Lecture 12: Testing of Feature Interactions

Gregory Gay
TDA594 - December 10, 2020
Creating System-Level Tests

- **Identify Independently Testable Functionality**
  - Identify functionality that can be tested in (relative) isolation.

- **Identify Choices**
  - Identify the choices you control when testing.

- **Identify Representative Input Values**
  - Identify values for each choice that lead to different function outcomes.

- **Generate Test Case Specifications**
  - Identify abstract test cases based on choice combinations.

- **Generate Test Cases**
  - Identify concrete input/expected output pairs.
Test Specifications

• May end up with thousands of test specifications.
• Which do you turn into concrete test cases?
  • Filter impossible or redundant combinations of values.
  • Try to capture all (2-way, 3-way, N-way) feature interactions.
Feature Interactions

• Features are expected to interact.
  • Usually this is planned!
  • Sometimes unplanned interactions break the system.
  • We should select tests that thoroughly test feature interactions.
Today’s Goals

• Understand how feature interactions can create software faults.

• Examine how to select system-level tests to increase likelihood of detecting feature interaction faults.
  • Category-Partition Method
  • Combinatorial Interaction Testing
Feature Interactions

• **Feature interactions** result from combined behaviors of individual features.
  - **Inadvertent feature interactions** cause unexpected behavior (ex. incorrect output, changes in timing)

• Feature Interaction Problem
  - Detect, manage, resolve inadvertent feature interactions.
Fire and Flood Control

- FireControl activates sprinklers when fire detected.
- FloodControl cuts water supply when water detected on floor.
- Interaction means building burns down.
WordPress Plug-Ins

- Weather and emoji plug-ins tested independently.
- Their interaction results in unexpected behavior.
Feature Interactions

Unit test vs. Integration test
N-Way Interactions

- Interactions between two features are called 2-way interactions.
- If N features interact, this is an N-way interaction.
  - A, B, C have three 2-way interactions, one 3-way interaction.
N-Way Interactions

- Features: Locking, Logging, Undo.
- Nested #ifdef indicate N-way interactions
  - 2-way: 3
  - 3-way: 1

```java
class Stack {
    boolean push(Object o) {
        #ifdef LOCKING
        Lock lock = lock();
        if(lock == null) {
            #ifdef LOGGING
            log("lock failed for: "+o);
            #endif
            return false;
        }
        #endif
        #ifdef UNDO
        rememberValue();
        #endif
        elementData[size++] = o;
        /*****/
    }
    #ifdef LOGGING
    void log(String msg) { /*****/ }
    #endif

    #ifdef UNDO
    boolean undo() {
        #ifdef LOCKING
        Lock lock = lock();
        if(lock == null) {
            #ifdef LOGGING
            log("undo-lock failed");
            #endif
            return false;
        }
        #endif
        restoreValue();
        /*****/
        #ifdef LOGGING
        log("undone.");
        #endif
    }
    #endif

    void rememberValue() { /*****/ }
    void restoreValue() { /*****/ }
    #endif
}
```
Category-Partition Method
Category-Partition Method

Generates test specifications from requirements.

- **Choices, representative values, and constraints.**
  - **Choices:** What you can control when testing.
  - **Representative Values:** Logical options for each choice.
  - **Constraints:** Limit certain combinations of values.

- Generate a list of test specifications to cover.
  - Apply more constraints to further limit set.
Identify Choices

• Identify high-level functions and their parameters.
  • *Direct input, environmental parameters (i.e., databases), and options for variation points.*

• Identify characteristics of each parameter.
  • What can we control when we test? *(the choices)*
  • What are the possible values for these choices?

• **Choices** are also called *categories* if you look up category-partition method.
Example: Computer Configurations

• Web shop that sells custom computers.
• A configuration is a set of options for a model.
  • Some combinations are invalid (i.e., display port monitor with HDMI video output).
• Function: `checkConfiguration(model, configuration)`
  • What are the parameters?
  • What are the choices to be made for each parameter?
Example: Computer Configuration

- **Model**: Identifies a product and determines constraints on available components. Identified by a model number. Characterized by a set of slots. Slots may be required (must be filled) or optional (may be left empty).

