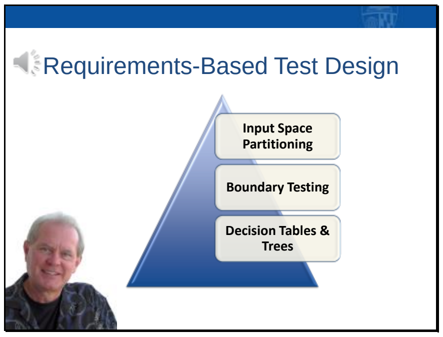
1

Software Testing

Requirements-Based Test Design

In this lecture, I'm going to discuss requirements-based test design.

2



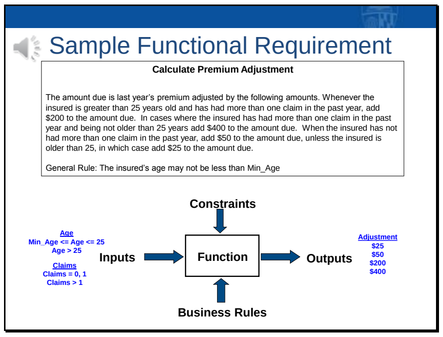
Requirements-Based Test Design

In this lecture I'm going to discuss several black box techniques that can be used to design test cases based on the requirements for a software product...input space partitioning, boundary testing, and decision tables and decision trees.

One of the nice things about these techniques is that they are systematic and repeatable...you can replicate how many tests you need, and demonstrate why a certain number of tests are needed, which can give you lots of credibility when someone asks how you came up with a certain number of tests.

And...they can be used when requirements are written using a traditional approach as well as when use cases are used to document a product's functional requirements.

3



Sample Functional Requirement

Calculate Premium Adjustment

The amount due is last year's premium adjusted by the following amounts. Whenever the insured is greater than 25 years old and has had more than one claim in the past year, add $200 to the amount due. In cases where the insured has had more than one claim in the past year and being not older than 25 years add $400 to the amount due. When the insured has not had more than one claim in the past year, add $50 to the amount due, unless the insured is older than 25, in which case add $25 to the amount due.

General Rule: The insured's age may not be less than Min_Age

Let's take an example and see how these techniques can be applied.

Suppose an insurance company application needed to calculate policy premium adjustments that are based on a person's age and the number of claims filed in the past year. The textual description describes this requirement in detail.

We're going to start by mapping a requirement's inputs and outputs to the functional requirements model that was introduced earlier in this course. That will be very helpful when we ultimately derive test case data, and it fits nicely into the techniques we're going to discuss.

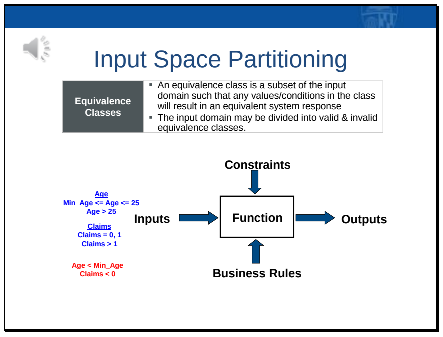
So...let's map the inputs and outputs for this requirement…calculate premium adjustment…into our functional requirement model.

The inputs in this case are two variables …an age and a number of claims. And these input variables could further be broken down into subcategories as indicated. Note that there is a general rule that specifies the minimum insurable age.

The output in this example would be one of four adjustment amounts.

And…most of the text you see in the box would be the business rules…which specify how to calculate the adjustment given the input values.

4



Input Space Partitioning

Equivalence Classes

- An equivalence class is a subset of the input domain such that any values/conditions in the class will result in an equivalent system response
- The input domain may be divided into valid & invalid equivalence classes.

The first technique that will help us to design tests is called input space partitioning.

Input space partitioning is used to partition the input side of a function into a set of equivalence classes that can help us identify test conditions.

