Specification-based Testing

Meta Activity: Team Disruptions

Common disruptions to learning in teams include: talking about topics that are off-task, teammates answering questions on their own, entire teams working alone, limited or no communication between teammates, arguing or being disrespectful, rushing to complete the activity, not being an active teammate, not coming to a consensus about an answer, writing incomplete answers or explanations, ignoring ideas from one or more teammates.

Questions (10 min) **Start time:**

1. Pick four of the disruptions listed above. For each one, find something from the role cards that could help improve the team's success. Use a different role for each disruption.

a) Manager:

limited communication between teammates

b) Presenter:

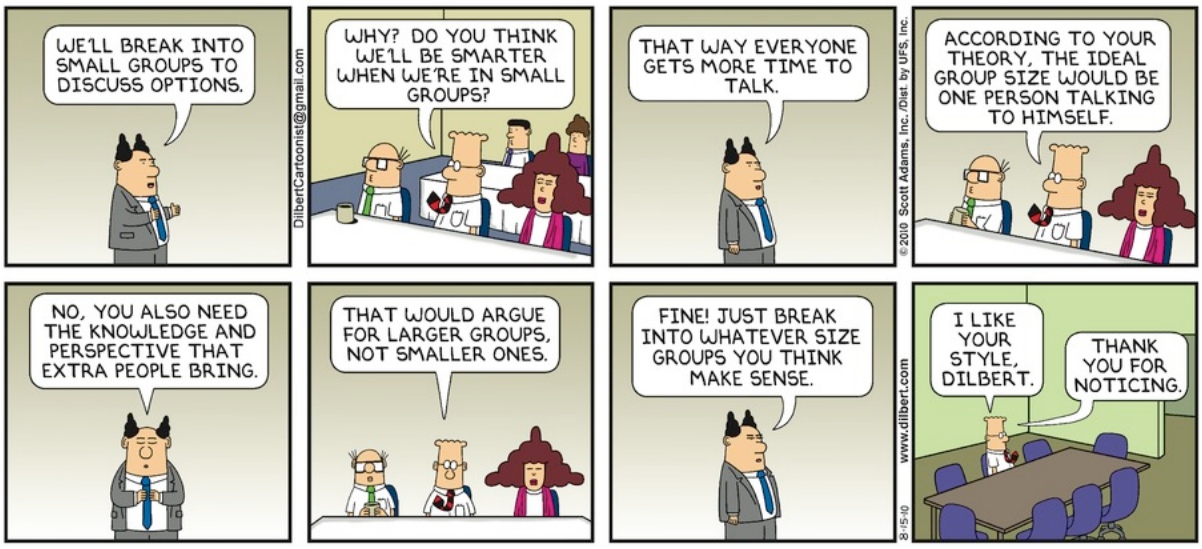
ignoring ideas from one or more teammates

c) Recorder:

writing incomplete answers or explanations

d) Reflector:

teammates answering questions on their own



Dilbert by Scott Adams. © Andrews McMeel Syndication. http://dilbert.com/strip/2010-08-15

Model 1 Specification-based Testing (Black-box testing)

Last week we started categorizing tests. This week we expand on that categorizing and generalize the methodologies it encompassed. Two main methodologies in black-box testing (or specification-based testing) are ***equivalence partitioning*** and ***boundary value analysis***. Equivalence partitioning involves:

a) Looking at each input individually and asking: "What are the possible classes (or partitions) of inputs I can provide?"

b) Combining these partitions when dealing with multiple inputs.

c) Looking at the different classes of output: "Does it return arrays? Can it return an empty array? Can it return nulls?"

Boundary value analysis involves identifying ***on points***, ***off points***, ***in points*** and ***out points*** at the boundary between the different partitions and making sure that your test suite covers all those cases.

substringsBetween

Method `substringsBetween()` searches a string for substrings delimited by a start and end tag, returning all matching substrings in an array.

- str—The string containing the substrings. Null returns null; an empty string returns another empty string.
- open—The string identifying the start of the substring. An empty string returns null.
- close—The string identifying the end of the substring. An empty string returns null.

The method returns a string array of substrings, or null if there is no match.