- **Configuration**: Set of <slot, component> pairs. Must correspond to the required and optional slots of the model. Available components and a default for each slot are determined by the model. Slots may be empty (may be default for optional slots). Components can be compatible or incompatible with a model or with each other.
Example: Configuration Choices

- **Parameter: Model**
  - Model number
  - Number of required slots (must have a component)
  - Number of optional slots (component or empty)
- **Parameter: Configuration**
  - Selected configuration valid for model?
  - Number [required/optional] slots with non-empty selections.
  - Selected components for [required/optional] slots OK?
- **Parameter: Product Database**
  - Number of models in database
  - Number of components in database
Identify Representative Values

- Many values can be selected for each choice.
- Partition each choice into classes of values.
  - Consider all outcomes of function.
  - Consider logical ranges or groupings.
- Test specification is a selection of values for all choices.
  - Concrete test case fills values for each abstract selection.
# Values for Each Choice

**Parameter: Model**

- **Choice: Model number**
  - malformed
  - not in database
  - valid
- **Choice: Number of required slots**
  - 0
  - 1
  - many
- **Choice: Number of optional slots**
  - 0
  - 1
  - many

**Parameter: Product Database**

- **Choice: Number of models in database**
  - 0
  - 1
  - many
- **Number of components in database**
  - 0
  - 1
  - many

**Parameter: Configuration**

- **Choice: Configuration Matches Model**
  - complete correspondence
  - omitted slots in configuration
  - extra slots in configuration
  - mismatched number of required and optional slots
- **Choice: Number of empty required slots that are empty**
  - all required slots filled
  - some required slots empty
  - all required slots empty
- **Choice: Number of optional slots that are empty**
  - all optional slots filled
  - some optional slots empty
  - all optional slots empty
- **Choice: Selected components for required slots**
  - all valid
  - some kept at default
  - >= 1 incompatible with slot
  - >= 1 incompatible with another component
  - >= 1 not in database
- **Choice: Selected components for optional slots**
  - all valid
  - some kept at default
  - >= 1 incompatible with slot
  - >= 1 incompatible with another component
  - >= 1 not in database
Generate Test Case Specifications

• Test specification = selection of values for choices.

• **Constraints** limit number of specifications.
  • Eliminate impossible pairings.
  • Remove unnecessary options.
  • Choose a subset to turn into concrete tests.

1944 tests (all combinations) → 678 Tests → 40 Tests!
- Seven choices with three values, one with four values, two with five values.
  - $3^7 \times 5^2 \times 4 = 218700$ test specifications
- Not all combinations correspond to reasonable specifications.

Parameter: Model
- **Choice: Model number**
  - malformed
  - not in database
  - valid
- **Choice: Number of required slots**
  - 0
  - 1
  - many
- **Choice: Number of optional slots**
  - 0
  - 1
  - many

Parameter: Product Database
- **Choice: Number of models in database**
  - 0
  - 1
  - many
- **Number of components in database**
  - 0
  - 1
  - many

Parameter: Configuration
- **Choice: Configuration Matches Model**
  - complete correspondence
  - omitted slots in configuration
  - extra slots in configuration
  - mismatched number of required and optional slots
- **Choice: Number of empty required slots that are empty**
  - all required slots filled
  - some required slots empty
  - all required slots empty
- **Choice: Number of optional slots that are empty**
  - all optional slots filled
  - some optional slots empty
  - all optional slots empty
- **Choice: Selected components for required slots**
  - all valid
  - some kept at default
  - $>= 1$ incompatible with slot
  - $>= 1$ incompatible with another component
  - $>= 1$ not in database
- **Choice: Selected components for optional slots**
  - all valid
  - some kept at default
  - $>= 1$ incompatible with slot
  - $>= 1$ incompatible with another component
  - $>= 1$ not in database

$3 \times 5 \times 4 = 218700$ test specifications
Constraints Between Values

• IF-CONSTRAINT
  • This value only needs to be used under certain conditions (if X is true, use value Y)

• ERROR
  • Value causes error regardless of values of other choices.

