Requirements-based testing that finds all faults - part 1.

Mike Holcombe.

This is the first of 6 articles that focus on a new technique for system testing that is based on the generation of test sets from requirements. One important property of this method is that the tests will reveal ALL possible faults in an implementation provided certain conditions, conditions that are both practical and reasonable, hold. Full details are in [1].

Identifying the business processes.

Introduction.

Our starting point is: the clients (and the prospective users of any solution) are the *most important individuals* in the exercise. It is only by understanding *their needs* and *their objectives* that the designers can achieve their's - which should be client satisfaction and high user productivity.

It has been widely commented that one of the key causes of the failure of a software design project is lack of understanding by the designers of the clients requirements. It could also be mentioned that the client may, also, not be fully appreciative of their problem, let alone a possible solution.

However, it is not even as simple as that, businesses change, often rapidly, and so the problem alters and so many projects suffer from late changes in requirements - which can result in serious failure.

It is partly these problems that are motivating the interest in the lightweight approaches to software construction such as eXtreme Programming (XP), ([2, 3]), an approach that actually puts testing FIRST and abandons heavyweight design notations.

We are thus brought to the position where we must find some effective mechanism to enable us to describe and understand the client's problem before we can even contemplate the construction of a solution. It is from these requirements that we generate test cases. The tester can only do so much to prevent disasters caused by requirements change, ensuring that new test cases are generated from the changed requirements will benefit the project.

Our first starting point is to examine the context of the client's problem area and to build a *simple business process model*. If we can insure ourselves, somehow, from the changing requirements problem as well that would be an advantage.

Identifying business processes.

There are many ways in which a business may be modelled some of which have been made into methods supported by tools of various types. There is, however, a lack of precision about these techniques which makes them unsuitable for the sort of analysis that we wish to carry out. We will introduce an alternative and simple method which can be used to derive detailed models of the systems that we wish to build to support the business process

Many processes share the following common features:

- there is some *receipt* of information or data necessary for the process to operate;
- there are some aspects of the *context* of the business, internal knowledge of some sort required for the process to operate;
- the process then *evaluates* the received data in the light of this context and produces or *generates* some resulting observable output, information or other product;
- the context is *revised* or updated ready for the next time that a process operates.

Examples of this might include the process of updating a customer's records, whereby the new information about the customer is fed into the process, the database provides the context and the output is the confirmation through printout or screen display that the record has been updated. The database updating is the context being revised in the light of this process operating.

We now form a table listing these processes, together with the events and operations that they involve. As a strategy to try to identify the parts of the requirements that might change during the course of the project we estimate how *stable* that requirement might be in the table. This is, of course, hard to do but it does at least force the designer and the client to try to look forward to see how the business is changing and what the implications might be. As we have noted above, the problems caused by changing requirements are extremely serious.

Example.
Suppose that we are building a simple customer and orders database. We might identify a number of *user stories* such as the following:

1. Customer details are entered customer by customer.
2. Customer details can be edited.
3. Orders are placed by a customer
4. Orders can be edited when necessary.

The details of the structure of the customer and orders details are not described at this level of detail. We try to build an abstract model of the user interface and then refine it. The test approach permits us to generate an abstract high level test strategy and to refine the test cases *in parallel* with the design thus saving enormously in test case size for large examples - a recent case study, involving 3 million transitions, demonstrated this, [4]. In other industrial projects requirements have been split down into small functional elements, implemented and tested separately and the integration guided by the requirements-based testing strategy.

story	function	input	current memory	output	updated memory	change risk
1	click(customer)	customer button click	-	new customer screen	-	low
1	enter(customer)	customer details entered	current customer database	confirmation details screen	-	medium (nature of details liable to change)

1	confirm(customer)	customer confirm button clicked	current customer database	OK message and start screen button	updated customer database	low
3	click(order)	orders button clicked	-	new orders screen	-	low
3	enter(order)	new order details entered	current orders database	confirmation orders screen	-	high (nature of details of orders liable to change)
3	confirm(order)	orders confirm button clicked	current orders database	Ok message and start screen button	updated orders database	low
3	quit()	click on return to start button	-	start screen	-	low

We infer, from the user stories and the table, an interface with the user that includes screens with buttons labelled customer, order, OK etc. We also infer a screen which enables the details of what an order is to be input through the keyboard, drop-down menu etc. We also infer that there will be some message screens and other types of information output involved in a process. Finally we have the interal memory or database which, in this case, could be implemened in a variety of ways, a relational database, a simple flat file structure or an XML tagged text file. The details are not important at this stage it is a mechanism where we can store information about customers, orders etc..

We can thus identify the set or type, **input**, of data (including the events such as mouse clicks on buttons, typing data entry values in forms etc.) that it is possible to supply the process, a set, **output**, of possible output data (screen changes, calculated values etc.) and a set, **memory**, consisting of the internal context or data associated with the process (which might be the contents of some cells in a database table) then we can model the business process as a type of function of the form:-

process: (*memory*, *input*) = (*output, memory*)

Now, each process may be quite a complex operation and it is likely that we will need to break it up into a collection of smaller processes, sometimes these will be just simpler processes organised in a sequence of actions, however, more complex arrangements are also likely. We will return to this issue in part 2.

In our example the input click(customer) corresponds to the input caused by clicking a mouse over the customer button.

In the next part we will discuss how we can organise processes into what we call *enterprises*, a more complete and self contained system which will provide the foundation for our test generation process.

References.

1. M. Holcombe & F. Ipate, "Correct systems - building a business process solution", Springer, Applied Computing Series, 1998.
2. K. Beck, "Extreme Programming Explained: Embrace Change", Addison Wesley, 1999.
3. XProgramming Web site, On-line at <http://www.XProgramming.com/ >

4. K. Bogdanov & M. Holcombe, "Test generation for a statechart with a relatively large number of states", submitted.