hints (2)

• If you keep finding the same failure many times, you should consider filtering out the path leading to that failure
  • It will save you time while inspecting graphs
  • Also, make sure, that you don’t apply filters too early, otherwise you might miss some of the failures

• Information messages are not to be considered failures if they match those listed in the appendixes A (in-lab task) and C (homework task)
  • Don’t write oracles matching these messages

• Check out Testar manual for more information about the tool
Structure of Lecture 8

- GUI Testing
  - GUITAR
  - TESTAR
- Visual Testing
  - Sikuli IDE
  - Lab 7
- Security Testing
- Usability Testing
- A/B Testing
Sikuli IDE

Capture images of the GUI

Then write tests using those images

... e.g., to test email forwarding with Gmail ...
How Sikuli X mimics the user’s behavior:

- Find all the check boxes for emails
- Shift coordinate
- Click on the subject line
- Wait for everything to load
- Select the options ‘Reply’ and ‘Forward’ in the list of options

...
Sikuli IDE

Many hints will be given in the Lab 7 session

... and can be found in the documentation.

1 Introduction

The purpose of this lab is to learn how Visual GUI Testing helps testers automate the testing of GUI interfaces. Visual GUI Testing (VGT) tools are usually used for acceptance testing but can also be used for regression testing. The tool that we will be working with in this Lab is “Sikuli”.

Most automated testing techniques approach testing from low-level abstraction. GUI intensive systems however require high-level tests. To battle the problem of automating high-level tests, techniques such as “Record and Play” and “VGT” were developed.

Record and Play

A tool that uses this technique, records the coordinates of GUI-components, with which the tester manually interacts. Then, the recording of the test can be played again to simulate the user’s interaction with the system. This technique provides a much better solution than manual testing, because the test needs to be done manually only once. However, this technique has a major flaw. “Record and Play” tools are usually sensitive to GUI layout, which means that it is nearly impossible to run the tests consistently on different screens, resolutions, machines etc.

VGT

Visual GUI testing is a technique that uses image recognition to interact with what is shown on the screen. In the process of VGT, the tester creates a script with valid instructions. GUI interaction, that was previously done by specifying the exact coordinates of the pixels of the element, are now done by taking a screenshot of the GUI element in question. VGT tools interact with the computer’s screen to search for matches of these screenshots. This makes automating high-level tests much more reliable and useful.
Lab 7 – Visual GUI Testing (Sikuli IDE)

Lab 7 (week 32: Apr 06 & 07) – Visual GUI Testing (9 points)

Lab 8 Instructions & Tools

Submission Deadlines:
- Tuesday Labs: Monday, 12 Apr, 23:59
- Wednesday Labs: Tuesday, 13 Apr, 23:59

- Penalties apply for late delivery: 50% penalty, if submitted up to 24 hours late; 100 penalty, if submitted more than 24 hours late
Structure of Lecture 8

• GUI Testing
  • GUITAR
  • TESTAR
• Visual Testing
  • Sikuli IDE
  • Lab 7
• Security Testing
• Usability Testing
• A/B Testing
Software Product Quality Model – ISO 25010 Standard
Security Testing – What?

- Security Testing = Type of testing that …
  - intends to *uncover vulnerabilities* of the system and
  - determine that its data and resources are *protected from possible intruders*

- Focus Areas:
  - **Network security:** This involves looking for vulnerabilities in the network infrastructure (resources and policies).
  - **System software security:** This involves assessing weaknesses in the various software (operating system, database system, and other software) the application depends on.
  - **Client-side application security:** This deals with ensuring that the client (browser or any such tool) cannot be manipulated.
  - **Server-side application security:** This involves making sure that the server code and its technologies are robust enough to fend off any intrusion.
Security Testing – Why?

• Primary purpose of security testing is to identify the vulnerabilities and subsequently repairing them

• Security Testing helps in
  • improving the current system
  • ensuring that the system will work for longer time
  • finding out loopholes that can cause loss of important information
Security Testing – Basic Concepts

• Confidentiality
• Integrity
• Authentication
• Authorization
• Accountability
• Non-repudiation
• [Availability]
Security Testing – Basic Concepts

• Confidentiality
• Integrity
• Authentication
• Authorization
• Accountability
• Non-repudiation
• [Availability]

is about …

... ensuring that information is accessible only for those with authorized access and

... preventing information theft
Security Testing – Basic Concepts

- Confidentiality
- **Integrity**
- Authentication
- Authorization
- Accountability
- Non-repudiation
- [Availability]

is about …

… allowing the receiver of data to determine that the information which the data is providing is correct
Security Testing – Basic Concepts

