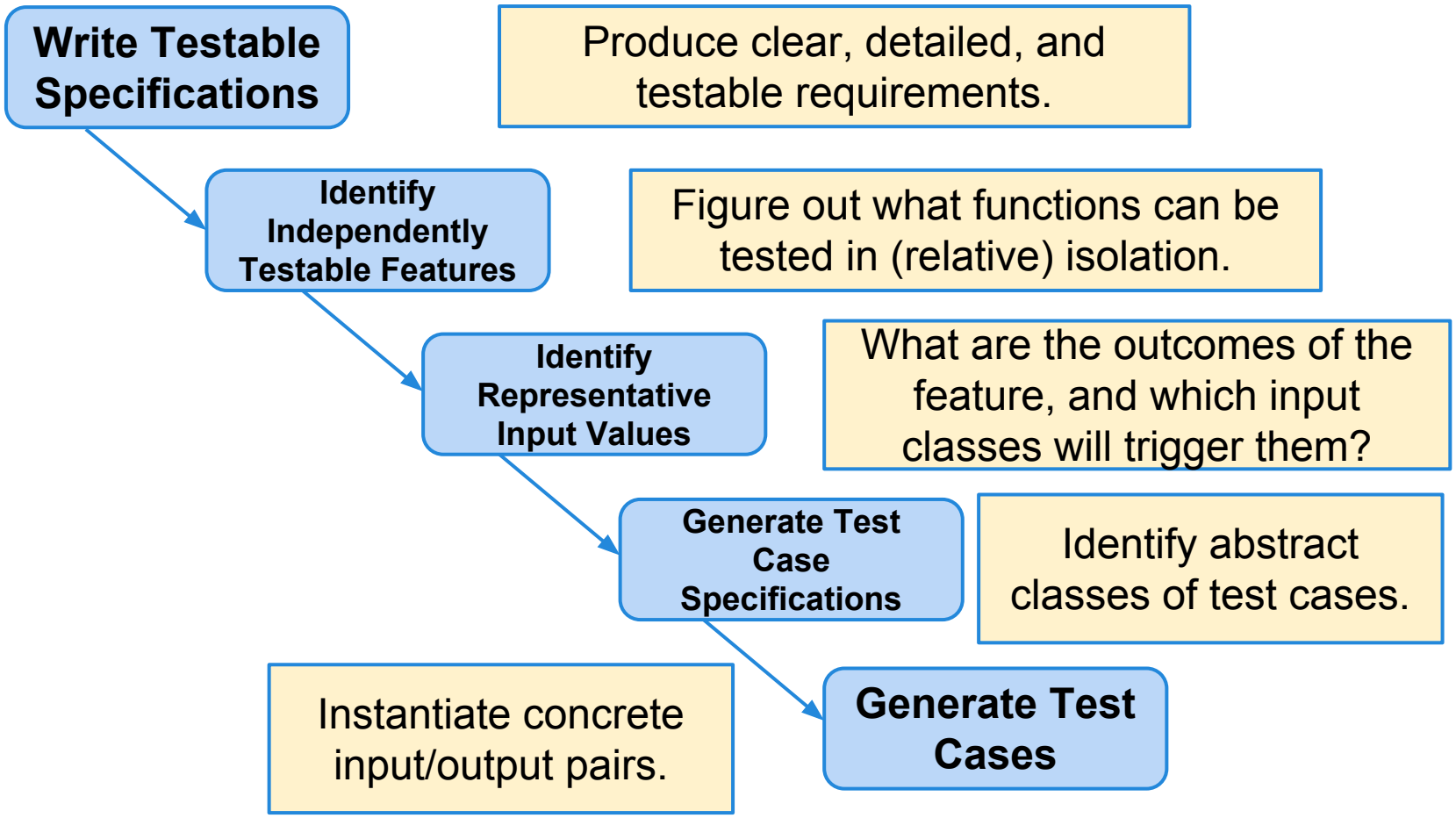
Combinatorial Testing

CSCE 747 - Lecture 5 - 02/01/2018

Creating Requirements-Based Tests



Activity - Functional Testing

You are asked to develop a simple C++ container class SetOfE containing elements of type E with the following methods:

- `void insert(E e)`
- `Bool find(E e)`
- `void delete(E e)`

Using domain partitioning, develop functional test cases for the methods. You can define your test cases as input/output pairs.

