Lecture 6: Exploratory Testing

Gregory Gay
DIT635 - February 7, 2020

How do you come up with tests?

Today's Goals

- Test Plans
  - Document describing testing process and scope.
- Exploratory Testing
  - Human-driven testing of the project, to gain familiarity with the system and conduct high-level testing.
  - Often focused on "tours" of the software features.

Test Plans

Test Plans

- Plan for how we will test the system.
  - **What** is being tested (units of code, features).
  - **When** it will be tested (required stage of completion).
  - **How** it will be tested (what scenarios do we run?).
  - **Where** we are testing it (types of environments).
  - **Why** we are testing it (what purpose do tests serve?).
  - **Who** will be responsible for writing test cases (assign responsibility to team members).

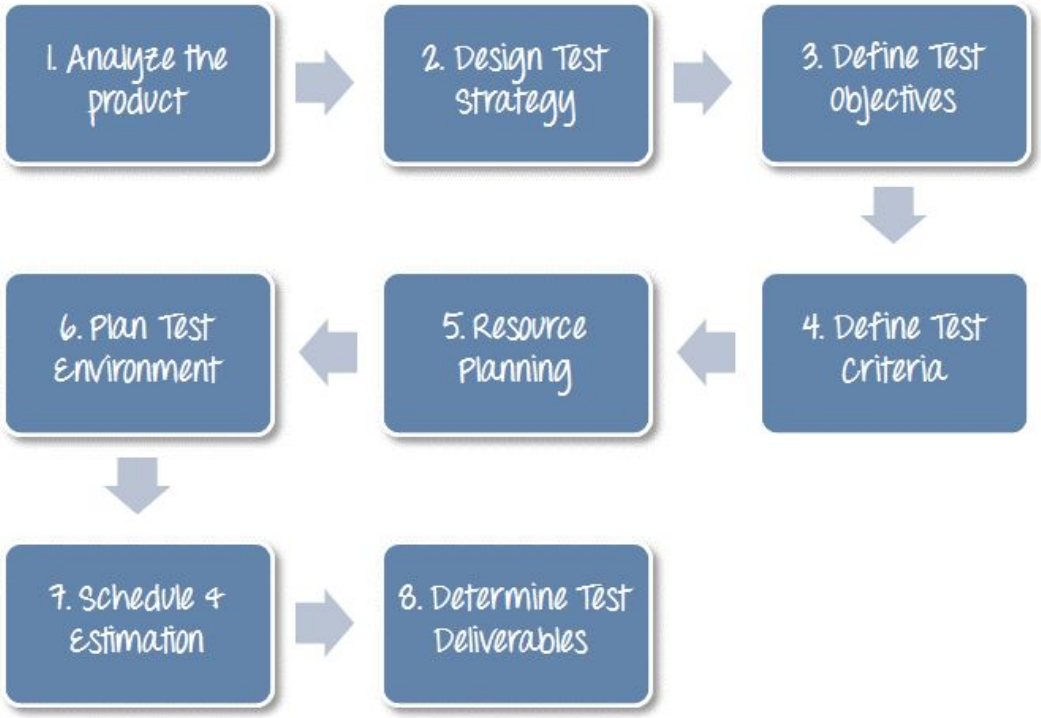
Where Does a Test Plan Come From?

- Most test plans are derived from requirement specifications.
  - The specification defines “correct” behavior.
  - The specification exists in some form before code is written, and guides development.
  - Test plans and cases can be developed and refined as the code is built.
- **Functional Testing:** The process of deriving tests from the requirement specifications.

Why Make a Test Plan?

- Guides the development team.
  - Rulebook for planning test cases.
- Helps people outside the team understand the testing process.
- Documents rationale for scope of testing, how we judge results, why we chose a strategy.
  - Can be reused when making decisions in future projects.

