

# Replace and renew

Professor Mike Holcombe completes his series “Requirements-based Testing That Finds All Faults” by explaining how to adapt for change when using the X-Machine approach

*Dealing with change is a constant issue in software development.* Changing requirements, changing technology and changing personnel have a major impact on the way a project will develop. These changes may impact on the testing. Although we usually encourage that testing is considered at an early stage of a project it is often a problem if the project is changing in such a way that testing a moving target becomes difficult.

## Requirements change

Changes to the requirements are bound to occur and the way that we deal with them will be a vital aspect of a successful project.

There are a number of different manifestations of requirements change, some are more serious than others. We have been trying to identify those areas of the system that might be subject to change from an early stage in our requirements capture and analysis. Hopefully, we will not be too far wrong but you can never tell.

We will consider several types of change and how to deal with them. Some are serious and will involve us in redoing a lot of our previous work, some are more easily dealt with and won't affect the project outcome too much. There is always a price to pay if the change is significant and the client should realise this. We have to be agile and adaptable as well as to be honest with the client and to talk to him/her frequently. That way we may see the changes coming and prepare for them. We should also explain to the client the costs of the changes, the delays that may occur, the reduction in quality if it isn't thought through properly and so on. Ask the question: do you

really need this change? If the change is fundamental to the way the client's business or organisation is evolving then we need to embrace it with enthusiasm.

## Changes to basic business model and functionality

These sorts of changes can occur at almost any stage of the project. Sometimes they are the result of the team not understanding the client's requirements or business processes at the time and are thus a correction of what was originally thought. These changes need to be related to the current state of the project. If the project has developed a requirements document then the changes may require the introduction or substitution of new requirements statements and these should be expanded into stories. Then the changes have to be tracked through the development of the project so that, for example, implications for the underlying database, if one is involved, are considered. When we revisit the integration of the stories into an X-Machine model with its accompanying user interfaces and input and output requirements.

The test sets will need to be redefined properly at the systems level. The classes associated with these changes must be identified and the unit tests updated to reflect the new requirements, the code needs to be re-programmed and tested in the usual way.

It is not always easy to achieve these changes without extra work. If the stories have not been implemented yet then things are much easier. As always the later a change is identified the more expensive it can be. It

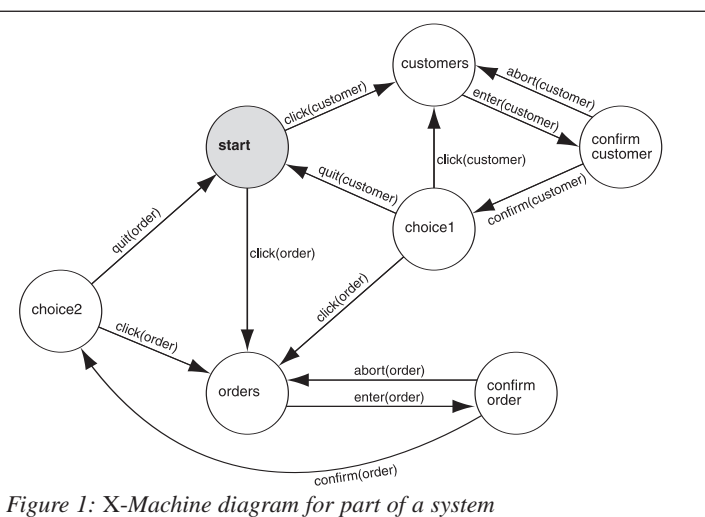


Figure 1: X-Machine diagram for part of a system

seems that agile methodologies are more able to cope with change than others. If the project is fixed scope or fixed price then there will be times when significant change will not be compatible with the fixed end date for project completion.

Whatever the situation it is vital that all the new parts of the system are properly

documented, so that we have to maintain the key information about the system:

- the stories
- the models
- the system tests
- the system metaphor or architecture
- the classes and methods
- the unit tests
- the code
- the user manual and maintenance manual.

This updating may be accompanied by comments to indicate what has been done.

## Dealing with change - refining stories:

When the client tells us that there needs to be a change we have to decide how to deal with this and what it means for the system testing. We need to consider what sort of change it is.

### a) Changes to the underlying data model

Suppose that the new requirement is to have more information about the customers, perhaps an indication of their credit worthiness or whether they qualify for some discount. This involves changing the internal memory of the model, we can do this quite easily and then introduce a new area of the interface to provide the extra functionality.