- Confidentiality
- Integrity
- **Authentication**
- Authorization
- Accountability
- Non-repudiation
- [Availability]

is about …
… establishing the identity of the user
Security Testing – Basic Concepts

• Confidentiality
• Integrity
• Authentication
• Authorization
• Accountability
• Non-repudiation
• [Availability]

is about …

… determining that a data requester is allowed to receive a service or perform an operation
Security Testing –
Basic Concepts

- Confidentiality
- Integrity
- Authentication
- Authorization
- **Accountability**
- Non-repudiation
- [Availability]

is about …

… taking responsibility (and transparency about the responsibilities taken) for establishing security
Security Testing –
Basic Concepts

• Confidentiality
• Integrity
• Authentication
• Authorization
• Accountability
• **Non-repudiation**
• [Availability]

is about …

… preventing the later denial that an action happened or a communication took place, etc.
Security Testing – Basic Concepts

• Confidentiality
• Integrity
• Authentication
• Authorization
• Accountability
• Non-repudiation
• [Availability]

is about …
… assuring that information and communications services will be ready for use when expected/needed
Main security testing practices include:

- Vulnerability Scanning
- Security Scanning
- Penetration Testing
- Ethical Hacking
- Risk Assessment
- Security Auditing
- Posture Assessment
- Password cracking
Security Testing – What?

Main security testing practices include:

- **Vulnerability Scanning** — Involves scanning of the application for all known vulnerabilities. Typically done with the help of vulnerability scanning software, e.g., Nessus, Nikto, Gendarme, Flawfinder, etc.
- Security Scanning
- Penetration Testing
- Ethical Hacking
- Risk Assessment
- Security Auditing
- Posture Assessment
- Password cracking
Security Testing – What?

Main security testing practices include:

- Vulnerability Scanning
- **Security Scanning**
- Penetration Testing
- Ethical Hacking
- Risk Assessment
- Security Auditing
- Posture Assessment
- Password cracking

Involves scanning the usage of a system in operation to monitor whether attacks on the system are being tried or perhaps prepared.
Security Testing – What?

Main security testing practices include:

- Vulnerability Scanning
- Security Scanning
- **Penetration Testing**
- Ethical Hacking
- Risk Assessment
- Security Auditing
- Posture Assessment
- Password cracking

Testers try to (unauthorized) enter into the application with the help of another software system and via (combinations of) loopholes that the application has kept open unknowingly.
Security Testing – What?

Main security testing practices include:

- Vulnerability Scanning
- Security Scanning
- Penetration Testing
- **Ethical Hacking**
- Risk Assessment
- Security Auditing
- Posture Assessment
- Password cracking

Involves the performance of penetration tests over the wide network on the system under test

Conducted by ethical hackers who try to find possible problems in the system
Security Testing – What?

Main security testing practices include:

- Vulnerability Scanning
- Security Scanning
- Penetration Testing
- Ethical Hacking
- **Risk Assessment**
- Security Auditing
- Posture Assessment
- Password cracking

Analyzing and deriving the risk as a function of potential damage (or loss) and the possibility of damage occurrence. Typically carried out in the form of scenario analysis, interviews, and expert discussions.
Security Testing – What?

Main security testing practices include:

- Vulnerability Scanning
- Security Scanning
- Penetration Testing
- Ethical Hacking
- Risk Assessment
- **Security Auditing**
- Posture Assessment
- Password cracking

Involves hands-on internal inspection of Operating Systems and Applications, often via line-by-line inspection of the code.

A security audit is a systematic evaluation of the security of a software system used by a company.
Security Testing – What?

Main security testing practices include:

- Vulnerability Scanning
- Security Scanning
- Penetration Testing
- Ethical Hacking
- Risk Assessment
- Security Auditing
- **Posture Assessment**
- Password Cracking

combines Security Scanning, Ethical Hacking and Risk Assessments to show an overall Security Posture (=attitude towards security) of the organization using a software system.
Security Testing – What?

Main security testing practices include:

- Vulnerability Scanning
- Security Scanning
- Penetration Testing
- Ethical Hacking
- Risk Assessment
- Security Auditing
- Posture Assessment
- **Password Cracking**

Password cracking programs can be used to identify weak passwords. Password cracking verifies that users are employing sufficiently strong passwords.
Security Testing – What?

Example of a basic security test:

- Log into the web application.
- Log out of the web application.
- Click the BACK button of the browser (Check if you are asked to log in again or if you are provided the logged-in application.)

- Most types of security testing involve complex steps and out-of-the-box thinking but, sometimes, it is simple tests like the one above that help expose the most severe security risks.

The Open Web Application Security Project (OWASP) is a great resource for software security professionals.