Writing a Test Plan



Slides derived from https://www.guru99.com/what-everybody-ought-to-know-about-test-planing.html

Analyze the Product

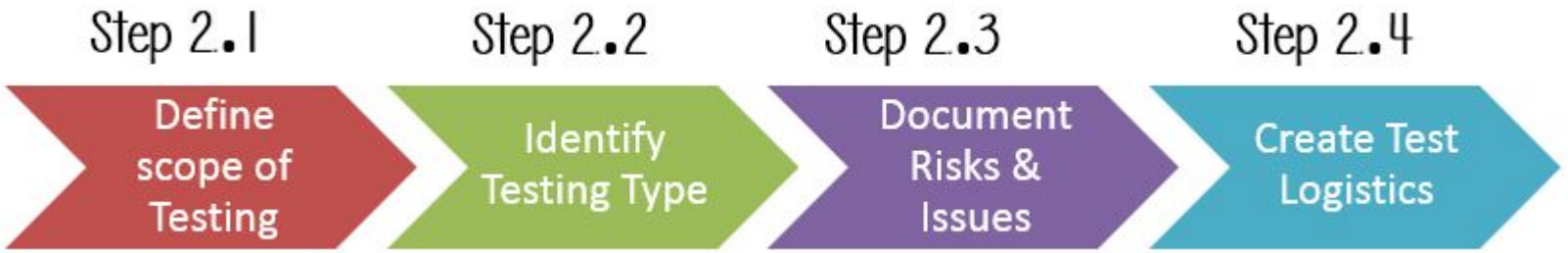
- You must understand the product being built before you can test it.
  - What are the needs of the users?
  - Who will use the product?
  - What will it be used for?
  - What are the dependencies of the product?
- Review requirements and documentation.
- Interview stakeholders and developers.
- Perform a product walkthrough (if code is running).

Analyze the Product

- Banking Website
  - What features do we want to see?
  - Account creation, deletion, manipulation.
  - Fund transfers
  - Fund withdrawal
  - Check deposit
  - …?

Develop the Test Strategy

- Document defining:
  - Test Objectives (and how to achieve them)
  - Testing Effort and Cost



Testing Scope

- **What are you planning to test?**
  - Software, hardware, middleware, ...
- … AND… What are you **NOT** going to test?
  - Gives project members a clear understanding about what you are responsible for.
- Must take into account:
  - Requirements
  - Budget
  - Skills of your testing team

Testing Scope

- Banking website
  - Requirements specified only for functionality and the external interface.
    - **These are in-scope.**
  - No requirements for database or client hardware.
  - No quality requirements (performance, availability).
    - **These are out-of-scope.**

Identify Testing Types

For the banking site:

- API Testing
  - Focus on verifying access points and interfaces.
- Integration Testing
- System Testing
  - Functionality likely spread over multiple classes, many features interact
- Acceptance Testing

Could limit:

- Unit Testing (final product matters more than individual classes)

Can skip:

- Install/Uninstall (web app)

- Which should we apply?
  - Consider the project domain.
- Which can we skip or limit to save money?

Document Risks and Issues

- Risks - events that could prevent the success of testing. Have probability of occurrence and effect.
  - Issues are risks that have already impacted project.
- Document risks and how to avoid them or reduce their impact.

Document Risks and Issues

Risk	Mitigation
Team lacks required skills for website testing	Plan training course on web testing
The project schedule is too tight	Prioritize each testing activity
New security vulnerabilities may appear	Review attack reports on competing websites and update practices at regular intervals
A lack of cooperation negatively affects your employees' productivity	Encourage each team member, and inspire them to greater efforts
Wrong budget estimate and cost overruns	Establish the scope before beginning work, pay a lot of attention to project planning and constantly track and measure the progress

Slides derived from https://www.guru99.com/what-everybody-ought-to-know-about-test-planing.html