An equivalence class is a subset of the input domain such that any values in that class will result in an equivalent system response.

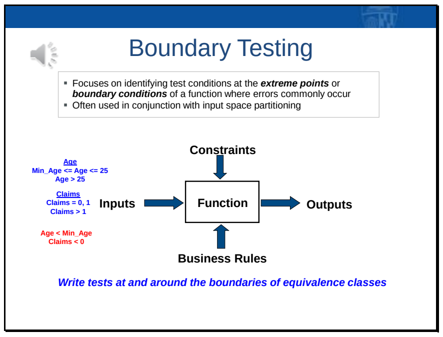
For example, a person who is 29 years old and has no claims will have the same adjustment as a person who is 40 and has no claims. Similarly, a person who is 21 with 3 claims will have the same adjustment as someone who is 25 and has 3 claims.

We also model the invalid equivalence classes. In this example, two invalid classes are age less than the minimum allowed and a negative number of claims. We could also set up an invalid equivalence class for invalid types, but we won‘t do that to keep things simple.

Modeling the inputs of a function using equivalence classes helps us to see possibilitied for reducing the number of tests we need to develop and to ensure we adequately cover the input possibilities...and helps in the boundary testing technique as well.

At this point, we could generate test cases, by essentially taking random values that fall within each of the equivalence classes. For example, we could have a test case for a person who is 24 and has 1 claim, and a person who is 24 and has 3 claims, and a person who is 33 and has no claims, and a person who is 33 and has 4 claims.

5



Boundary Testing

- Focuses on identifying test conditions at the ***extreme points*** or ***boundary conditions*** of a function where errors commonly occur
- Often used in conjunction with input space partitioning

Age
Min_Age <= Age <= 25
Age > 25
Claims
Claims = 0, 1
Claims > 1
Age < Min_Age
Claims < 0
Constraints
Inputs
Function
Outputs
Business Rules

Write tests at and around the boundaries of equivalence classes

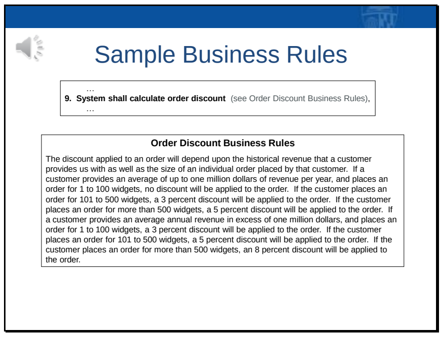
The next technique is the boundary testing technique… and here's a formal definition.

Boundary testing focuses on identifying test conditions at the extreme points of the equivalence classes.

In the premium adjustment example we would use this technique to write test cases around the boundaries of the two input variables.

So…maybe we'll add tests for someone who is 25 and has no claims, someone who is 25 and has 1 claim, someone who is at the minimum age with no claims, someone at the minimum age with one claim, someone at the minimum age with 2 claims, someone below the minimum age with 1 claim, and someone below the minimum age with a negative number of claims.

6



Sample Business Rules

…
9. System shall calculate order discount (see Order Discount Business Rules),
…

Order Discount Business Rules

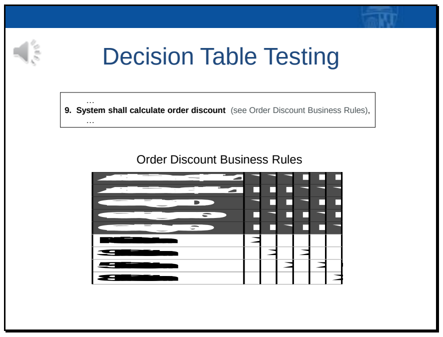
The discount applied to an order will depend upon the historical revenue that a customer provides us with as well as the size of an individual order placed by that customer. If a customer provides an average of up to one million dollars of revenue per year, and places an order for 1 to 100 widgets, no discount will be applied to the order. If the customer places an order for 101 to 500 widgets, a 3 percent discount will be applied to the order. If the customer places an order for more than 500 widgets, a 5 percent discount will be applied to the order. If a customer provides an average annual revenue in excess of one million dollars, and places an order for 1 to 100 widgets, a 3 percent discount will be applied to the order. If the customer places an order for 101 to 500 widgets, a 5 percent discount will be applied to the order. If the customer places an order for more than 500 widgets, an 8 percent discount will be applied to the order.

