Unit Testing in Angular

Agenda

- Introduction
- Part One: Testing strategies
- Part Two: Applied testing strategies in Angular
- Conclusion
- Q/A

Introduction

Unit testing is the practice of software engineering for automating test of individual units of a program with the intention to detects defects. Although the relation between unit tests and software quality remains unclear, some studies have shown that teams practicing unit test had lower software defects.

Part One

The most important consideration in program testing is the design and creation of test cases. How any case will be required to test the software ? It is practically impossible. Given constraints in time and cost, it is of the best interest to think about What subset of all tests case has the highest probability of detecting the most errors ? To this question, there exist different strategies which are:

- Functional Testing
- Structural Testing

Functional Testing

Functional Testing is a higher level testing in which we validate that the software is working as specified. Functional Testing happens early in the Software Testing lifecycle at the specification phase. It help deduce test scenarios and find early bug or incoherence in the software. There exist many techniques, but we are going to evaluate two of them:

- Decision Table
- Cause-Effect Graphing Techniques

Decision Table

Decision create a structure that make it easier to visualize business logic cases.

Structure

		Rule 1	Rule 2	Rule 3	Rule 4
IF	Condition 1	Y	Y	N	N
AND	Condition 2	Y	N	Y	Y
AND	Condition 3	–	N	Y	–
AND	Condition 4	–	–	Y	N
THEN	Action 1	X		X	
AND	Action 2	X			X
AND	Action 3		X		
AND	Action 4		X	X	

Steps to create

- List all stubs conditions (m)
- Calculate the number of possible rules: `Number of Rules N = 2^m (m: number of stubs conditions)`
- Place all the conditions in a table
- Check covered conditions
- Fill the table with actions

Example

	As admin, I'd like to trigger jobs between 02:AM and 06:AM or let the system do it automatically.							
	Stubs conditions							
	Is user == ADMIN	Y	Y	Y	N	N	N	N	Y
	Is time between 02:06AM	N	Y	Y	Y	N	N	Y	N
	Is user == SYSTEM	N	N	Y	Y	Y	N	N	Y
	Export data	No	Yes	?	Yes	No	No	No	?
	Questions:							
	1: Open question to Product owner ? Do the system export automatically between 02:am and 06:AM ?							
	2: Is it possible to have the ADMIN user and SYSTEM at the simultanously ?							

Summary

- Provide great overview of possible scenario
- Provide new insight and missing assumptions of state(i.e: If we have YYY)
- However can become unusable if initial number of stubs conditions is too hight.

Cause-Effect Graphing

One drawback of decision table is that the number of stub conditions can explode quickly and makes the technic useless. Image having 9 conditions, we will end up with `n = 2^9 = 512 rules`. So testing combination can be very large.

Cause-Effect Graphing is the use of Graph in Combination with Logical operator (AND, OR, NOT) to design software requirements. Cause-Effect Graphing help reduce the initial testing combination and help select, in a systematic way, a high-yield of test cases.

Steps to create a Graph

1. Identify all causes
2. Identify all effects
3. Design cause and effect graph
4. Apply constraints
5. Design limited entry decision table from graph
6. Write test cases using every column of the decision table

Example

> As an Admin I would like to export data between 02-06AM or let the system do it automatically.
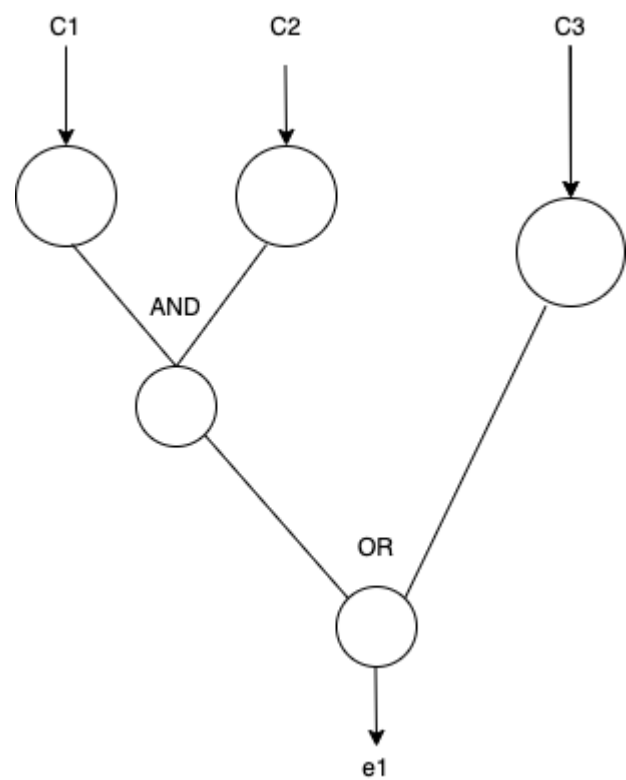
Solution

Causes

- c1: Is Admin
- c2: We are between 02-06 AM
- c3: It is admin

Effects

- e1: Export data

Graph

Decision Table

We can see from this picture that there are three test three test cases:

- IS ADMIN and Is between 02-06 AM then export data
- Is system then export data. (When ? On user click ? between 02-06: AM) ?
- Nothing if neither (ADMIN and between 02-06 am) or IS SYSTEM

Stubs conditions				
Is user == ADMIN	Y			
Is time between 02:06AM	Y			
Is user == SYSTEM		Y		
Export data	Yes	Yes		
Question				
1. When do the system user export data ?				

