
# Lecture 7: Exploratory Testing

Gregory Gay
DIT635 - February 9, 2022

# Today's Goals

- Introduce 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.

# Exploratory Testing

- Testers check the system on-the-fly.
  - Guided by scenarios.
  - Often based on ideas noted before beginning.
- Testing as a thinking idea.
  - About discovery, investigation, and role-playing.
  - Tests end-to-end journeys through app.
  - Test design and execution done concurrently.

# Automation vs Human-Driven

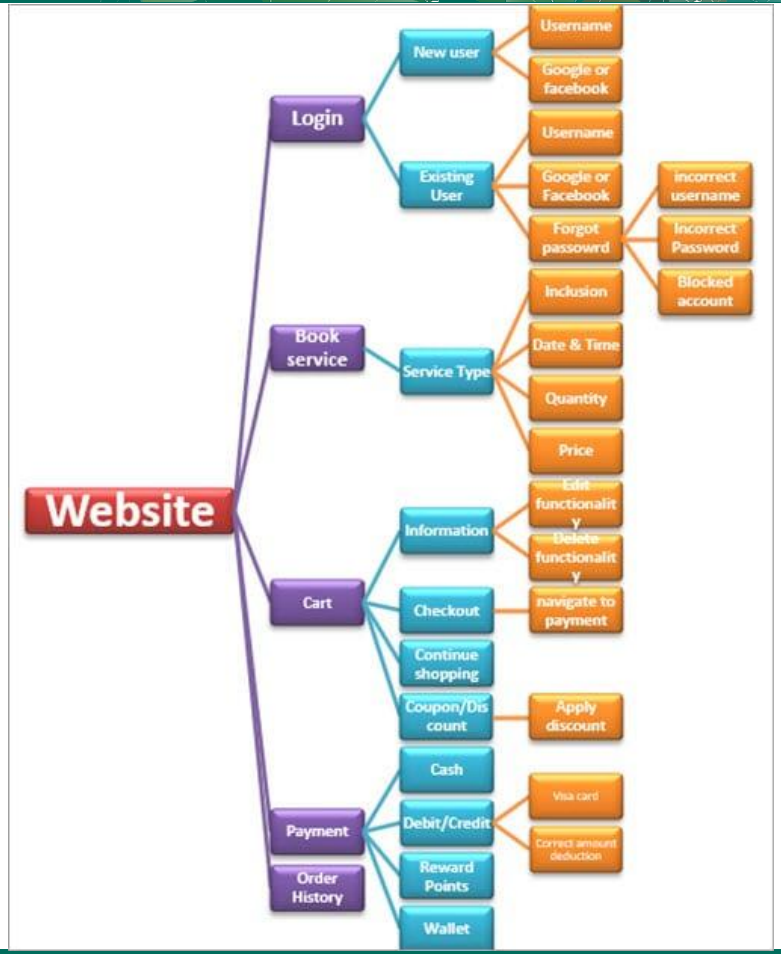
- Unit/System Testing heavily use automation.
  - Tests written as code.
  - Executed repeatedly, often on check-in.
- Exploratory/Acceptance Testing often human-driven
  - Humans interact with app.
  - Based on scenarios, without pre-planned input.
  - Some tool support, but not often repeated exactly.

# Exploratory Testing

- Tester write down ideas to give direction, then create tests “live”.
  - Tester chooses next action based on results seen.
- 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

- **Fault Information:** Describe each fault. File a bug report, include tracker ID.
- **Issues Information:** If an issue prevents or complicates testing, describe it.
  - Include **data files** (screenshots, recordings, files).
- **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.
- Track time spent testing, number of faults reported, time spent on set-up, time spent on testing, time spent analyzing issues, features covered.
- Allows time management and process observability.