So we now have:

```
customer_details : name, address,
postcode, phone, fax, email,
discount
```

where discount is either yes or no.

The screen is now changed to allow a discount yes/no button to be chosen or a discount flag to be checked.

This will impact on our test sets by requiring the discount data to be present in the tests. We need to identify all these changes on story cards so that they are properly documented and we are in a position to know what to do.

### b) Changes to the structure of the interface, perhaps the introduction of a new screen

This will mean altering the state machine model by introducing a new state, for example. To access this state a new transition together with an accessing function will have to be

defined. The activities within this screen and the exiting from it will also lead to new transitions and functions that need to be defined. Each of these will identify a new story which, in turn, will define a further requirement.

### c) Adding a new function

Here we are inserting a new transition with its corresponding function into a diagram. Here a similar strategy applies, for every test sequence that gets to the state where this new function originates we develop a new test sequence that triggers the new function. We then complete each of these new test sequences by creating paths through the machine following on from this function. Again, this will lead us to new test sequences.

### d) Changing the functionality of a function

The basic strategy will not be affected here unless there are issues with the preconditions for the function. If the precondition for the operation of the new function is different from that of the replaced function then the test sets may have to be changed in a more subtle way. In other words we may need to test for the non-operation of the function by choosing previous data values and memory values so that the function does not operate. This can only be dealt with on a case by case basis.

to `enter(order)`, as well as altering the lower level diagram where the process is expanded into more atomic processes.

The order details diagram also needs to be changed (figure 2).

their transitions. In fact there is no point in introducing a new state unless there is a process that needs to be dealt with separately. This might be because we decide that a particular case in the business process needs to be dealt with in a separate way and this might mean designing a new user screen specially for this event. This is often better than trying to cover all the possibilities in one screen. The client will have a view on this. So we might then break a process down into several processes with their own states.

The original function which is represented as `f` or `g` is split into two separate functions `f` and `g`.

As before we must update the story cards with the new functions and generate new tests to cover the changing model structure.

The interfaces will also have to be changed and this is important to do at the same time, lest we forget.

The methods implementing the functions will now need different tests and this is another thing that must be attended to.

### Adding processes

The final type of change is where we introduce a new process which will operate between two existing states. Here we need to consider, carefully, the way that this process will be triggered from the state, we could easily get into a non-deterministic situation if there is any mistake here. The input to the new process together with the expected memory condition at that moment must not overlap with the input and memory conditions for any other process that might already be there.

Thus if we have a process of the form:

```
process: input, memory
```

which is valid for a set of pairs of values from the sets `input` and `memory` and another process of the form:

```
process': input', memory'
```

which is valid for a set of pairs of values from the different sets `input'` and `memory'` then we need to make sure that there are no values which satisfy the condition that they belong to both at the same time.

In mathematics, if the domain of `process` is the set  $I \times M$  and the domain of `process'` is the set  $I' \times M'$  then we need that the following holds:

$$(I \times M) \cap (I' \times M') = \emptyset$$

There are several ways in which this can be assured.