Earlier in the course we discussed using decision tables to document complex business rules as part of the requirements phase in a project. Decision tables can also be used to help us design test cases.

Here's an example in which a step in the use case requires the software product to make a calculation. The business rules for the calculation are pretty complex, and have been documented in narrative form. The narrative is clear and nicely structured…but the rules are innately complex. From the testing perspective, how many test cases should be used to test this function? It's not obvious at all, is it?

Let's use a decision table to model the business rules and see if that can help us out.

7



Decision Table Testing

…
9. System shall calculate order discount (see Order Discount Business Rules),
…

Order Discount Business Rules

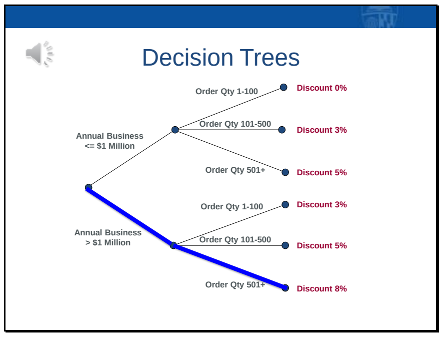
[illegible]

Here's the decision table. It has six columns, corresponding to the basic functional combinations specified in the business rules. So…we can write a test for each column in the table and we'll be certain that all the basic combinations are covered. Pretty simple.

We can also apply the boundary and input space partitioning techniques to get a full set of test cases.

Now, if a decision table had been used in the requirements, we'd save a lot more time developing test cases, wouldn't we? I actually worked on a project in which almost 50% of the expected test effort was saved because the business analysts used decision tables for specifying complex business rules.

8



Decision Trees

Yet another technique is to use decision trees. This is a decision tree for the order discount business rules.

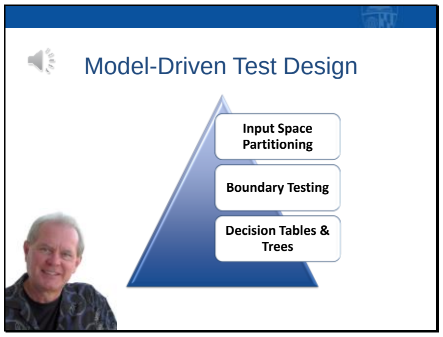
The node at the left-most part of the tree is called the root of the tree. The straight lines are called the branches of the tree, and the nodes at the right-most end of the tree are called the leaves of the tree. Each set of branches emanating from a node corresponds to possible equivalence classes for one of the input variables.

You read the tree from left to right. As an example...if we have a transactions for a customer that provides more than a million dollars in revenue , and the order quantity is 800 widgets , then the order discount will be 8 percent.

In this tree, each path from the root of the tree to a leaf on the tree corresponds to a column in the prior decision table. There are 6 paths, corresponding to the 6 columns in the decision table example.

Decision trees are more useful than decision tables when there is a business rule algorithm that requires iterating through steps more than once.

9



Model-Driven Test Design

The techniques I just discussed are all examples of model-driven test design. We started with a requirement, modeled its business rules, and then used the model to help generate test cases.

Note that using these techniques we can quickly estimate how many test cases would be required...without having to actually come up with specific test case data. That's very powerful...because it can help with estimating total test effort very quickly during the test planning activity. And...if we applied the techniques over and over, we'd come up with pretty much the same number of test cases...so the techniques are repeatable...and defensable.