# Tips for Exploratory Testing

- Divide the application into modules or features, then try to further divide.
- 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

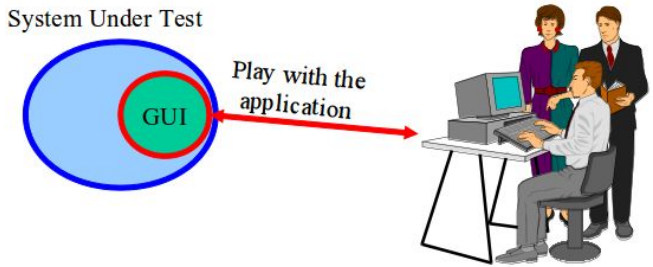
- Test all input fields.
- Check for all possible error messages.
- Test all negative scenarios.
  - Invalid input, mistakes in usage.
- Check the GUI against standards.
- Check the integration of the application with other external applications.
- Check for complex business logic.
- Try to do the ethical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

# Pair-Based Exploratory Testing

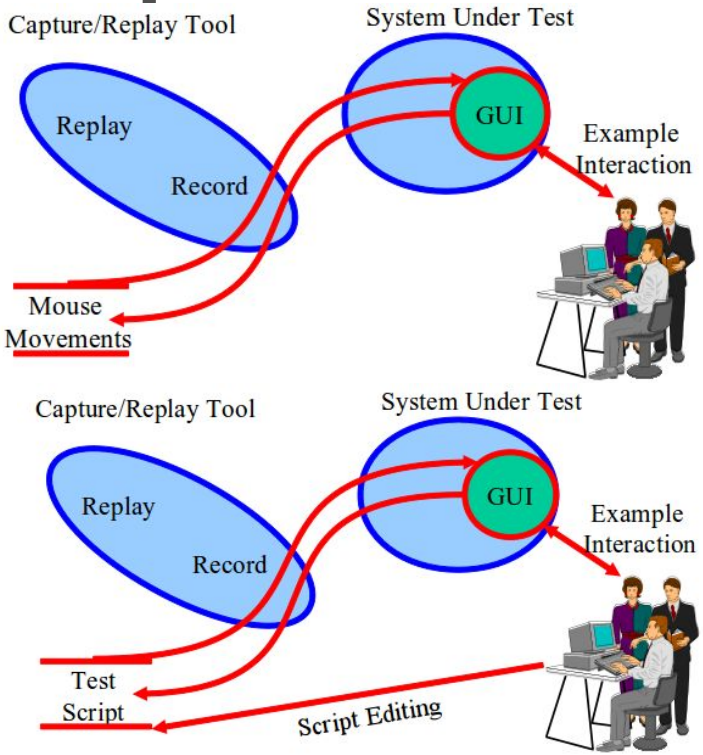
- Two people share a computer and test together.
  - One person uses the computer, the other suggests actions and takes notes.
  - Can be used to train new developers or testers.
- Benefits of pair testing:
  - Increases focus.
  - Leads to more constructive ideas.
  - Avoids biased input selection.

# 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

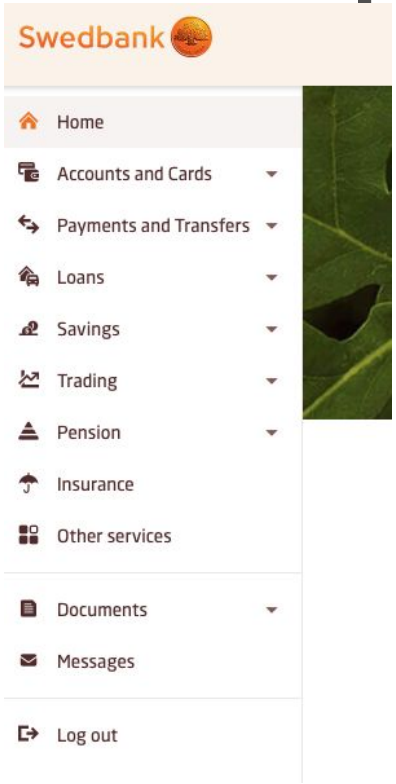



- Record input during exploratory testing.
  - The “Capture”
- Capture can be replayed to reproduce outcomes.
- Capture scripts can be extended and altered to form new test cases.

# Automating Exploratory Testing

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

# Example System




- Banking Webapp
- **How would you perform exploratory testing?**
  - Scenarios you would try?
  - Features you would focus on?