Summary

- Easy to produce minimal test subset sufficient for the specification

Structural Testing

Structural testing is, essentially looking into the implementation and deduce tests cases that will cover the implementation. This happens generally late in the implementation phase of the software.

Path coverage using Cyclomatic complexity

Cyclomatic complexity is the deduction of possible executions from the analysis of the implementation of a program/function. It help create a subset of test that greatly satisfy the coverage of a given implementation.
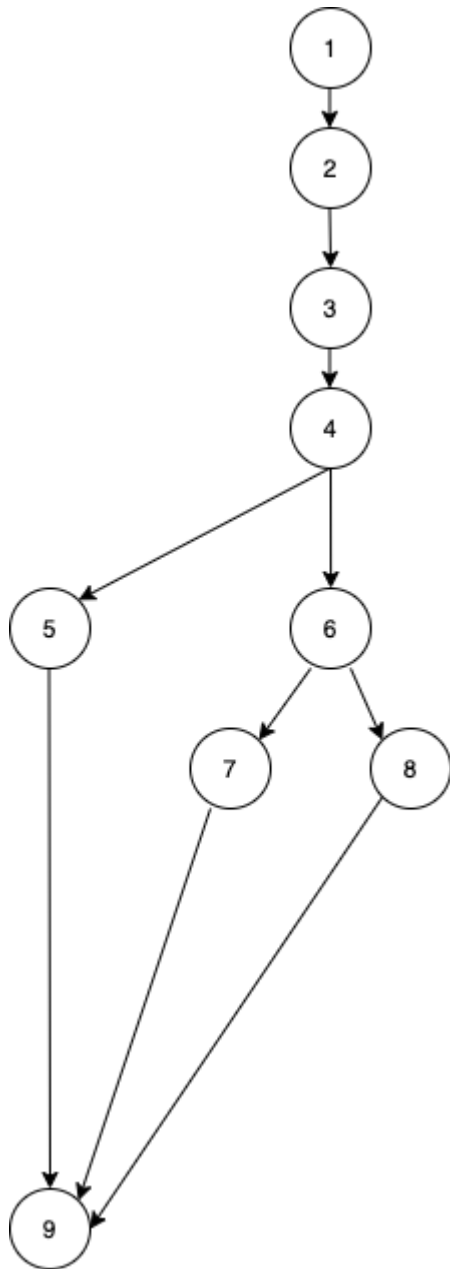
Example

Let's consider the current function for exporting data:

```
const executeExport = (user: User) => {  //(1)
    let result = true; // (2)
    const date = new Date(); // (3)
    if(user?.role === 'ADMIN' && this.timeIsBetween02and06(date)) { //(4)
        this.executeExport(); // (5)
    } else if(this.user?.role === 'SYSTEM') { // (6)
        this.executeExport(); // (7)
    } else {
        result = false;// (8)
    }
    return result; // (9)
}
```
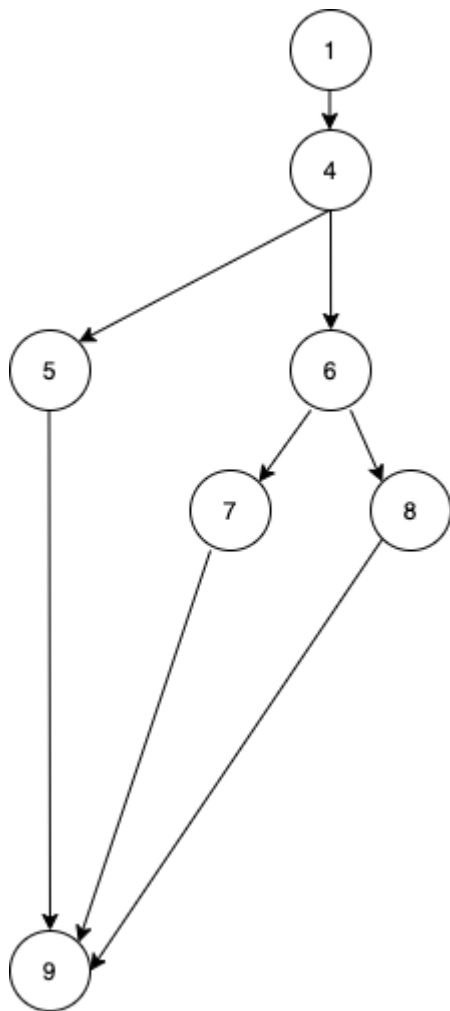
The corresponding path diagram

Cyclomatic complexity:

V = e - n + 2P = 10 - 9 + 2 = 3

We have the simplified graph

Cyclomatic complexity:

V = e - n + 2P = 8 - 7 + 2 = 3

The following possible executions path that need to be covered

> a: 1 -> 4 -> 5 -> 9

> b: 1 -> 4 -> 6 -> 7 -> 9

> c: 1 -> 4 -> 6 -> 8 -> 9

Disadvantage

This implementation doesn't tell us if the functionality is correct or not. For example do the Product Owner expect the 'SYSTEM' user to be able to execute export only between 02-06 am ? It is hence important to very the business logic using a Decision Table or a Cause-Effect graph before starting an implementation.