• SINGLE
  • Only a single test with this value is needed.
  • Corner cases that should give “good” outcome.
Example - Substring

\texttt{substr(string str, int index)}

**Choice: Str length**

- \texttt{length = 0} \textbf{property zeroLen, TRUE if length = 0}
- \texttt{length = 1}
- \texttt{length >= 2}

**Choice: Str contents**

- contains letters and numbers \textbf{if \!zeroLen}
- contains special characters \textbf{if \!zeroLen} \textbf{SINGLE}
- empty \textbf{if zeroLen}

**Choice: index**

- value < 0 \textbf{ERROR}
- value = 0
- value = 1
- value > 1
Example - Configuration Constraints

Parameter: Product Database

- Choice: Number of models in database
  - 0 [error]
  - 1 [single]
  - many

- Choice: Number of components in database
  - 0 [error]
  - 1 [single]
  - many

Parameter: Model

- Choice: Model number
  - malformed [error]
  - not in database
  - valid

- Choice: Number of required slots
  - 0 [single]
  - 1
  - many [property RSMANY]

- Choice: Number of optional slots
  - 0 [single]
  - 1
  - many [property OSMANY]

8 (error cases) + 6 (single cases) + (1^7\times2^1\times3^2) (RSMANY = true/OSMANY = true) + (1^5\times2^3\times3^2) (false/true) + (1^5\times2^3\times3^2) (true/false) + (1^3\times2^5\times3^2) (false/false) = 464 test specifications

Parameter: Configuration

- Choice: Configuration Matches Model
  - complete correspondence
  - omitted slots in configuration [error]
  - extra slots in configuration [error]
  - mismatched number of required and optional slots [error]

- Choice: Number of empty required slots that are empty
  - all required slots filled
  - some required slots empty [if RSMANY]
  - all required slots empty

- Choice: Number of optional slots that are empty
  - all optional slots filled
  - some optional slots empty [if OSMANY]
  - all optional slots empty

- Choice: Selected components for required slots
  - all valid
  - some kept at default [single]
  - >= 1 incompatible with slot
  - >= 1 incompatible with another component
  - >= 1 not in database [error]

- Choice: Selected components for optional slots
  - all valid
  - some kept at default [single]
  - >= 1 incompatible with slot
  - >= 1 incompatible with another component
  - >= 1 not in database [error]
Activity - find service

find(pattern, file)

- Finds instances of a pattern in a file
  - find("john", myFile)
    - Finds all instances of john in the file
  - find("john smith", myFile)
    - Finds all instances of john smith in the file
  - find("\"john\" smith", myFile)
    - Finds all instances of "john" smith in the file
Activity - find Service

- Parameters: pattern, file
- What can we vary for each?
  - What can we control about the pattern? Or the file?
- What values can we choose for each choice?
  - File name:
    - File exists with that name
    - File does not exist with that name
- What constraints can we apply between choice values? (if, single, error)

https://bit.ly/3gmRI7I
Example - find Service

- Pattern size:
  - Empty
  - single character
  - many characters
  - longer than any line in the file
- Quoting:
  - pattern has no quotes
  - pattern has proper quotes
  - pattern has improper quotes (only one “)
- Embedded spaces:
  - No spaces
  - One space
  - Several spaces

\[(2^2 \times 3^3 \times 4^1) = 108 \text{ test specifications}\]

- File name:
  - Existing file name
  - no file with this name
- Number of occurrence of pattern in file:
  - None
  - exactly one
  - more than one
- Pattern occurrences on any single line:
  - One
  - more than one
ERROR and SINGLE Constraints

- Pattern size:
  - Empty
  - single character
  - many character
  - longer than any line in the file
  - Quoting:
    - pattern has no quotes
    - pattern has proper quotes
  - pattern has improper quotes (only one “"
- Embedded spaces:
  - No spaces
  - One space
  - Several spaces

4 (error) + 2 (single) + (1^2 * 2^3 * 3^1) = 30

- File name:
  - Existing file name
  - no file with this name [error]
- Number of occurrence of pattern in file:
  - None
  - exactly one [single]
  - more than one
- Pattern occurrences on target line:
  - One
  - more than one [single]
IF Constraints

- Pattern size:
  - Empty
  - single character
  - many character
  - longer than any line in the file

- Quoting:
  - pattern has no quotes
  - pattern has proper quotes
  - pattern has improper quotes (only one “”)

- Embedded spaces:
  - No spaces
  - One space
  - Several spaces

- File name:
  - Existing file name
  - no file with this name

- Number of occurrence of pattern in file:
  - None
  - exactly one
  - more than one

- Pattern occurrences on target line:
  - One
  - more than one

4 (error) + 2 (single) + (1^3*2^3) (quoted = true) + (1^4*2^2) (quoted = false) = 18
Let’s take a break.
Combinatorial Interaction Testing
Limiting Num. of Test Specifications

- Full set = 432 specifications
- No natural IF, SINGLE, ERROR constraints for these features.
- What is important to cover?