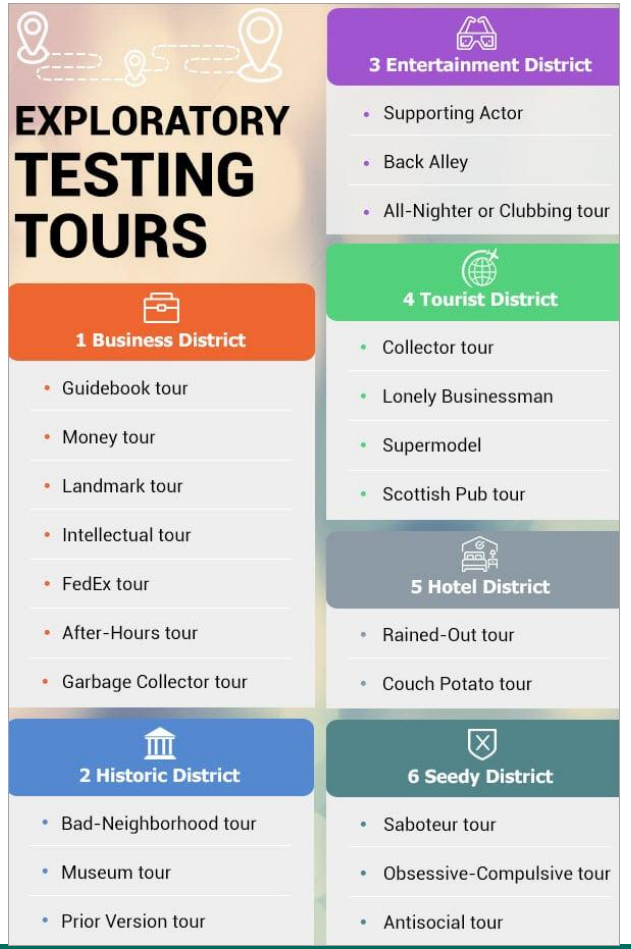
# Tours in Exploratory Testing

# 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.
- A **tour** is a plan for exploratory testing.
  - Includes a set of objectives, based on visiting different “districts”, to focus on during testing.
  - Should take less than four hours.

# Exploratory Tours

- Features split into “districts” based on type and how we test.
  - Business = core functionality
  - Seedy = security aspects
- “Tours” related to each district.
  - Each prescribes a way of exploring the software.



# Business District

- Most important features.
  - The functionality that will get users to buy software.
- Tours focus on features that are used most often.
  - **Guidebook Tour:** Focuses on common user journeys, covered in user manuals and tutorials.
  - **Fed-Ex Tour:** Focuses on how data is passed and transformed between these features.

# Guidebook Tour


- Cities advertise top attractions, and ensure they are clean and safe.
- Software offers user manuals and tutorials, illustrating step-by-step use of features.
  - Follow tutorials and execute each step.
  - Tests both functionality and accuracy of tutorials.
  - If software and tutorial do not match, report an issue.

# Guidebook Variants

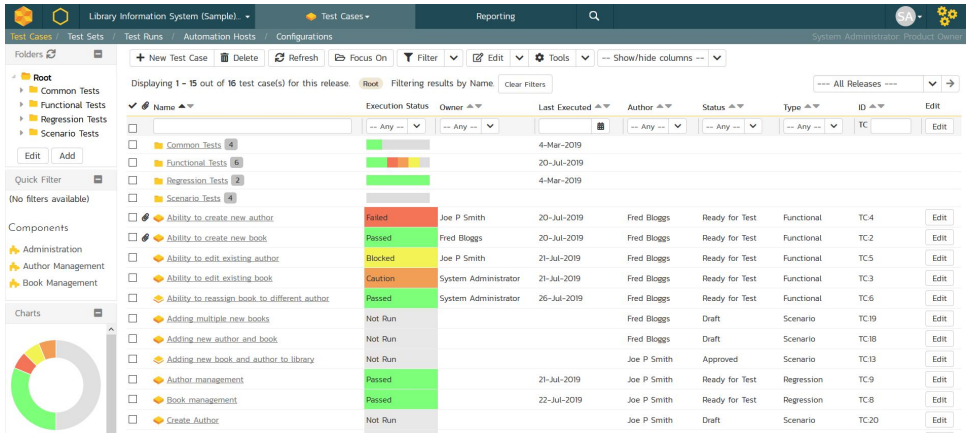
- “Blogger’s Tour”
  - Follow guides and scenarios from StackOverflow, blogs, books, other tutorials.
- “Pundit’s Tour”
  - Create tests based on complaints.
  - Try to reproduce their issues.
- “Competitor’s Tour”
  - Perform tour on competing products and their guides.
  - Identify potential improvements to your system.