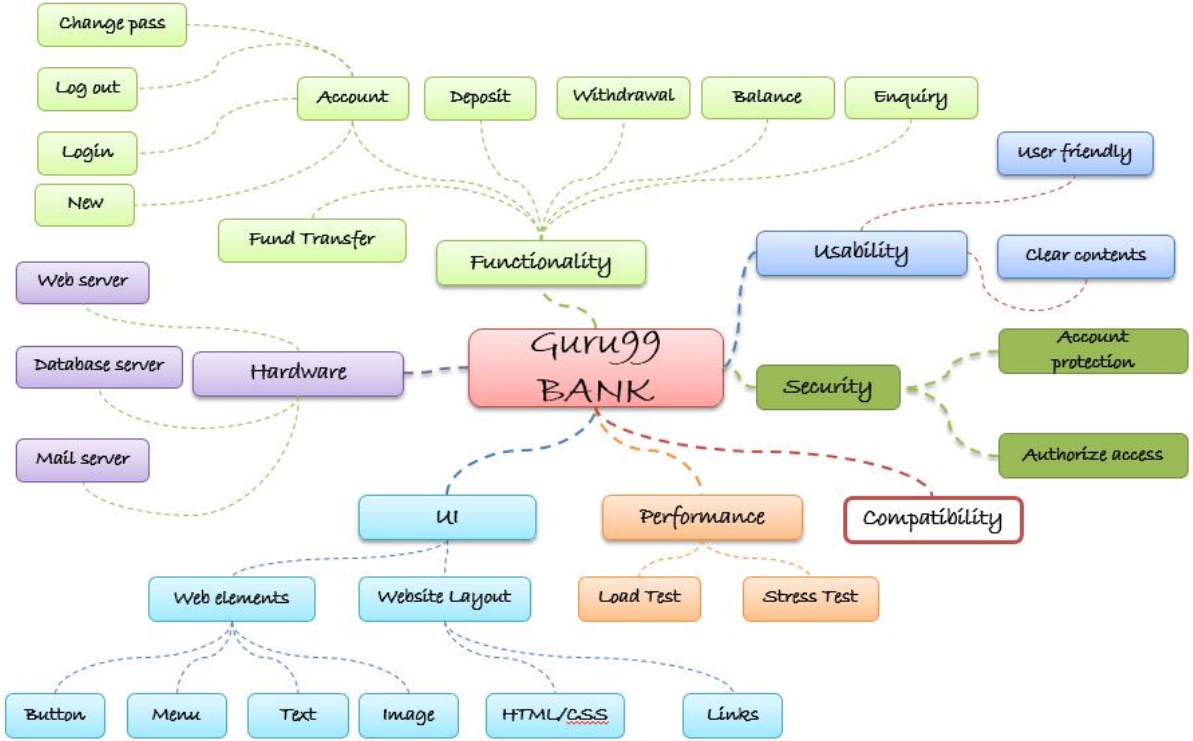
Create Test Logistics

- Who will write and execute test cases?
  - What types of testers do you need?
    - Skills needed for the targeted domain
  - What is the budget for testing?
    - How many people can you hire to test?
- When will each testing activity occur?
  - When to design and when to execute tests.
  - Pair with appropriate stage of development.
    - Unit development -> unit testing -> system testing -> ...

Define Test Objectives

- What are the goals of the testing process?
  - What features need to be tested?
  - What system elements need to be tested?
  - What quality attributes do we need to demonstrate?
  - For each feature or quality, what scenarios do we want to walk through?
- Does not include a list of specific tests
  - But, at a high level, should detail scenarios we plan to examine by writing one or more test cases.

Define Test Objectives



Slides derived from https://www.guru99.com/what-everybody-ought-to-know-about-test-planing.html

Define Test Criteria

- When have we completed our testing objectives?
  - For qualities, set appropriate thresholds.
    - Availability, ROCOF, throughput, etc.
  - For functionality, commonly defined using:
    - **Run Rate: Number of Tests Executed / Number Specified**
    - **Pass Rate: Number of Passing Tests / Number Executed**
    - Often aim for 100% run rate and a high pass rate (> 95%)