2. What are the different categories and the partitions in each category for this problem?

Input: The str category with four partitions (null string, empty string, string of length 1, and string of length > 1), the open category with four partitions (the same as str), the close category with four partitions (also the same as str), and the (str, open, close) category with five partitions (string does not contain either the open or close tags, string contains the open tag but does not contain the close tag, string contains the close tag but does not contain the open tag, string contains both the open and close tags, string contains both the open and close tags multiple times)
Output: array of strings (null array, empty array, single item, multiple item) and each individual string (empty, single character, multiple character)

3. The chapter claims that an upper bound for the number of tests can be obtained by combining the input partitions alone (4 x 4 x 4 x 5 = 320). Why were the partitions for the output not included?

The inputs give all the possible kinds of output (cover the entire space of possibilities).

4. In your own words, explain why and how we can prune the original set of 320 possible tests to get to the 21 described in the chapter?

The null and empty string partitions may be tested only once and not more than that.
For the string of length 1 case, given that the string has length 1, two tests may be enough: one where the single character in the string matches open and close, and one where it does not.

5. What are some interesting boundaries between the partitions identified earlier? Provide concrete ***on point***, ***off point***, ***in point*** and ***out point*** tests for each.

Between empty and non-empty string: (”,’a’,’b’) vs. (’z’,’a’,’b’)
Between no string and one string for output: (’xy’,’a’,’b’) vs. (’axb’,’a’,’b’)

addDigits

The method receives two numbers, left and right (each represented as a list of digits), adds them, and returns the result as a list of digits. Each element in the left and right lists of digits should be a number from [0–9]. An `IllegalArgumentException` is thrown if this pre-condition does not hold.

- left—A list containing the left number. Null returns null; empty means 0.
- right—A list containing the right number. Null returns null; empty means 0.

The program returns the sum of left and right as a list of digits.

6. What are the different categories and the partitions in each category for this problem?

Partition: Invalid input
- null
Partition: Input representation (size of the list + leading zeros)
- Empty array, Single digit , Multiple digits, Zeroes on the left for each input list
Partition: Relative length of the lists
– length(left list) > length(right list)
– length(left list) < length(right list)
– length(left list) = length(right list)
Partition: Outcome (carry or no carry)
– Sum without a carry
– Sum with a carry: one carry at the beginning
– Sum with a carry: one carry in the middle
– Sum with a carry: many carries
– Sum with a carry: many carries, not in a row
– Sum with a carry: carry propagated to a new (most significant) digit

7. What are the most interesting combinations between these partitions for our test suite?

Nulls and empties (4 for either being null or empty)
Single digits (2, both lists of size 1, one test with carry and one without)
Multiple digits with same size (6 with all the 6 outcome conditions of carry and no carry)
Multiple digits with different lengths (one for left longer than right, and one for right longer than left) (6 x 2)
Zeroes on the left (2, one test with carry and one without)

8. What are some interesting boundaries between the partitions identified earlier? Provide concrete ***on point***, ***off point***, ***in point*** and ***out point*** tests for each.

On point for carry to a new most significant digit: 99 + 1 (the right list contributes to the first carry, but does not contribute to the second carry).

subtractDigits

Let's consider the complement operation of add: subtract. This method is identical to the add method, except that it subtracts digits instead of adding them.

9. What are the different categories and the partitions in each category for this problem?

Partition: Invalid input
- null
Partition: Input representation (size of the list + leading zeros)
- Empty array, Single digit , Multiple digits, Zeroes on the left for each input list
Partition: Relative length of the lists
– length(left list) > length(right list)
– length(left list) < length(right list)
– length(left list) = length(right list)
Partition: Outcome (borrow or no borrow)
– subtract without borrow
– subtract with a borrow: one borrow at the beginning
– subtract with a borrow: one borrow in the middle
– subtract with a borrow: many borrows
– subtract with a borrow: many borrows, not in a row
– subtract with a borrow: borrow from the most significant digit

10. What are the most interesting combinations between these partitions for our test suite?

Nulls and empties (4 for either being null or empty)
Single digits (2, one with borrow and one without)
Multiple digits same length (6 for all outcomes)
Multiple digits with different lengths (5 all of the outcomes except borrow in the least significant digit because it's covered by the single digits test)
Zeroes on the left (2, one with borrow and one without)
Same number (2, one single and one multiple)

11. What are some interesting boundaries between the partitions for this problem? Provide concrete ***on point***, ***off point***, ***in point*** and ***out point*** tests for each.

On point for borrow from the most significant with equal digit at the same index: [2,1,0] - [1,1,1] = [9,9]