# Fed-Ex Tour


- When a package is sent, it is handled by many people and passes through many locations.
  - In software, data is passed, transformed, and passed again before output appears.
- Examine how data is manipulated.
  - Validate data after operations.
  - Look at serialization/deserialization.
  - (ex: how does shopping site handle mailing addresses?)

# Fed-Ex Tour Example

- **Test Case Management System**
  - Client app pulls “work items” from a server and displays it in GUI for manipulation.
    - Test cases, bug reports
  - Relies on server connection for almost all functionality.
  - Many clients can modify same work items concurrently.



# Fed-Ex Tour Example

- **Test Case Management System**
- Must keep data items in sync between clients.
  - Bug 1: Modify name of test case, go back to view the plan. Must manually refresh to see the updated name.
  - Bug 2: Modifying the name of a test plan while a second client had it open would crash the app.
  - Bug 3: If a test plan is linked to a deleted CI build, the app will crash when the plan is opened.

# Historic District

- Historic districts contain important old buildings.
- In software, these are older features still in use.
- Tours verify that they still work and are fault-free.
  - **Bad Neighborhood Tour:** Ensure that faulty code now works, and that fixes did not introduce new faults.
  - **Museum Tour:** Ensure that unchanged code still works as intended.

# Bad Neighborhood Tour

- Complex features may have had many faults fixed over time.
- Focus on those features and ensure that:
  - Reported faults have actually been fixed.
  - New faults have not been introduced or uncovered.
- Also check related features for introduced faults.

# Museum Tour

- Older features may not have been modified or retested recently.
- Verify that old code still works in the current system.
  - Check modification dates in repository, and ensure oldest elements are retested.
  - Such elements often lack tests, are hard to modify, not tested up to current standards.

# Let’s take a break.

# 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: Making document look nice.
- Tours visit supporting features and ensures they are properly intertwined with core features.
  - **Supporting Actor:** Features on-screen with core features
  - **All-Nighter Tour:** Run the software for a long time.

# Supporting Actor Tour

- Many features might be linked to a core feature.
  - When we search for a product (core feature), we see “reviews” and “similar items” (non-core features).
- Focus testing on features that share the screen with core features.
  - Will be used often.
  - Make sure they can be accessed from the core feature.

# Tourist District

- Captures the experience of being a tourist - visit functions quickly and avoid deep inspection of individual features.
- **Souvenir Tour:** Run quick tests on functions, examine actions and identify gaps, plan round 2.
- **Supermodel Tour**: Test the GUI thoroughly, look for GUI errors, inconsistencies, usability errors.

# Supermodel Tour

- Ignore the functionality and focus on the GUI.
- As you try different functions:
  - Does GUI render properly and quickly?
  - Are transitions clean?
  - Are colors and styles used consistently?
  - Is GUI usable and accessible by those with dyslexia or colorblindness?

# Supermodel Tour Example

- **Dynamics AX Client**
  - Resource planning system acquired by Microsoft.
  - Shift from APIs to heavy GUI development.
  - Led to take-up of exploratory testing.
    - Found MANY bugs missed by API tests.
    - Many new scenarios and interactions not considered before.
    - Testers learned that they knew very little about their own app.
    - Now: exploratory testing before new features merged.