Resource Planning

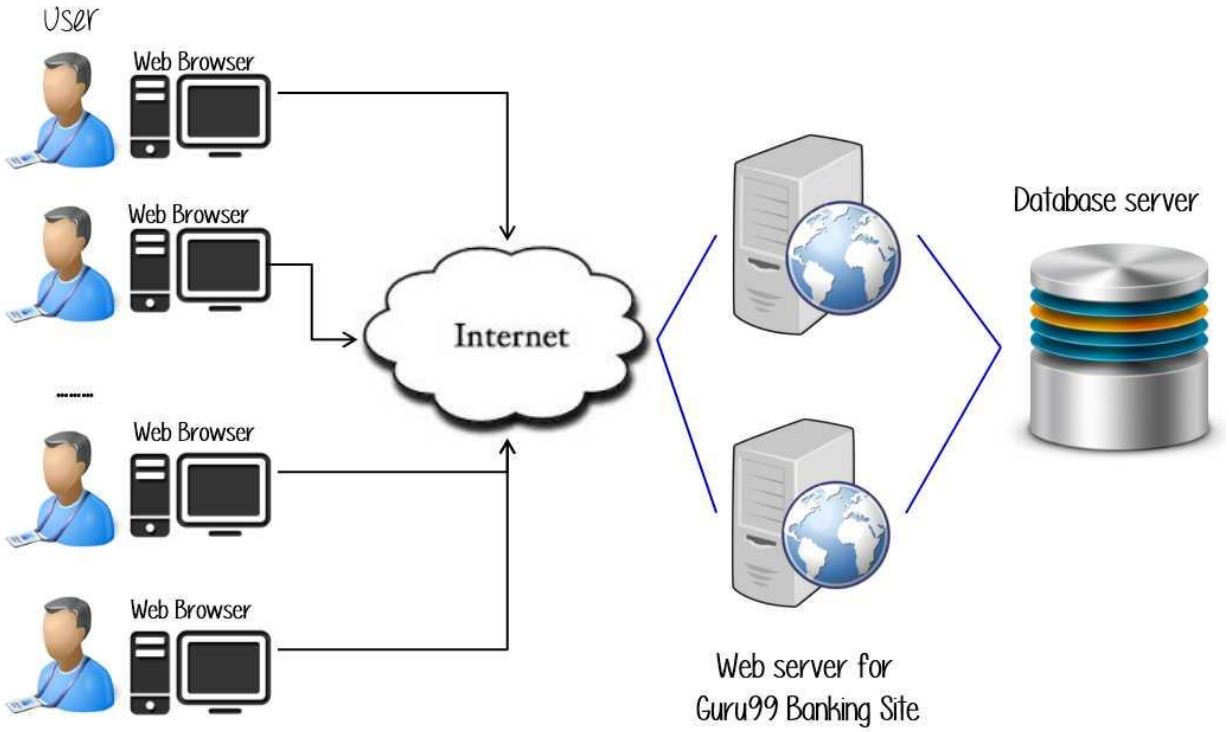
- Summarize all resources that you have to complete the testing.
  - Allows estimation and adjustment of testing scope, objectives, and exit criteria.
- Human Resources: Managers, testers, developers who assist in testing, system administration.
- System Resources: Servers, testing tools, network resources, physical hardware.

Plan Test Environment

- Where will you execute test cases?
  - Software and hardware execution environment
  - Often defined as part of continuous integration.
- Need to account for:
  - Requirements on both server and client-side.
  - Different networking conditions (bandwidth, load).
  - Different client or server-side hardware.
  - Different numbers of concurrent users.

Plan Test Environment

- When testing, we might vary:
  - Number of clients, servers
  - Networking conditions
  - Contents of database server.
  - Connection to database server.



Schedule Estimation

- Break testing plans into individual tasks, each with an effort estimation (in person-hours)
  - Create test specification, 170 person-hours
  - Write unit tests, 80 person-hours
  - Write API tests, 50 person-hours
  - Perform test execution, 1 person-hour (per suite execution)
  - Write test report, 10 person-hours
  - …

Plan Test Deliverables

- Before: Test plan document, test specifications
- During: Executable tests, simulators, test input data, traceability matrix, error and execution logs.
- After: Test report, fault reports, installation and test procedure guidelines, release notes

Let’s take a break.