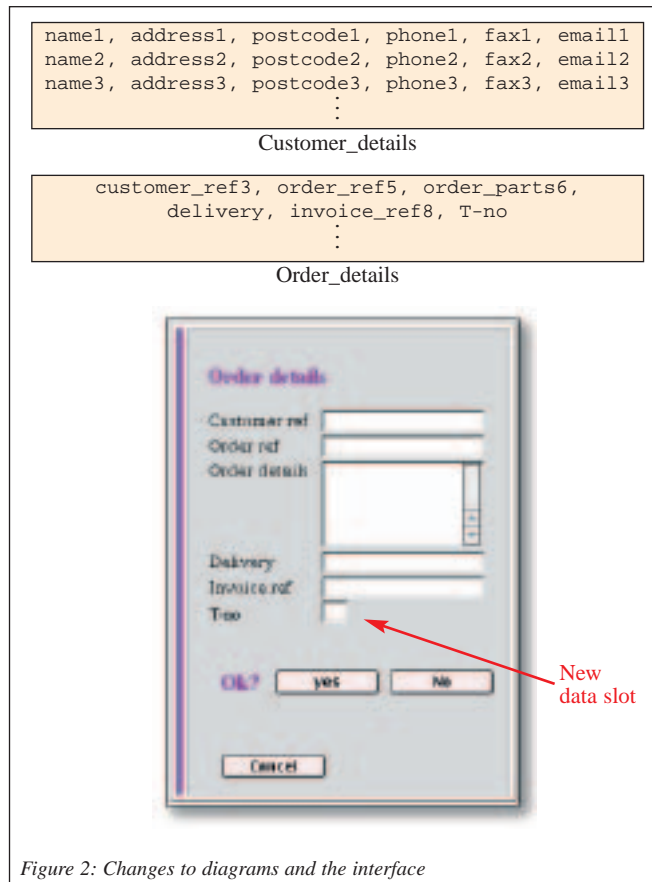


Figure 2: Changes to diagrams and the interface

## 5. Changing the model

The changes are captured using revised story cards and now we need to integrate them into our model so that we can see the effect that they have on the system and how they impact the test sets.

The X-Machine model is built from the user stories in order to provide a basis for functional system testing. The changes will involve a number of different transformations of the model which can be considered separately.

## 6. Changing a process

Suppose that we have the model in figure 1 and the process `enter(order)` is changed in some way; perhaps the information being input is different. This needs to be reflected in two ways. The definition of the function is different and so the interface that provides the user with the capability to input the information will need to be changed. The database will also probably have to change to accommodate the new data being input. Suppose that we need to collect more information about the order, for example the customer's tax number (T-no).

We should make it clear that this process has changed in the model by amending the diagram at the top level by changing the label

### Removing states

Here we consider the issues related to removing a state, perhaps the client does not want a particular feature any longer.

If we remove a state then we must remove all the processes that lead to that state and which all those that leave that state. This may interfere with the flow of business processes and it is vital that we check this thoroughly before committing ourselves to the new model of the requirements.

All those processes (or transitions) can now be removed from our architecture and the user interfaces associated with them also. It is vital that we then revisit the system tests to see what the implications of this are. Since much of the diagram is unchanged all the tests that involve sequences that visit this state can be removed also. Take care over this.

It may be necessary to introduce a new process or two to link states before and after the removed state in order to make the whole system work. We look at this next.

### Adding states

When introducing a new state it will also require the introduction of new processes and

- a) Choosing I and I' to be completely different with no values in common. What this means is that if, for example I is the set of inputs corresponding to all strings of characters of length less than 30, say (perhaps these are names of customers) and I' is the set of all strings of numbers of length less than 10 (maybe these are customer reference numbers) then we can be sure that the two processes do not interfere with each other, in other words when we try to trigger either of the functions then only one can work.
- b) Choosing M and M' to be completely different with no shared values.
- c) If I and I' share values or M and M' share values then we must make sure that there are no pairs which are the same.

For example if I and I' were both all strings of characters of length less than 30 and M and M' were boolean (True or False) then process can only work when M is T and process' can only work when M' is F. At any moment when the computation reaches the start state of the two processes either the memory value in M is T in which case process works or the value of M is F and thus process' operates.

If these conditions are not valid then we must adapt I' or M' to ensure that they are, otherwise we are in danger of designing a non-deterministic machine which could behave unpredictably.

The new function will be a method in some suitable class. It is possible that the class that implements the method corresponding to the new process' is the same class that implements process. In this case we extend the unit tests to include the process' method. Otherwise we create completely new unit tests for the new class.

### Testing for changed requirements

The system testing will now have to take these changes into account. We will either redo the complete system test set - not a good idea because of the work involved - or test the new

part of the machine separately and then runs a smaller number of integrating tests. These could be developed in the following way (see figure 4):

In every test of the form:

```
...g1 ; old_function ;
g2...
```

we create a new test where the function old\_function1 is replaced by a test from the new machine such as:

```
... ; g1 ;
new_function1 ; func2
; func3 ; g2 ; ...
```

This will lead to a number of new tests. We do all the old tests as well.

The system tests also need updating. Since we have extended the state diagram with a new transition we need to write tests that will trigger this transition at all possible occasions. This means both legitimate paths through the updated machine but also we need to try to trigger process' from every other state to ensure that we have not inadvertently introduced additional, unintended, functionality into the system.

In practice we look at the two models, the original one and the new, revised one and focus on those test sequences that involve the changed parts of the machines. In this way we can preserve many of our original tests and know that we will not have to retest them. However, if there is any doubt as to whether you should retest then you should retest!