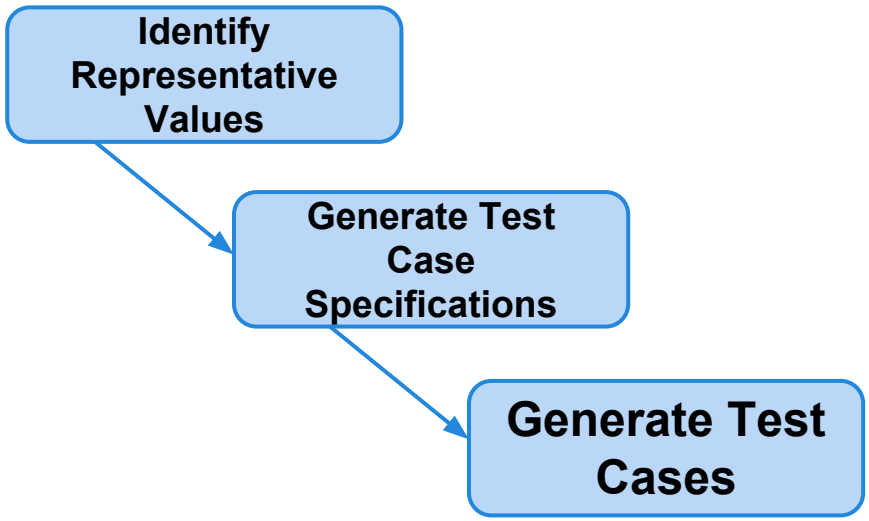
For example, to test `insert(E e)`, one test case could be:
Input: Empty Container/any e **Expected output:** e in Container.

Question 6 (2) - Solution

Insert	*Empty/ any e*	*e in container*
	E with one element / any e	*e in container*
	E with multiple elements / any e	*e in container*
	Very large E/ any e	*e in container*
	E containing e/ e	*Error or no change*
	Any E/ malformed e	*Error*
Exists	*E containing e/ e*	*True*
	E not containing e/ e	*False*
	Very large E containing e/ e	*True*
	E with only element e/ e	*True*
	Any E / malformed e	*Error*
	Empty / e	*False*

Delete	*E containing e/ e*	*e no longer in list*
	E not containing e/ e	*no change (or error)*
	Any E / malformed e	*error*
	Very large E containing e/ e	*e no longer in list*
	Empty / e	*no change*

Building a Test Suite



Smarter process than random testing, but still comes down to brute force:

- May still be an infeasibly high number of test specifications.
- Each specification can be transformed into MANY concrete test cases. How many should be tried?

How do we arrive at an effective, reasonably-sized test suite?

Today's Goals:

- Category-Partition Method
  - Assists in identifying test specifications, estimating the number of tests, and forming a subset that meets your budget
- Combinatorial Interaction Testing
  - Method of covering n-way combinations of parameter values with a small number of tests.

Category-Partition Method

Category-Partition Method

A method of generating test specifications from requirement specifications.

- A small number of additional steps on the process discussed last class.
- Requires identifying *categories*, *choices*, and *constraints*.
- Once identified, these can be used to automatically generate a list of test specifications to cover.

Identify Independently Testable Features and Parameter Characteristics

- Identify features and their parameters.
- Identify **characteristics** of each parameter.
  - What are the controllable attributes?
  - What are their possible values?
    - May be defined partially by other parameters and their characteristics.
    - May not correspond to variables in the code.
- The parameter characteristics are called ***categories***.

Example: Computer Configurations

- Your company sells custom computers.
- A *configuration* is a set of options for a *model* of computer.
  - Some combinations are invalid (i.e., VGA monitor with HDMI video output).
- Testing feature:
  - **`checkConfiguration(model,components)`**
  - What are the parameters?
  - Next - what are the choices to be made for each parameter?

Parameter Characteristics

- Turn to the requirements specifications.
- **Model:** A model identifies a specific product and determines a set of constraints on available components. Models are identified by a model number. Models are characterized by logical slots on a bug. Slots may be required (must be filled) or optional (may be left empty).
- **Set of Components:** A set of <slot, component> pairs, which must correspond to the required and optional slots associated with the model. A component is a choice that can be varied within a model. Available components and a default for each slot is determined by the model. The special value “empty” is allowed and may be the default for optional slots. In addition to being compatible or incompatible with a model, components may be compatible or incompatible with each other.