OWASP Top 10 security threats for 2013 were:
- Injection
- Broken Authentication and Session Management
- Cross-Site Scripting (XSS)
- Insecure Direct Object References
- Security Misconfiguration
- Sensitive Data Exposure
- Missing Function Level Access Control
- Cross-Site Request Forgery (CSRF)
- Using Known Vulnerable Components
- Unvalidated Redirects and Forwards
The Open Web Application Security Project (OWASP) is a great resource for software security professionals. Be sure to check out the Testing Guide: https://www.owasp.org/index.php/Category:OWASP_Testing_Project

OWASP Top 10 security threats for 2013 were:
- Injection
- Broken Authentication and Session Management
- Cross-Site Scripting (XSS)
- Insecure Direct Object References
- Security Misconfiguration
- Sensitive Data Exposure
- Missing Function Level Access Control
- Cross-Site Request Forgery (CSRF)
- Using Known Vulnerable Components
- Unvalidated Redirects and Forwards
How to avoid SQL injection vulnerability?

Instead of:

```java
String query = "SELECT * FROM Users WHERE Username= "
    + request.getParameter("username")
    + "AND Password= "
    + request.getParameter("password");

try {
    Statement statement = connection.createStatement();
    ResultSet results = statement.executeQuery(query);
}
```

Which might result in a SQL query string like this:

```
SELECT * FROM Users WHERE Username='1' OR '1' = '1' AND
Password='1' OR '1' = '1'
```
How to avoid SQL injection vulnerability?

Use java ‘prepared statement’:

```java
String username = request.getParameter("username");
String password = request.getParameter("password");
// perform input validation to detect attacks

String query = "SELECT * FROM Users WHERE Username= ? AND Password= ?";

PreparedStatement pstmt = connection.prepareStatement(query);
pstmt.setString( 1, username);
pstmt.setString( 2, password);

ResultSet results = pstmt.executeQuery();
```

Example with Hibernate Query Language (HQL) can be found here:
https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html
Structure of Lecture 8

- GUI Testing
  - GUITAR
  - TESTAR
- Visual Testing
  - Sikuli IDE
  - Lab 7
- Security Testing
- Usability Testing
- A/B Testing
Software Product Quality Model – ISO 25010 Standard

Usability
Testing Usability Requirements

<table>
<thead>
<tr>
<th>Problem counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1: At most 1 of 5 novices shall encounter critical problems during tasks Q and R. At most 5 medium problems on list.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Task time</th>
</tr>
</thead>
<tbody>
<tr>
<td>R2: Novice users shall perform tasks Q and R in 15 minutes. Experienced users tasks Q, R, S in 2 minutes.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Keystroke counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>R3: Recording breakfast shall be possible for guest. No mouse.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Opinion poll</th>
</tr>
</thead>
<tbody>
<tr>
<td>R4: 80% of users shall find system easy to use and recommend system to others.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Score for understanding</th>
</tr>
</thead>
<tbody>
<tr>
<td>R5: Show 5 users 10 common error messages. Ask for the cause. 80% of the users shall agree.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Design-level reqs</th>
</tr>
</thead>
<tbody>
<tr>
<td>R6: System shall use screen pictures in app. xx, buttons work as app. yy.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Product-level reqs</th>
</tr>
</thead>
<tbody>
<tr>
<td>R7: For all code fields, user shall be able to select value from drop-down list.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Guideline adherence</th>
</tr>
</thead>
<tbody>
<tr>
<td>R8: System shall follow style guide zz. Menus shall have at most three levels.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Development process reqs</th>
</tr>
</thead>
<tbody>
<tr>
<td>R9: Three prototype versions shall be made and usability tested during design.</td>
</tr>
</tbody>
</table>
Testing Usability Requirements

How to test:
- Define several (typical) usage scenarios involving tasks Q and R
- Select test users and classify as 'novice' and 'experienced'
- Let 5 (or better 10, 15) novices perform the scenarios
- Observe what problems they encounter
- Classify and count observed problems
Usability Test Types + Environment

Rubin’s Types of Usability Tests (Rubin, 1994, p. 31-46)

**Exploratory test** – early product development

**Assessment test** – most typical, either early or midway in the product development

**Validation test** – confirmation of product’s usability

**Comparison test** – compare two or more designs; can be used with other three types of tests
Usability Testing – What? How?

• Test Focus
  • Understandability
    • Easy to understand?
  • Ease of learning
    • Easy to learn?
  • Operability
    • Matches purpose & environment of operation?
    • Ergonomics: color, font, sound,...
  • Communicativeness
    • In accordance with psychological characteristics of user?