Exploratory Testing

Exploratory Testing

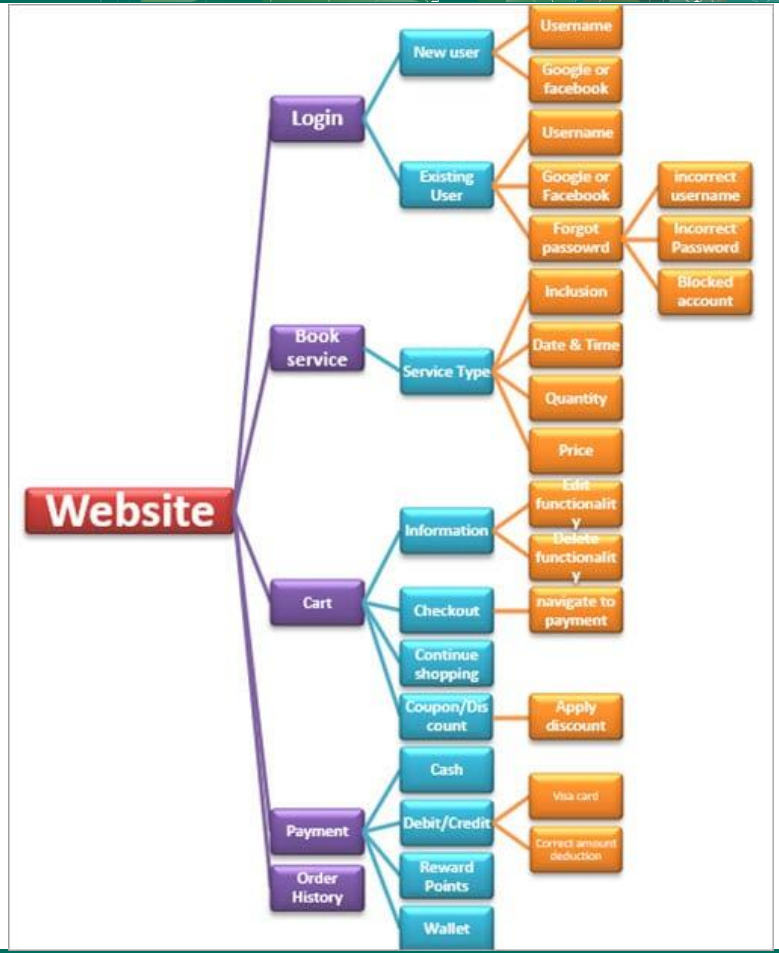
- Tests are not created in advance.
- Testers check the system on-the-fly,
  - Often based on ideas noted before beginning.
- Testing as a thinking idea.
  - About discovery, investigation, and role-playing.
- Test design and execution done concurrently.
  - Often by directly using the software and its user interfaces.

Exploratory Testing

- Tester will write down an idea to give direction, then explore the system to create critical, practical, and userful tests.
  - Requires minimal planning. Tester chooses next action based on result of current action.
- Can find subtle faults missed by formal testing.
  - Allows tester to better learn system functionality, and identify new ways of using features.

Example

- Start with functionality you know well (Login)
- Examine possible options and list them.
- Use your findings to plan the next steps.
- As you learn and observe, more test cases will emerge.



Session-Based Exploratory Testing

- Time-based method to structure exploratory testing.
  - Conducted with no e-mail, phone, messaging.
  - Short (60min), Normal (90m), Long (120m)
- Primary components:
  - **Mission**
    - The purpose of the session.
    - Provides focus for the tester.
  - **Charter**
    - Individual testing goals to be completed in this session.
    - Could be a list of features or scenarios.

Session Report Items

- **Mission:** Overall goal
  - “Analyze Login Feature on Website”
- **Charter:** Features and scenarios to focus on.
  - “Login as existing user with username and password”
  - “Login as existing user with Google account”
  - “Login as existing user with Facebook account”
  - “Enter incorrect username and password to verify validation message”
  - “Block your username and verify the validation message”
  - “Use Forgot Password link to reset password”

Session Report Items

- **Start and end time of session**
- **Duration of session**
- **Testing notes:** journal of actions taken
  - Opened login page
    - Verified default screen.
    - Verified that existing and new user account links exist.
  - Opened existing user login
    - Verified successful login with username, Google, and Facebook.
    - Verified validation messages.

Session Report Items

- Include **data files** (screenshots, recordings, files), especially if a bug is found.
- **Fault Information:** Describe each fault found. File a bug report for each, include tracker ID in report.
- **Issues Information:** If an issue prevents or complicates testing, describe it in the report.
- **Set-up Time:** % of time required to set-up.
- **Test Design and Execution Time:** % of time spent purely on testing

Session Debrief

- Short meeting between tester and manager to review the findings.
- Session-based testing allows tracking of time spent testing, number of bugs reported, time spent on set-up, time spent on testing, time spent analyzing issues, features covered.
- Allows management of the time spent on exploratory testing and process observability.

Tips for Exploratory Testing

- Divide the application into modules or features, then try to further divide.
  - Start ET from the smallest subdivisions.
  - This will give greater coverage.
- Make a checklist of all the features and put a check mark when each is covered.
- Start with a basic scenario and then gradually enhance it to add more features to test it.