Categories

- **Model**
  - Model number
  - Number of required slots
  - Number of optional slots
- **Components**
  - Correspondence of selection with model slots
  - Number of required components with non-empty selections
  - Number of optional components with non-empty selections
  - Selected components for required slots
  - Selected components for optional slots
- **Product Database**
  - Number of models in database
  - Number of components in database

Identify Representative Values

- For each category, many values that can be selected for concrete test cases.
- We need to identify *classes of values*, called ***choices***, for each category.
  - A test specification is a combination of choices for all categories.
- Consider all outcomes of a feature.
- Consider boundary values.

Categories

- **Model**
  - Model number
  - Number of required slots
  - Number of optional slots
- **Components**
  - Correspondence of selection with model slots
  - Number of required components with non-empty selections
  - Number of optional components with non-empty selections
  - Selected components for required slots
  - Selected components for optional slots
- **Product Database**
  - Number of models in database
  - Number of components in database

Choices for Each Category

- **Model**
  - Model number
    - malformed
    - not in database
    - valid
  - Number of required slots
    - 0
    - 1
    - many
  - Number of optional slots
    - 0
    - 1
    - many
- **Product Database**
  - Number of models in database
    - 0
    - 1
    - many
  - Number of components in database
    - 0
    - 1
    - many
- **Components**
  - Correspondence of selection with model slots
    - omitted slots
    - extra slots
    - mismatched slots
    - complete correspondence
  - Number of required(optional) components with non-empty selections
    - 0
    - < number required (optional)
    - = number required (optional)
  - Selected components for required (optional) slots
    - some default
    - all valid
    - >= 1 incompatible with slot
    - >= 1 incompatible with another component
    - >= 1 not in database

Generate Test Case Specifications

- Test specifications are formed by combining choices for all categories.
- Number of possible combinations may be impractically large, so:
  - Eliminate impossible pairings.
  - Identify constraints that can remove unnecessary options.
  - From the remainder, choose a subset of specifications to turn into concrete tests.

Choices for Each Category

- **Model**
  - Model number
    - malformed
    - not in d
    - valid
  - Number of re
    - 0
    - 1
    - many
  - Number of op
    - 0
    - 1
    - many
- **Product Database**
  - Number of m
    - 0
    - 1
    - many
  - Number of co
    - 0
    - 1
    - many
- **Components**
  - Correspondence of selection with model slots

s
ondence
tional)
empty

ed (optional)
ed (optional)
for required

e with slot
e with another
component
    - >= 1 not in database

- Seven categories with three choices.
- Two categories with 6 choices.
- One category with 4 choices.
- Results in $3^7 \times 6^2 \times 4 = 314928$ test specifications
- However… not all combinations correspond to reasonable specifications.

Identify Constraints Among Choices

Three types of constraint:

- IF
  - This partition only needs to be considered if another property is true.
- ERROR
  - This partition should cause a problem no matter what value the other input variables have.
- SINGLE
  - Only a single test with this partition is needed.

Applying Constraints

```
substr(string str, int index)
```

Str length

length 0 — PROPERTY zeroLen

length 1

length >= 2

Str contents

contains special characters — if !zeroLen

contains lower case only — if !zeroLen

contains mixed case — if !zeroLen

Input index

value <0 — ERROR

value = 0

value = 1

value > 1

value = MAXINT — SINGLE

Applying Constraints

- **Model**
  - Model number
    - malformed **[error]**
    - not in database **[error]**
    - valid
  - Number of required slots
    - 0 **[single]**
    - 1 **[property RSNE] [single]**
    - many **[property RSNE], [property RSMANY]**
  - Number of optional slots
    - 0 **[single]**
    - 1 **[property OSNE][single]**
    - many **[property OSNE], [property OSMANY]**
- **Product Database**
  - Number of models in database
    - 0 **[error]**
    - 1 **[single]**
    - many
  - Number of components in database
    - 0 **[error]**
    - 1 **[single]**
    - many
- **Components**
  - Correspondence of selection with model slots
    - omitted slots **[error]**
    - extra slots **[error]**
    - mismatched slots **[error]**
    - complete correspondence
  - Number of required components with non-empty selections
    - 0 **[if RSNE] [error]**
    - < number required **[if RSNE] [error]**
    - = number required **[if RSMANY]**
  - Number of optional components with non-empty selections
    - 0
    - < number optional **[if OSNE]**
    - = number optional **[if OSMANY]**
  - Selected components for required (optional) slots
    - some default **[single]**
    - all valid
    - >= 1 incompatible with slot
    - >= 1 incompatible with another component
    - >= 1 not in database **[error]**

Example - Find Command

Bash command: find

`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

Example - Find Command

- Parameters: pattern, file
- What can we vary for each?
  - Our categories.
  - What can we control about the pattern? Or the file?
- What choices can we make for each category?
  - Our categories
  - **File name:**
    - Name of an existing file provided
    - File does not exist

Example - Find Comm

1944 tests if we consider all combinations.