• Test Environments
  • Free form tasks
  • Procedure scripts
  • Paper screens
  • Mock-ups
  • Field trial
Evaluating UI Designs

Inspection Methods
- Heuristic Evaluation
  - Cognitive Walkthrough
  - Guidelines Review

Usability Testing
- Laboratory Experiment
  - Field Study
Evaluating UI Designs

- Inspection Methods
  - Heuristic Evaluation
    - Cognitive Walkthrough
    - Guidelines Review
- Usability Testing
  - Laboratory Experiment
  - Field Study

Cheap

Expensive
Evaluating UI Designs

- **Inspection Methods**
  - **Heuristic Evaluation**
    - evaluates design on how well it supports user in learning task
    - usually performed by expert in cognitive psychology
    - expert `walks though' design to identify potential problems using psychological principles
    - Scenarios may be used to guide analysis

- **Usability Testing**
- **Cognitive Walkthrough**
Evaluating UI Designs

Inspection Methods
- Cognitive Walkthrough
  - usability criteria (heuristics) are identified
  - design examined by experts to see if these are violated
- Heuristic Evaluation
- Guidelines

Usability Testing
- Laboratory Experiment
- Field Study
Heuristic Evaluation by Inspection

List of 10 Heuristics according to (Nielsen, 2005):

<table>
<thead>
<tr>
<th></th>
<th>Visibility of system status</th>
<th></th>
<th>Match between the system and the real world</th>
<th>3</th>
<th>User control and freedom</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Error prevention</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Help and documentation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

List violations of heuristics:

Rank by severity: 0...4
0: positive (or neutral) aspect of system
... 4: major, catastrophic aspect of system


Heuristic Evaluation

List of 10 Heuristics according to (Nielsen, 2005):

<table>
<thead>
<tr>
<th>#</th>
<th>Heuristic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Visibility of system status</td>
</tr>
<tr>
<td>2</td>
<td>Match between the system and the real world</td>
</tr>
<tr>
<td>3</td>
<td>User control and freedom</td>
</tr>
<tr>
<td>4</td>
<td>Consistency and standards</td>
</tr>
<tr>
<td>5</td>
<td>Error prevention</td>
</tr>
<tr>
<td>6</td>
<td>Recognition rather than recall</td>
</tr>
<tr>
<td>7</td>
<td>Flexibility and efficiency of use</td>
</tr>
<tr>
<td>8</td>
<td>Aesthetic and minimalist design</td>
</tr>
<tr>
<td>9</td>
<td>Help users recognize, diagnose, and recover from errors</td>
</tr>
<tr>
<td>10</td>
<td>Help and documentation</td>
</tr>
</tbody>
</table>

Visibility of system status:
At all times, the system should inform the end user what is currently ongoing, e.g., via timely and adequate feedback.

List violations of heuristics:

Rank by severity: 0...4
0: positive (or neutral) aspect of system ...
4: major, catastrophic aspect of system

---


Heuristic Evaluation

List of 10 Heuristics according to (Nielsen, 2005):

<table>
<thead>
<tr>
<th></th>
<th>Visibility of system status</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Match between the system and the real world</td>
</tr>
<tr>
<td>3</td>
<td>User control and freedom</td>
</tr>
<tr>
<td>4</td>
<td>Consistency and standards</td>
</tr>
<tr>
<td>5</td>
<td>Error prevention</td>
</tr>
<tr>
<td>6</td>
<td>Recognition rather than recall</td>
</tr>
<tr>
<td>7</td>
<td>Flexibility and efficiency of use</td>
</tr>
<tr>
<td>8</td>
<td>Aesthetic and minimalist design</td>
</tr>
<tr>
<td>9</td>
<td>Help users recognize, diagnose, and recover from errors</td>
</tr>
<tr>
<td>10</td>
<td>Help and documentation</td>
</tr>
</tbody>
</table>

Match between the system and the real world:

The system should speak the language of the end user (i.e., it should use adequate terminology from the domain of the end user).

The system should follow the conventions of the real world and present information in a logic and well-structured form.


Heuristic Evaluation

List of 10 Heuristics according to (Nielsen, 2005):

<table>
<thead>
<tr>
<th>Rank</th>
<th>Heuristic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Visibility of system status</td>
</tr>
<tr>
<td>2</td>
<td>Match between the system and the real world</td>
</tr>
<tr>
<td>3</td>
<td>User control and freedom</td>
</tr>
<tr>
<td>4</td>
<td>Consistency and standards</td>
</tr>
<tr>
<td>5</td>
<td>Error prevention</td>
</tr>
<tr>
<td>6</td>
<td>Recognition rather than recall</td>
</tr>
<tr>
<td>7</td>
<td>Flexibility and efficiency of use</td>
</tr>
<tr>
<td>8</td>
<td>Aesthetic and minimalist design</td>
</tr>
<tr>
<td>9</td>
<td>Help users recognize, diagnose, and recover from errors</td>
</tr>
<tr>
<td>10</td>
<td>Help and documentation</td>
</tr>
</tbody>
</table>

User control and freedom:

End users often select a certain function by accident; in that case, they should be able to easily find a way out of this unwanted system state, e.g., by simply choosing ‘undo’ or ‘redo’ functionality.