Tips for Exploratory Testing

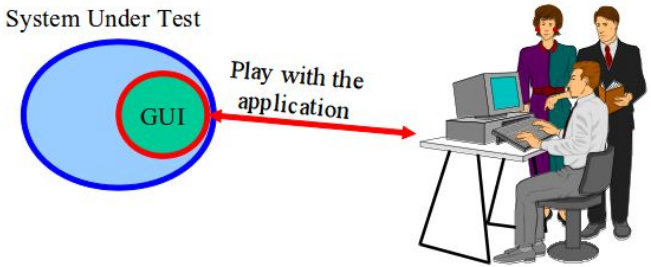
- Test all input fields.
- Check for all possible error messages.
- Test all the negative scenarios.
  - Invalid input, mistakes in usage.
- Check the GUI against standards.
- Check the integration of the application with other external applications.
- Check for complex business logic.
- Try to do the ethical hacking of the application.

Pair-Based Exploratory Testing

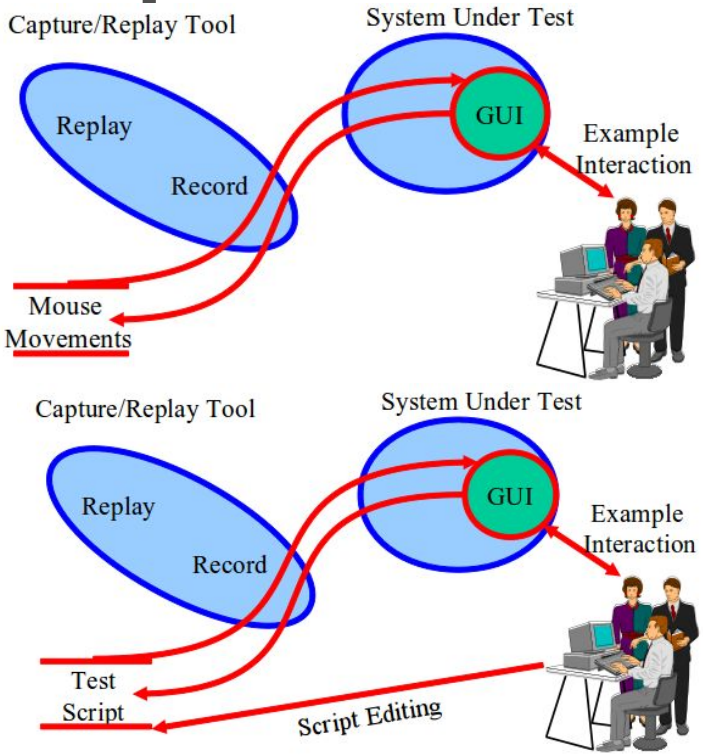
- Two people share a computer and test together.
  - One person uses the computer, the other suggests actions and takes notes.
  - Each brings their own strengths.
  - Can be used to train new developers or testers.
- Benefits of pair testing:
  - Increases focus.
  - Leads to more constructive ideas.
  - Avoids biased input selection.

Automating Exploratory Testing

- Use tools to streamline bug reporting and reproduction, snapshots, preparation of executable test suites for future use.
- A tool captures and records the activities performed by the tester.
  - Called **capture and replay tools**.



Capture and Replay Tools



- Tool records input (key presses, mouse clicks, touches, etc.) during exploratory testing.
  - The “Capture”
- Capture can be replayed to reproduce outcomes and crashes.
- Capture scripts can be extended and altered to form new test cases.

Automating Exploratory Testing

- Provides clear steps to reproduce bug.
- Can also be used to judge performance.
- Often used in pair exploratory testing.
  - Second tester watches replay from first tester.
  - Second tester looks for ways to extend the tests.
  - The first tester does the same with the second tester's replay.
  - Exchange again at the end to confirm results.