- Pattern size:
  - Empty
  - single character
  - many character
  - longer than any line in the file
- Quoting:
  - pattern is quoted
  - not quoated
  - improperly quoated
- Embedded spaces:
  - No spaces
  - One space
  - Several spaces
- Embedded quotes:
  - no quotes
  - one quote
  - several quotes
- 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

IF Constraints

678 Tests

- Pattern size:
  - Empty
  - single character **[not empty]**
  - many character **[not empty]**
  - longer than any line in the file **[not empty]**
- Quoting:
  - pattern is quoted **[quoted] [if not empty]**
  - not quoted **[if not empty]**
  - improperly quoted **[if not empty]**
- Embedded spaces:
  - No spaces
  - One space **[if not empty and quoted]**
  - Several spaces **[if not empty and quoted]**
- Embedded quotes:
  - no quotes
  - one quote **[if not empty]**
  - several quotes **[if not empty and quoted]**
- File name:
  - Existing file name
  - no file with this name
- Number of occurrence of pattern in file:
  - None **[if not empty]**
  - exactly one **[match] [if not empty]**
  - more than one **[match] [if not empty]**
- Pattern occurrences on target line:
  - One **[if match]**
  - more than one **[if match]**

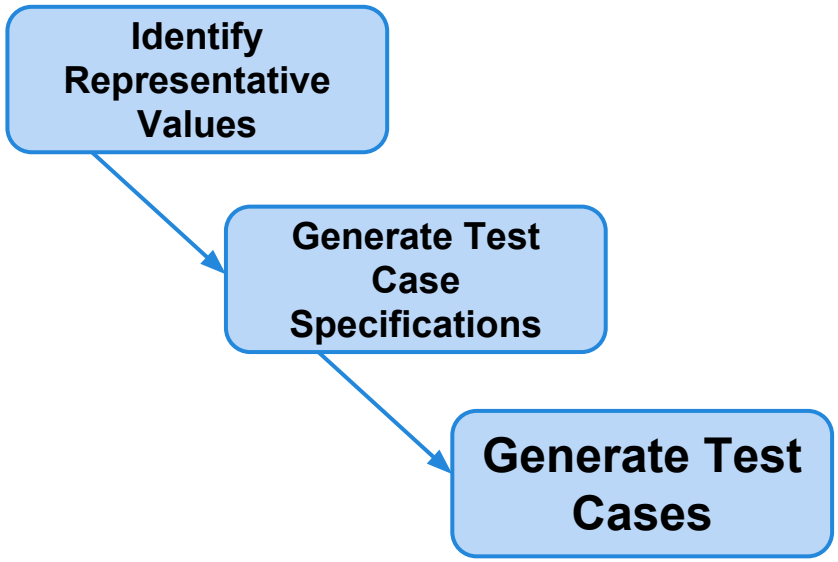
ERROR and SINGLE (

40 Tests!

- Pattern size:
  - Empty
  - single character **[not empty]**
  - many character **[not empty]**
  - longer than any line in the file **[not empty]** **[error]**
- Quoting:
  - pattern is quoted **[quoted]** **[if not empty]**
  - not quoted **[if not empty]**
  - improperly quoted **[if not empty]** **[error]**
- Embedded spaces:
  - No spaces
  - One space **[if not empty and quoted]**
  - Several spaces **[if not empty and quoted]**
- Embedded quotes:
  - no quotes
  - one quote **[if not empty]**
  - several quotes **[if not empty and quoted]** **[single]**
- File name:
  - Existing file name
  - no file with this name **[error]**
- Number of occurrence of pattern in file:
  - None **[if not empty]** **[single]**
  - exactly one **[match]** **[if not empty]**
  - more than one **[match]** **[if not empty]**
- Pattern occurrences on target line:
  - One **[if match]**
  - more than one **[if match]** **[single]**

Combinatorial Interaction Testing

Dealing With All of These Test Specifications



- Category-partition testing takes exhaustive enumeration as a base approach and adds constraints to reduce the number of tests.
- This is only reasonable when constraints reflect real conditions.
- If constraints are added solely to reduce the number of combinations, then you will produce bad tests.