Rank by severity: 0...4

0: positive (or neutral) aspect of system
...
4: major, catastrophic aspect of system


Heuristic Evaluation

List of 10 Heuristics according to (Nielsen, 2005):

<table>
<thead>
<tr>
<th></th>
<th>Visibility of system status</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Match between the system and the real world</td>
</tr>
<tr>
<td>3</td>
<td>User control and freedom</td>
</tr>
<tr>
<td>4</td>
<td>Consistency and standards</td>
</tr>
<tr>
<td>5</td>
<td>Error prevention</td>
</tr>
<tr>
<td>6</td>
<td>Recognition rather than recall</td>
</tr>
<tr>
<td>7</td>
<td>Flexibility and efficiency of use</td>
</tr>
<tr>
<td>8</td>
<td>Aesthetic and minimalist design</td>
</tr>
<tr>
<td>9</td>
<td>Help users recognize, diagnose, and recover from errors</td>
</tr>
<tr>
<td>10</td>
<td>Help and documentation</td>
</tr>
</tbody>
</table>

Consistency and standards:
End users should not have to ask themselves whether different words (or symbols) and actions mean the same or do the same. Conventions that are typical for the used platform should be honored.

0: positive (or neutral) aspect of system

4: major, catastrophic aspect of system


### Heuristic Evaluation

#### List of 10 Heuristics according to (Nielsen, 2005):

<table>
<thead>
<tr>
<th></th>
<th>Heuristic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Visibility of system status</td>
</tr>
<tr>
<td>2</td>
<td>Match between the system and the real world</td>
</tr>
<tr>
<td>3</td>
<td>User control and freedom</td>
</tr>
<tr>
<td>4</td>
<td>Consistency and standards</td>
</tr>
<tr>
<td>5</td>
<td>Error prevention</td>
</tr>
<tr>
<td>6</td>
<td>Recognition rather than recall</td>
</tr>
<tr>
<td>7</td>
<td>Flexibility and efficiency of use</td>
</tr>
<tr>
<td>8</td>
<td>Aesthetic and minimalist design</td>
</tr>
<tr>
<td>9</td>
<td>Help users recognize, diagnose, and recover from errors</td>
</tr>
<tr>
<td>10</td>
<td>Help and documentation</td>
</tr>
</tbody>
</table>

---

**Error prevention:**

Even better than error messages is it to design the UI such that no errors occur, i.e., situations in which the end user might make a mistake should be avoided.

Before a transaction with potentially incorrect end user input is executed, a dialog seeking confirmation is useful.

---

http://www.useit.com/papers/heuristic/heuristic_list.html

Heuristic Evaluation

List of 10 Heuristics according to (Nielsen, 2005):

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Visibility of system status</td>
</tr>
<tr>
<td>2</td>
<td>Match between the system and the real world</td>
</tr>
<tr>
<td>3</td>
<td>User control and freedom</td>
</tr>
<tr>
<td>4</td>
<td>Consistency and standards</td>
</tr>
<tr>
<td>5</td>
<td>Error prevention</td>
</tr>
<tr>
<td>6</td>
<td>Recognition rather than recall</td>
</tr>
<tr>
<td>7</td>
<td>Flexibility and efficiency of use</td>
</tr>
<tr>
<td>8</td>
<td>Aesthetic and minimalist design</td>
</tr>
<tr>
<td>9</td>
<td>Help users recognize, diagnose, and recover from errors</td>
</tr>
<tr>
<td>10</td>
<td>Help and documentation</td>
</tr>
</tbody>
</table>

Recognition rather than recall:

The cognitive load of the end user can be reduced by making objects, actions, and options visible and recognizable.

End users should not be forced to remember information when moving from one dialog step to the next.

Help text should always be easily visible or, at least, easy to find.
Heuristic Evaluation

List of 10 Heuristics according to (Nielsen, 2005):

<table>
<thead>
<tr>
<th></th>
<th>Heuristic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Visibility of system status</td>
</tr>
<tr>
<td>2</td>
<td>Match between the system and the real world</td>
</tr>
<tr>
<td>3</td>
<td>User control and freedom</td>
</tr>
<tr>
<td>4</td>
<td>Consistency and standards</td>
</tr>
<tr>
<td>5</td>
<td>Error prevention</td>
</tr>
<tr>
<td>6</td>
<td>Recognition rather than recall</td>
</tr>
<tr>
<td>7</td>
<td>Flexibility and efficiency of use</td>
</tr>
<tr>
<td>8</td>
<td>Aesthetic and minimalist design</td>
</tr>
<tr>
<td>9</td>
<td>Help users recognize, diagnose, and recover from errors</td>
</tr>
<tr>
<td>10</td>
<td>Help and documentation</td>
</tr>
</tbody>
</table>

Flexibility and efficiency of use:

Shortcuts and special keys – often not known by novice users – can be helpful to expert users and make their interaction with the system more efficient.