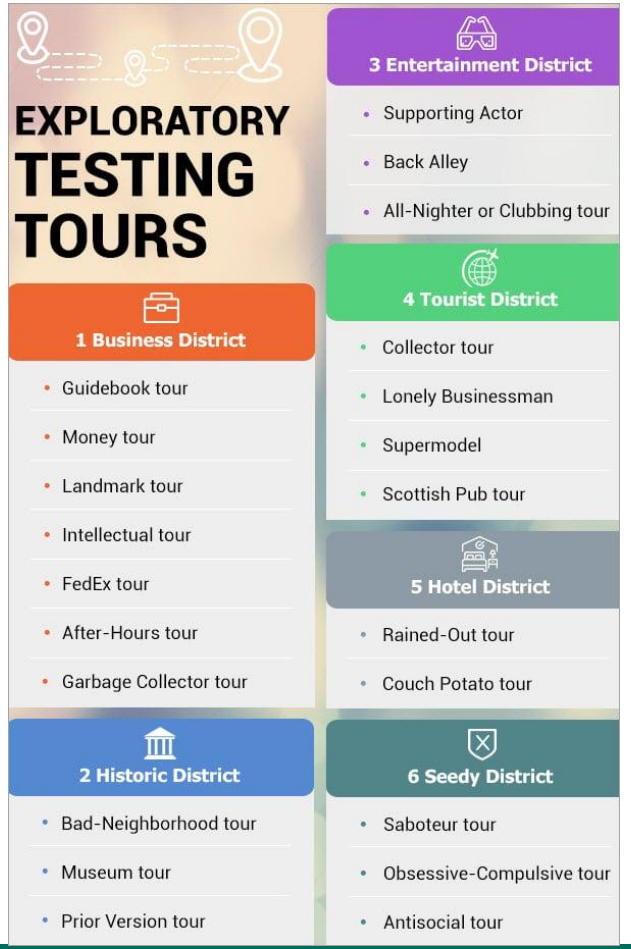
Let’s take a break.

Using “Tours” in Exploratory Testing

- A tourist seeks to visit as many districts of a city as possible within the time budget.
  - In software, the “city” is the system, and the “districts” are aspects of the system.
    - Business district = core features, entertainment district = supporting features
- A **tour** is a plan for exploratory testing.
  - Includes a set of objectives, based on visiting different “districts”, to focus on during testing.
  - A typical tour should take **less than four hours**.

Exploratory Tours

- Software split into aspects (districts)
- Several “tours” related to each district.
  - Each prescribes a way of exploring the software.



Business District

- Focused on the features designed to increase sales
- Tours vary in focus, but all are intended to give a good impression of the software (to attract sales).
- **Guidebook Tour**
  - Focused on users that strictly follow the steps in support information, instructions, and user manuals.
  - Walk through all guides to the software.
  - Look for unclear or incorrect instructions, steps that do not correspond to actual workflow, uninformative prompts

Business District

- **Money Tour**
  - Advertising material should be accurate.
  - Compare advertising to the real software.
  - Look for outdated screenshots or lists of features, grammar mistakes, logical mistakes.
- **Landmark Tour**
  - Cities have famous “highlights”.
  - Define a list of “core” software features and execute each step-by-step.
  - Try all critical functionality, and detect blocking faults.

Business District

- **Intellectual Tour**
  - Try to “outsmart” the program.
  - Enter long filenames, huge number of products to shopping card, invalid form data.
- **FedEx Tour**
  - Packages get passed around and handled by many people. Likewise, you can examine how data is passed around the software.
  - Validate data after operations that manipulate it.
  - Look at results of serialization/deserialization.

Business District

- **After-Hours Tour**
  - When the user closes a program, that does not mean it stops working.
  - System can backup, archive data, apply updates, etc.
  - Tester verifies the success of background tasks.
  - Monitor background operations, looking for irregular traffic, memory leaks.

Historic District

- Towns have historic areas with older buildings.
- Software often relies on legacy code, previous product versions, older dependencies.
- **Bad-Neighborhood Tour**
  - Focus on fragments of the software where bugs have been previously discovered.
  - Ensure that older bugs are solved.
  - Look for new bugs exposed by fixing that code.

Historic District

- **Museum Tour**
  - Museums are parts of the program that have not been changed in some time.
  - May not work properly given changes to the rest of the system, and may not have been retested.
  - Focus on verifying that they still work.
- **Prior Version Tour**
  - Software updates intended to make UI/functionality better
  - Compare old and new version to ensure positive changes

Entertainment District

- Entertainment districts fill in the gaps in a vacation when you want to relax.
- In software, this represents supporting features that aren't part of critical functionality.
  - Word processor
    - Business District: Constructing a document (inserting text, images), Entertainment District: Make it look nice (format layout)
- Tours visit supporting features and ensures they are properly intertwined with core features.