Website Display Options

- Display Mode
  - full-graphics
  - text-only
  - limited-bandwidth
- Color
  - monochrome
  - color-map
  - 16-bit
  - true-color
- Language
  - English
  - French
  - Spanish
  - Portuguese
- Screen Size
  - Handheld
  - Laptop
  - Full-size
- Fonts
  - Minimal
  - Standard
  - Document-loaded

Combinatorial Interaction Testing

- Some parameter combinations may cause faults, so we can't just try each choice once.
- But we do not need all combinations either - many will be redundant.
- Instead, pick a number $k < n$ (n = number of parameters), and generate all k-way combinations.
  - Exhaustive enumeration grows exponentially with the number of parameters.
  - Pairwise combinations grow logarithmically.

Combinatorial Interaction Testing

- Choose two parameters, enumerate all combinations.
- Adding the third is multiplicative.
- Instead, consider all n-way combinations of values
  - (2-way in this case)
- Each tuple contains three pairings. Careful selection of those pairings covers more combinations.

Display-Mode	Screen Size	Fonts
Full-graphics	Handheld	Minimal
Full-graphics	Laptop	Standard
Full-graphics	Fullsize	Document-Loaded
Text-Only	Handheld	Standard
Text-Only	Laptop	Document-Loaded
Text-Only	Fullsize	Minimal
Limited-Bandwidth	Handheld	Document-Loaded
Limited-Bandwidth	Laptop	Minimal
Limited-Bandwidth	Fullsize	Standard

Covering Arrays

- In functional testing, we want to *cover* a large number of the parameter combinations.
  - We seek coverage of strength *k*.
    - *k=n* means we have covered all combinations.
    - *k < n* means all *k-way* combinations are covered.
- A covering array of strength *k* is the smallest array that covers all *k-way* combinations.
- Finding the smallest array is NP-hard.
  - However, greedy and heuristic searches can produce near-optimal solutions.

Example - Online Shop

Card Type	Card Number	Expr. Date	Product Type	Quantity
Amex	Correct	1+ Year	Book	1
Discover	Incorrect Length	Today	Video	0
Visa	Invalid Digits	Yesterday	Software	-1
		Invalid Date		2

- 432 possible test cases (3*3*4*3*4)
- How many if we cover all pairs of values?

Expr Date	Quantity	Card Type
1+ Year	1	
1+ Year	0	
1+ Year	-1	
1+ Year	2	
Today	1	
Today	0	
Today	-1	
Today	2	
Yesterday	1	
Yesterday	0	
Yesterday	-1	
Yesterday	2	
Invalid Date	1	
Invalid Date	0	
Invalid Date	-1	
Invalid Date	2	

Expr Date	Quantity	Card Type
1+ Year	1	Amex
1+ Year	0	Discover
1+ Year	-1	Visa
1+ Year	2	Amex
Today	1	Discover
Today	0	Visa
Today	-1	Amex
Today	2	Discover
Yesterday	1	Visa
Yesterday	0	Amex
Yesterday	-1	Discover
Yesterday	2	Visa
Invalid Date	1	Amex
Invalid Date	0	Discover
Invalid Date	-1	Visa
Invalid Date	2	(any value)

Constraining the Combinations

- Some combinations may not be possible in practice. Constraints can be used to remove invalid combinations.
  - Monochrome is only an option for handheld displays.
  - So, we remove any pairing of monochrome with laptop or full-size displays.

We Have Learned

- Requirements-based tests are derived by
  - identifying independently testable features
  - partitioning their input/output to identify equivalence partitions
  - combining inputs into test specifications
    - and removing impossible combinations
  - then choosing concrete test values for each specification

We Have Learned

- We may have too many test specifications to realistically implement.
  - Can impose constraints through category-partition testing.
  - Can use combinatorial interaction testing to cover all *n-way* pairs efficiently.

Next Class

- Assessing test suite adequacy
  - How do we measure “good enough” testing?
- Structural testing
  - Deriving tests from the source code of the system.

- Reading: Chapter 9, 12
- Homework:
  - Assignment 1 is out. Due February 8.
  - Any questions?