The system should be usable according to the needs of different user groups (=flexibility).


### Heuristic Evaluation

List of 10 Heuristics according to (Nielsen, 2005):

1. Visibility of system status
2. Match between the system and the real world
3. User control and freedom
4. Consistency and standards
5. Error prevention
6. Recognition rather than recall
7. Flexibility and efficiency of use
8. Aesthetic and minimalist design
9. Help users recognize, diagnose, and recover from errors
10. Help and documentation

**Aesthetic and minimalist design:**

(Standard) dialogs shouldn’t contain any irrelevant or rarely needed information. Every unnecessary information competes for the attention of the end user with the relevant information and thus diminishes its visibility.

**Aesthetics:** color, alignment, contrast, proximity.
Heuristic Evaluation

List of 10 Heuristics according to (Nielsen, 2005):

<table>
<thead>
<tr>
<th></th>
<th>Heuristic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Visibility of system status</td>
</tr>
<tr>
<td>2</td>
<td>Match between the system and the real world</td>
</tr>
<tr>
<td>3</td>
<td>User control and freedom</td>
</tr>
<tr>
<td>4</td>
<td>Consistency and standards</td>
</tr>
<tr>
<td>5</td>
<td>Error prevention</td>
</tr>
<tr>
<td>6</td>
<td>Recognition rather than recall</td>
</tr>
<tr>
<td>7</td>
<td>Flexibility and efficiency of use</td>
</tr>
<tr>
<td>8</td>
<td>Aesthetic and minimalist design</td>
</tr>
<tr>
<td>9</td>
<td>Help users recognize, diagnose, and recover from errors</td>
</tr>
<tr>
<td>10</td>
<td>Help and documentation</td>
</tr>
</tbody>
</table>

Help users recognize, diagnose, and recover from errors:

- Error messages should use simple language, not contain cryptic codes, clearly identify the problem, and make a constructive solution to solve the problem.

Rank by severity: 0...4
- 0: positive (or neutral) aspect of system
- ... 4: major, catastrophic aspect of system


### Heuristic Evaluation

#### List of 10 Heuristics according to (Nielsen, 2005):

<table>
<thead>
<tr>
<th>No.</th>
<th>Heuristic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Visibility of system status</td>
</tr>
<tr>
<td>2</td>
<td>Match between the system and the real world</td>
</tr>
<tr>
<td>3</td>
<td>User control and freedom</td>
</tr>
<tr>
<td>4</td>
<td>Consistency and standards</td>
</tr>
<tr>
<td>5</td>
<td>Error prevention</td>
</tr>
<tr>
<td>6</td>
<td>Recognition rather than recall</td>
</tr>
<tr>
<td>7</td>
<td>Flexibility and efficiency of use</td>
</tr>
<tr>
<td>8</td>
<td>Aesthetic and minimalist design</td>
</tr>
<tr>
<td>9</td>
<td>Help users recognize, diagnose, and recover from errors</td>
</tr>
<tr>
<td>10</td>
<td>Help and documentation</td>
</tr>
</tbody>
</table>

---

**Help and documentation:**

Ideally, a system should be usable without documentation (self-explanatory), sometimes it is useful to offer help.

Help texts and documentation should be easy to find and search.

Help texts and documentation should focus on the end users’ (typical) needs and offer concrete steps on how to use the system for a specific task.
Evaluating UI Designs

- Inspection Methods
  - Heuristic Evaluation
    - Cognitive Walkthrough
  - Guidelines Review
- Usability Testing
  - Laboratory Experiment
  - Field Study
Evaluating UI Designs

Written guidelines recommended for larger projects:
- Screen layout
- Appearance of objects
- Terminology
- Wording of prompts and error messages
- Menu’s
- Direct manipulation actions and feedback
- On-line help and other documentation
Guidelines for Screen Layout & Appearance

P.A.R.C.

- Proximity
- Alignment
- Repetition
- Contrast
Guidelines for Screen Layout & Appearance

P.A.R.C.

- Proximity
  - Objects that are close to another (related) appear to form groups.

- Alignment
- Repetition
- Contrast
Guidelines for Screen Layout & Appearance

P.A.R.C.
- Proximity
- Alignment
- Repetition
- Contrast

Alignment of objects generates clarity / avoids confusion.

- O-O-O-O-O-O-O-O-O-O-O
  in ALIGNMENT

- O-O-O-O-O-O-O-O-O-O-O
  out of ALIGNMENT
Guidelines for Screen Layout & Appearance

P.A.R.C.
- Proximity
- Alignment
- Repetition
- Contrast

Visual aspects such as color, shape and size link (similar) items together.
Guidelines for Screen Layout & Appearance

P.A.R.C.

Proximity
Alignment
Repetition
Contrast

Contrast makes differences easier recognizable.
Guidelines for Screen Layout & Appearance

P.A.R.C.

- Proximity
- Alignment
- Repetition
- Contrast

Contrast makes differences easier recognizable. Use: size, color, value, shape, position, direction, …
Evaluating UI Designs

Inspection Methods
- Heuristic Evaluation
  - Cognitive Walkthrough
  - Guidelines Review

Usability Testing
- Laboratory Experiment
  - Field Study
Evaluating UI Designs

Usability testing in a controlled environment
- There is a test set of users
- They perform pre-specified tasks
- Data is collected (quantitative and qualitative)
- Take mean and/or median value of measured attributes
- Compare to goal or another system
Evaluating UI Designs

1. Direct observation in actual use
   discover new uses
   take notes, don’t help, chat later
2. Logging actual use
   objective, not intrusive
   great for identifying errors
   which features are/are not used
   privacy concerns

Cognitive Walkthrough
Guidelines Review

Usability Testing
Field Study
Laboratory Experiment

Evaluating UI Designs
1. Direct observation in actual use
   discover new uses
   take notes, don’t help, chat later
2. Logging actual use
   objective, not intrusive
   great for identifying errors
   which features are/are not used
   privacy concerns

Cognitive Walkthrough
Guidelines Review

Usability Testing
Field Study
Laboratory Experiment

Evaluating UI Designs
1. Direct observation in actual use
   discover new uses
   take notes, don’t help, chat later
2. Logging actual use
   objective, not intrusive
   great for identifying errors
   which features are/are not used
   privacy concerns

Cognitive Walkthrough
Guidelines Review

Usability Testing
Field Study
Laboratory Experiment

Evaluating UI Designs
1. Direct observation in actual use
   discover new uses
   take notes, don’t help, chat later
2. Logging actual use
   objective, not intrusive
   great for identifying errors
   which features are/are not used
   privacy concerns

Cognitive Walkthrough
Guidelines Review

Usability Testing
Field Study
Laboratory Experiment

Evaluating UI Designs
1. Direct observation in actual use
   discover new uses
   take notes, don’t help, chat later
2. Logging actual use
   objective, not intrusive
   great for identifying errors
   which features are/are not used
   privacy concerns

Cognitive Walkthrough
Guidelines Review

Usability Testing
Field Study
Laboratory Experiment

Evaluating UI Designs
1. Direct observation in actual use
   discover new uses
   take notes, don’t help, chat later
2. Logging actual use
   objective, not intrusive
   great for identifying errors
   which features are/are not used
   privacy concerns

Cognitive Walkthrough
Guidelines Review

Usability Testing
Field Study
Laboratory Experiment

Evaluating UI Designs
1. Direct observation in actual use
   discover new uses
   take notes, don’t help, chat later
2. Logging actual use
   objective, not intrusive
   great for identifying errors
   which features are/are not used
   privacy concerns

Cognitive Walkthrough
Guidelines Review

Usability Testing
Field Study
Laboratory Experiment

Evaluating UI Designs
1. Direct observation in actual use
   discover new uses
   take notes, don’t help, chat later
2. Logging actual use
   objective, not intrusive
   great for identifying errors
   which features are/are not used
   privacy concerns

Cognitive Walkthrough
Guidelines Review

Usability Testing
Field Study
Laboratory Experiment
Evaluating UI Designs

3. Questionnaires and interviews with real users
   ask users to recall critical incidents
   questionnaires must be short and easy to return

4. Focus groups
   6-9 users
   skilled moderator with pre-planned script
   computer conferencing??

5. On-line direct feedback mechanisms
   initiated by users
   may signal change in user needs
   trust but verify

6. Bulletin boards and user groups
UI Design & Usability

Recommendation for those who want to know more about UI Design (and Usability):

• Course: LTAT.05.007 - Human Computer Interaction
• Teacher: Alexander Nolte
Structure of Lecture 8

- GUI Testing
  - GUITAR
  - TESTAR
- Visual Testing
  - Sikuli IDE
  - Lab 7
- Security Testing
- Usability Testing
- A/B Testing
A/B Testing

Two GUI Versions A & B
A/B Testing (cont’d)

Tool support

Visual Website Optimizer divides traffic between the two versions

Two GUI Versions A & B

Randomly selected
A/B Testing (cont’d)

Tools:
https://blog.crazyegg.com/2014/06/25/best-testing-software/

More Sales?