### Conclusions

These articles have attempted to explain a new approach to testing which is based on building a model of the requirements, namely an X-Machine. Further details on the background theory can be found in [1, 2, 5, 7]. The approach has been tested in a number of cases; see the previous article for an industrial evaluation. The approach is designed around a hierarchical approach which allows a system to be tested in parts and then tested at integration. The remarkable property is that if all the condi-

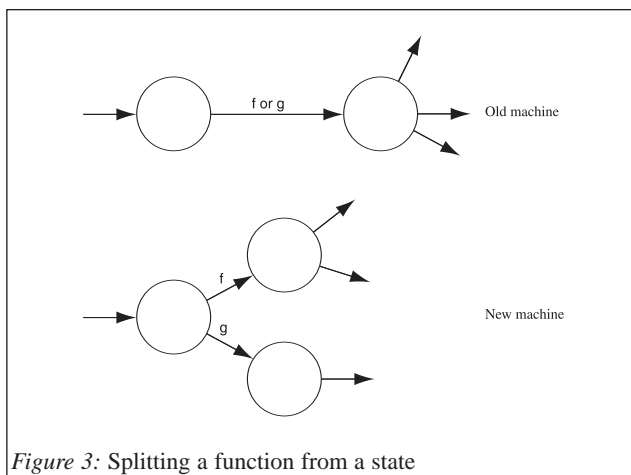


Figure 3: Splitting a function from a state

tions are satisfied then all faults will be detected. It is also scalable; in [4] an example system from the aerospace industry was considered. The approach taken there, which relates to a specific design language - Statecharts - demonstrated that systems with millions of states and transitions can be dealt with in this hierarchical way and, if the software is structured in a similar way to the specification, massive savings in test set size are achieved.

The approach has been tried out in many different applications, the test sets are very powerful and, even if the design for test conditions are not entirely satisfied, easily outperform other test approaches.

### References

- [1] F. Ipate and M. Holcombe, "A method for refining and testing generalised machine specifications." *Int. Jour. Comp. Math* 68, 197-219, 1998.
- [2] F. Ipate and M. Holcombe, "Specification and testing using generalised machines: a presentation and a case study." *Software Testing, Verification and Reliability* 8, 61-81, 1998.
- [3] F. Ipate and M. Holcombe, "Correct Systems - building business process solutions", Springer - Applied Computing Series, 1998. 206 pp.
- [4] K. Bogdanov and M. Holcombe, "Statechart testing method for aircraft control systems", *Software testing, verification and reliability*, 11:39, 54, 2001.
- [5] T. Balanescu, M. Gheorghe, M. Holcombe, F. Ipate, "Testing Collaborative agents defined as stream X-Machines with distributed grammars", ECAL 2001 Prague, Czech Republic, September 10-14, 2001 (accepted for a special issue of LNCS).
- [6] P. Kefalas, M. Holcombe & M. Gheorghe, "A Formal Method for the Development of Agent-based Systems", *Intelligent Agent Software Engineering*, ed. Valentina Plekhanova 2002.

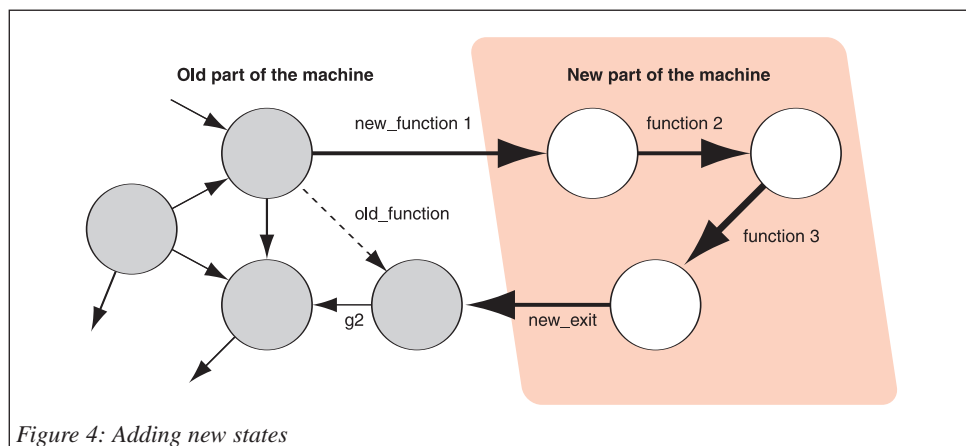


Figure 4: Adding new states