<table>
<thead>
<tr>
<th>Bandwidth Mode</th>
<th>Language</th>
<th>Fonts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desktop Site</td>
<td>English</td>
<td>Standard</td>
</tr>
<tr>
<td>Mobile Site</td>
<td>French</td>
<td>Open-Source</td>
</tr>
<tr>
<td>Text Only</td>
<td>German</td>
<td>Minimal</td>
</tr>
<tr>
<td></td>
<td>Swedish</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Advertising</th>
<th>Screen Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Advertising</td>
<td>Phone</td>
</tr>
<tr>
<td>Targeted Advertising</td>
<td>Tablet</td>
</tr>
<tr>
<td>General Advertising</td>
<td>Full Size</td>
</tr>
<tr>
<td>Minimal Advertising</td>
<td></td>
</tr>
</tbody>
</table>
Pairwise Interaction Testing

• Cover all k-way interactions (k < N).
  • Typically 2-way (pairwise) or 3-way.
• Set of all combinations grows exponentially.
• Set of pairwise combinations grows logarithmically.
  • (last slide) 432 combinations.
  • Possible to cover all pairs in 16 tests.
Example - Paragraph Effects

Paragraph spaces has two values: selected and unselected. Mirror indents has two values: selected and unselected. And finally, line spacing has three values: single, multiple and double.

\[2 \times 2 \times 3 = 12\] combinations
Example - Paragraph Effects
Example - Paragraph Effects

• Goal of CIT is to produce **covering array**.
  • Set of configurations that covers all K-way combinations.
    • (2-way here).
  • Cover in 6 test specifications.
### Example - Website Display

<table>
<thead>
<tr>
<th>Bandwidth Mode</th>
<th>Fonts</th>
<th>Screen Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desktop Site</td>
<td>Standard</td>
<td>Phone</td>
</tr>
<tr>
<td>Desktop Site</td>
<td>Open-Source</td>
<td>Tablet</td>
</tr>
<tr>
<td>Desktop Site</td>
<td>Minimal</td>
<td>Full Size</td>
</tr>
<tr>
<td>Mobile Site</td>
<td>Standard</td>
<td>Tablet</td>
</tr>
<tr>
<td>Mobile Site</td>
<td>Open-Source</td>
<td>Full Size</td>
</tr>
<tr>
<td>Mobile Site</td>
<td>Minimal</td>
<td>Phone</td>
</tr>
<tr>
<td>Text Only</td>
<td>Standard</td>
<td>Full Size</td>
</tr>
<tr>
<td>Text Only</td>
<td>Open-Source</td>
<td>Phone</td>
</tr>
<tr>
<td>Text Only</td>
<td>Minimal</td>
<td>Tablet</td>
</tr>
</tbody>
</table>

- Cover all combinations for two variables.
- Add a third, account for all combinations of pairs of values.
  - Each test specification can cover up to three pairs.
## Example - Website Display