# Supermodel Tour Example

- Actions that exposed **DynamicAX** issues:
  - Modify OS settings (brightness/contrast/resolution) and verify the elements display properly.
  - Access remotely and look for flickering or bad rendering.
  - Run with multiple monitors.
  - Combine with Supporting Actor Tour:
    - Open pop-up, but look for GUI issues around pop-up.
- Appearance faults often have major impact on user perception of the program.

# Supermodel Tour Example

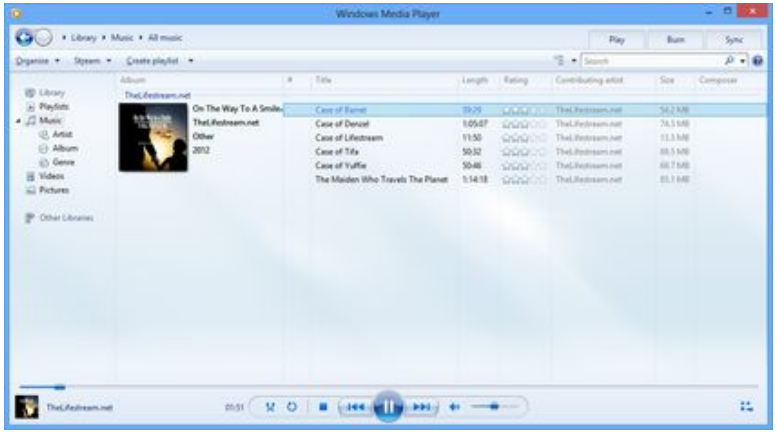
- **Windows Phone**
  - Mobile OS
  - Always connected, supports mobile, bluetooth, WiFi
  - Must consider memory, battery life, CPU speed, bandwidth.
  - Anyone can release apps that can cause potential issues on a device.

# Supermodel Tour Example

- **Windows Phone**
  - Set to an uncommon screen resolution.
    - Navigated to different calendar views.
    - When selecting a month, the month “view” was centered when it should have been top-justified.
    - Missing flag for screen resolution in this view.
  - Usability of Maps application.
    - Device knows current location, but does not use it as default when “Location A” field left blank.
    - Not a bug, but fixing would improve user experience.

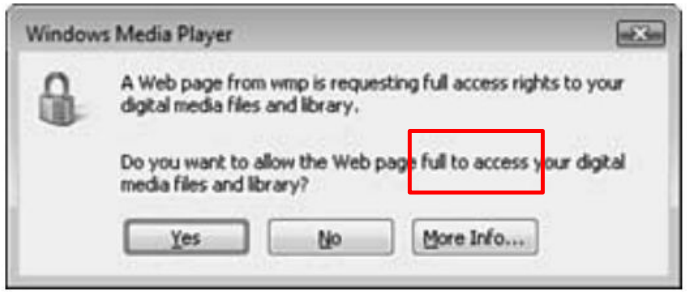
# Supermodel Tour Example

- **Windows Media Player**
  - Media player.
  - Sync, burn, rip, play many media types.
  - UI-centric application.
    - Inputs are text, check boxes, option icons, disc icons.
    - Output is audio, video, dialog boxes.



# Supermodel Tour Example

- **Windows Media Player**
  - Supermodel tour gave most rapid results to testers.
  - Many typographical mistakes found early in development.
    - Look at text and read slowly.
    - (count to two before going to the next word)
    - Not *serious*, but can harm your reputation.



# Hotel District


- Return to hotel to take a break.
- Focuses on secondary and supporting usage scenarios.
  - Software “at rest” can be very busy.
  - **Rained Out Tour:** Cancel running operations and see if problems are caused.
  - **Couch Potato Tour:** Leave fields blank and use default values to assess ability to process partial information.

# Rained-Out Tour

- Look for operations that can be cancelled.
  - Cancel midway through, 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 click back button or close entirely.

# Rained-Out Tour Example

- **DynamicsAX**
- Change the state of the software before cancelling.
  - Opened a pop-up within a form, then closed the form while pop-up was open.
    - App crashed because pop-up was still open.
  - After opening “User Setup” form, they left it open and switched to a different module.
    - Crash when they clicked Setup form’s cancel button.

# Rained-Out Tour Example

- **DynamicsAX**
- Reattempt scenario after cancelling.
  - New feature ensures that creates/updates/deletes for joined data occur within a single operation.
  - Cancel changes by clicking “Restore” button on toolbar.
  - Changes discarded and replaced by values in database.
  - Reattempted to update same record, leading to crash.

# Rained-Out Tour Example

- **Test Case Management System**
- Interrupted server requests and refresh actions can lead to issues.
  - Bug 1: Canceled initial connection to project. No longer able to manually connect to it.
  - Bug 2: Switching test suites during loading does not stop loading of the original suite.
  - Bug 3: Clicking refresh button several times causes slowdown, as each refresh is handled (not just the latest).

# Rained-Out Tour Example

- **Windows Phone**
  - Search for contact
    - Loaded > 4000 contacts. While searching, they changed the search string.
    - Changed filter clashed with original filter, incorrect results.
  - Bluetooth Connection
    - If focus shifts while connecting to device, can try to connect again
    - Multiple connection requests will be sent.
    - Device functions once connected, but multiple failure notices come back.

# Couch Potato Tour

- Tester does least interaction possible.
  - Leave default values in place, leave input fields blank, try to move forward without offering much data.
- Ensures software must execute code for processing blank or partial information and 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:** Directly attack software via malformed input or resource manipulation.
  - **Antisocial Tour:** Try unlikely input or perform actions in the wrong order.
    - (add 10000 songs, try to play empty playlist, order 10000000 pairs of shoes)

# 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.

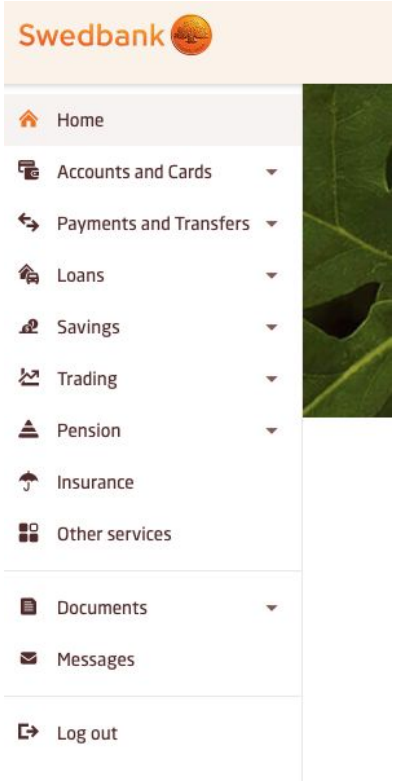
# Saboteur Tour Example

- **Test Case Management System**
- Change or remove necessary resources.
  - Bug 1: System crashes if connection to data server is closed at different points.
  - Bug 2: System crashes, restarts, crashes again, etc. if the config file is corrupted.
  - Bug 3: System crashes if config file is too large.
    - (also try making it read-only, changing file type, deleting)

# Saboteur Tour Example

- **Windows Phone**
  - Contact lists linked to call history, speed dial, texts, etc.
    - Delete linking database between contacts and speed dial.
    - Contacts still on device, so phone thinks data is synced.
    - However, speed dial is empty.
  - Airplane mode may not be accounted for.
    - IM client loses connection when airplane mode turned on.
    - However, does not realize it was disconnected.
    - User can still attempt to use client when nothing will work.

# Revisiting the Example System




- Banking Webapp
- **How would you perform exploratory testing?**
  - Scenarios you would try?
  - Features you would focus on?
  - Particular tours?

# 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, iterative fast rounds of exploration.
  - Hotel District: Focuses on supporting functionality
  - Seedy District: Attacks and misuse of software

# Next Time

- Unit Testing and Test Automation
  - Pezze and Young, Ch 17
- **Before Friday's Exercise Session:**
  - **Install an IDE (IntelliJ, Eclipse) and ensure that JUnit is installed and usable.**
- Assignment 1 due February 13
  - Questions?
  - (We will post Assignment 2 soon)


UNIVERSITY OF
GOTHENBURG


CHALMERS
UNIVERSITY OF TECHNOLOGY