Two GUI Versions A & B
A/B Testing – Real-World Example

Former US president Obama’s 2008 Election campaign

A/B Testing – Real-World Example

Button variations:
A/B Testing – Real-World Example

Media variations:

- Family Image
- Change Image
- Barack’s Video
- ...

(6 alternatives in total)
A/B Testing – Real-World Example

In total
4 x 6 = 24 combinations
(including the original button and medium)

<table>
<thead>
<tr>
<th>Button</th>
<th>Variation</th>
<th>Est. conv. rate</th>
<th>Chance to Beat Orig.</th>
<th>Observed Improvement</th>
<th>Conv./Visitors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td></td>
<td>7.51% ± 0.2%</td>
<td>–</td>
<td>–</td>
<td>5851 / 77858</td>
</tr>
<tr>
<td>Learn More</td>
<td></td>
<td>8.91% ± 0.2%</td>
<td>100%</td>
<td>18.6%</td>
<td>6927 / 77729</td>
</tr>
<tr>
<td>Join Us Now</td>
<td></td>
<td>7.62% ± 0.2%</td>
<td>73.5%</td>
<td>1.37%</td>
<td>5915 / 77644</td>
</tr>
<tr>
<td>Sign Up Now</td>
<td></td>
<td>7.34% ± 0.2%</td>
<td>13.7%</td>
<td>-2.38%</td>
<td>5660 / 77151</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Media</th>
<th>Variation</th>
<th>Est. conv. rate</th>
<th>Chance to Beat Orig.</th>
<th>Observed Improvement</th>
<th>Conv./Visitors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td></td>
<td>8.54% ± 0.2%</td>
<td>–</td>
<td>–</td>
<td>4425 / 51794</td>
</tr>
<tr>
<td>Family Image</td>
<td></td>
<td>9.66% ± 0.2%</td>
<td>100%</td>
<td>13.1%</td>
<td>4996 / 51696</td>
</tr>
<tr>
<td>Change Image</td>
<td></td>
<td>8.87% ± 0.2%</td>
<td>92.2%</td>
<td>3.85%</td>
<td>4595 / 51790</td>
</tr>
<tr>
<td>Barack’s Video</td>
<td></td>
<td>7.76% ± 0.2%</td>
<td>0.04%</td>
<td>-9.14%</td>
<td>3992 / 51427</td>
</tr>
<tr>
<td>Sam’s Video</td>
<td></td>
<td>6.29% ± 0.2%</td>
<td>0.00%</td>
<td>-26.4%</td>
<td>3261 / 51864</td>
</tr>
<tr>
<td>Springfield Video</td>
<td></td>
<td>5.95% ± 0.2%</td>
<td>0.00%</td>
<td>-30.3%</td>
<td>3084 / 51811</td>
</tr>
</tbody>
</table>

Sign-up rates for each section (button, medium)
A/B Testing – Real-World Example

In total
4 x 6 = 24 combinations
(including the original button and medium)

Combination 11:
“Learn More” & “Family Image”

Sign-up rates for each <button, medium>-combination
A/B Testing – Real-World Example

Former US president Obama’s 2008 Election campaign

A/B Testing – Multivariate Testing

- Only 2 items in previous example
A/B Testing (cont’d)

What to vary …
• Call-To-Actions – Placement, wording, size
• Copywriting – Value propositions, product descriptions
• Forms – Their length, field types, text on the forms.
• Layout – Homepage, content pages, landing pages
• Product pricing – Try testing for revenue by testing your prices
• Images/Videos – Their placement, content and size
• Amount of content on the page (short vs. long)

Link:
http://conversionxl.com/how-to-build-a-strong-ab-testing-plan-that-gets-results/

In Obama example:
A/B Testing -- Tools

A/B Split Test Significance Calculator by VWO

• A widely used tool for calculating the significance of your A/B testing results.

A/B Split and Multivariate Test Duration Calculator by VWO

• The calculator allows you to calculate maximum duration for which your test should run.

Crazyegg, Inspectlet, Clicktale and Mouseflow

• Heatmap software for tracking your visitor’s behavior on your site. You can get good data for hypotheses generation.
Next Week

• Quiz 8 → Moodle

• Lab 7:
  – Visual Testing

• Lecture 9:
  White-Box Testing (advanced):
  – Data-Flow Testing & Mutation Testing