<table>
<thead>
<tr>
<th>Bandwidth Mode</th>
<th>Language</th>
<th>Fonts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desktop Site</td>
<td>English</td>
<td>Standard</td>
</tr>
<tr>
<td>Mobile Site</td>
<td>French</td>
<td>Open-Source</td>
</tr>
<tr>
<td>Text Only</td>
<td>German</td>
<td>Minimal</td>
</tr>
<tr>
<td></td>
<td>Swedish</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Advertising</th>
<th>Screen Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Advertising</td>
<td>Phone</td>
</tr>
<tr>
<td>Targeted Advertising</td>
<td>Tablet</td>
</tr>
<tr>
<td>General Advertising</td>
<td>Full Size</td>
</tr>
<tr>
<td>Minimal Advertising</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Language</th>
<th>Advertising</th>
<th>Bandwidth Mode</th>
<th>Fonts</th>
<th>Screen Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>English</td>
<td>No Advertising</td>
<td>Desktop Site</td>
<td>Standard</td>
<td>Phone</td>
</tr>
<tr>
<td>English</td>
<td>Targeted Advertising</td>
<td>Mobile Site</td>
<td>Open-Source</td>
<td>Tablet</td>
</tr>
<tr>
<td>English</td>
<td>General Advertising</td>
<td>Text Only</td>
<td>Minimal</td>
<td>Full Size</td>
</tr>
<tr>
<td>English</td>
<td>Minimal Advertising</td>
<td>Mobile Site</td>
<td>Minimal</td>
<td>Phone</td>
</tr>
<tr>
<td>French</td>
<td>No Advertising</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>French</td>
<td>Targeted Advertising</td>
<td>Desktop Site</td>
<td>Minimal</td>
<td>Full Size</td>
</tr>
<tr>
<td>French</td>
<td>General Advertising</td>
<td>Mobile Site</td>
<td>Standard</td>
<td>Tablet</td>
</tr>
<tr>
<td>French</td>
<td>Minimal Advertising</td>
<td>Text Only</td>
<td>Open-Source</td>
<td>Phone</td>
</tr>
<tr>
<td>German</td>
<td>No Advertising</td>
<td>Text Only</td>
<td>Minimal</td>
<td>Tablet</td>
</tr>
<tr>
<td>German</td>
<td>Targeted Advertising</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>German</td>
<td>General Advertising</td>
<td>Desktop Site</td>
<td>Open-Source</td>
<td>Phone</td>
</tr>
<tr>
<td>German</td>
<td>Minimal Advertising</td>
<td>Mobile Site</td>
<td>Standard</td>
<td>Full Size</td>
</tr>
<tr>
<td>Swedish</td>
<td>No Advertising</td>
<td>Mobile Site</td>
<td>Open-Source</td>
<td>Full Size</td>
</tr>
<tr>
<td>Swedish</td>
<td>Targeted Advertising</td>
<td>Text Only</td>
<td>Standard</td>
<td>Phone</td>
</tr>
<tr>
<td>Swedish</td>
<td>General Advertising</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Swedish</td>
<td>Minimal Advertising</td>
<td>Desktop Site</td>
<td>Minimal</td>
<td>Tablet</td>
</tr>
</tbody>
</table>
Constraints

- Remove all ERROR/SINGLE cases before CIT.
  - Error output, one-time corner cases
- Constraints on value combinations specified:
  - OMIT(Text-Only, *, *, Full Size, *)
  - OMIT(*, *, *, Full Size, Minimal)
- Further reduces number of test specifications.
CIT Tools

• Pairwise Independent Combinatorial Testing (Microsoft): https://github.com/microsoft/pict
• .. Many more: http://www.pairwise.org/tools.asp
Activity - Browser Configuration

<table>
<thead>
<tr>
<th>Allow Content to Load</th>
<th>Notify About Pop-Ups</th>
<th>Allow Cookies</th>
<th>Warn About Add-Ons</th>
<th>Warn About Attack Sites</th>
<th>Warn About Forgeries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allow</td>
<td>Yes</td>
<td>Allow</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Restrict</td>
<td>No</td>
<td>Restrict</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Block</td>
<td>Block</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Full set of test specifications = 144
- Create set covering all pairwise value combinations.
  - Hint: Start with two variables with most values. Add one variable at a time.
# Activity Solution

<table>
<thead>
<tr>
<th>Allow Content</th>
<th>Allow Cookies</th>
<th>Pop-Ups</th>
<th>Add-Ons</th>
<th>Attacks</th>
<th>Forgeries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allow</td>
<td>Allow</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Allow</td>
<td>Restrict</td>
<td>No</td>
<td>No</td>
<td>-</td>
<td>No</td>
</tr>
<tr>
<td>Allow</td>
<td>Block</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Restrict</td>
<td>Allow</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Restrict</td>
<td>Restrict</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
<td>Yes</td>
</tr>
<tr>
<td>Restrict</td>
<td>Block</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Block</td>
<td>Allow</td>
<td>No</td>
<td>-</td>
<td>-</td>
<td>Yes</td>
</tr>
<tr>
<td>Block</td>
<td>Restrict</td>
<td>-</td>
<td>Yes</td>
<td>No</td>
<td>-</td>
</tr>
<tr>
<td>Block</td>
<td>Block</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
We Have Learned

• Process for deriving system-level tests often results in too many possible cases.

• Two methods that identify important interactions:
  • **Category-Partition Method:** Use constraints to eliminate unnecessary tests.
  • **Combinatorial Interaction Testing:** Identify important pairs of input values.
Next Time

• Automated test case generation using search-based techniques.

• Assignment 4 is out.
  • Due December 20. Questions?