Entertainment District

- **Supporting Actor Tour**
  - Focus on features that share screen with core features.
    - Proximity to core features mean they will be used.
  - Ex: product search shows reviews and similar items. Make sure those features work as well, as they might get used following a search.
- **Back Alley Tour**
  - Focus on the least commonly-used features.
  - Don’t devote much time to these, but they should still be tested. Somebody will depend on them.
  - **Mixed Destination** tour combines common/uncommon.

Entertainment District

- **All-Nighter Tour**
  - How long can the system run before collapsing?
  - Bad things tend to happen over time - memory leaks, data corruption, race conditions.
  - Closing and reopening program resets the slate.
  - Try to run it without restarting for as long as possible, to see how robust the system is over time.
  - Often uses dedicated machines that never get turned off, running replays over and over.

Tourist District

- Capture the experience of being a tourist - based on collected souvenirs, guided tours, etc.
- **Collector's Tour:**
  - Try each piece of functionality/scenario.
  - Collect the output.
  - Look at the output to see if you can come up with more scenarios.
  - Repeat the process until you can't think of anything else.

Tourist District

- **TOGOF Tour (Test-One-Get-One-Free):**
  - Run multiple copies of the program at once.
  - Try to open the same file in each, or cause them to work with memory/disk IO.
  - See if having multiple copies open can cause issues.
- **Scottish Pub Tour:**
  - Try to find out-of-the-way features or locations that only an expert would ever have stumbled into.
  - Look at Stack Overflow, forums, blogs for stories about weird cases with the software. Recreate those, and look for other obscure features that are undertested.

Hotel District

- Focuses on secondary or supporting functionality.
- **Rained-Out Tour:**
  - Look for operations that can be cancelled.
  - Feed them input that will cause a long operation.
  - Cancel midway through, and see if everything still works.
  - Good for finding failures related to the program's inability to clean up after itself.
    - Open files, corrupted memory or state.
    - Even if there is no cancel button, can always ctrl-c, alt-f4, click the "x" in the corner, etc.

Hotel District

- **Couch Potato Tour:**
  - Tester does the least interaction possible when executing the code.
  - Leave default values in place, leave input fields blank, try to move forward without offering much data and without doing much work.
  - Ensures the software must execute code for processing blank or partial information, must handle defaults.
    - We try so many complicated scenarios that we can miss or forget the defaults.

Seedy District

- Focused on attacking and breaking the system.
- **Saboteur Tour:**
  - Force the software to act.
  - Understand the resources it requires to successfully act.
  - Remove of restrict those resources.
  - Use corrupt input data, limit network connectivity, allow too little RAM, run many other apps at the same time.
  - Think of ways to creatively disrupt operations and try them out.

Seedy District

- **Antisocial Tour:**
  - **Opposite Subtour:**
    - Try the least likely input, or actions that make almost no sense.
    - Add 10000 songs to a playlist, or play an empty playlist.
    - Try to order 100000000 pairs of shoes.
  - **Crime Spree Subtour:**
    - Enter illegal data (SQL injections)
  - **Wrong Turn Subtour:**
    - Use legal input, but try to perform actions in the wrong order.

We Have Learned

- Test Plans form the basis for testing.
  - **What** is being tested (units of code, features).
  - **When** it will be tested (required stage of completion).
  - **How** it will be tested (what scenarios do we run?).
  - **Where** we are testing it (types of environments).
  - **Why** we are testing it (what purpose do tests serve?).
  - **Who** will be responsible for writing test cases (assign responsibility to team members).

We Have Learned

- Exploratory Testing
  - Tests are not created in advance.
  - Testers check the system on-the-fly,
    - Often based on ideas noted before beginning.
  - Testing as a thinking idea.
    - About discovery, investigation, and role-playing.
  - Test design and execution done concurrently.
    - Often by directly using the software and its user interfaces.

We Have Learned

- Tours apply different focus areas to exploration
  - Business District: Core features
  - Historic District: Legacy code and old software versions
  - Entertainment District: Supporting functionality, long execution sessions
  - Tourist District: Looks for gaps in the experience, rarely seen scenarios
  - Hotel District: Focuses on supporting functionality
  - Seedy District: Attacks and misuse of software

Next Time

- Exercise Session: Unit Testing
- Next class: Functional Testing
  - Pezze and Young, Chapters 10-11

- Assignment 1 due February 16

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY