An Entity Model of Software Testing: A Foundation for the Future

John Kent M.Sc.
Email: john.kent@simplytesting.com
Web: http://www.simplytesting.com

Abstract

The language used to describe software testing has been refined over the last twenty years with BSI 7925, ISTQB and IEEE-829 but there is still further work required in defining the entities used in testing. Test processes are discussed a great deal in testing circles, but a process without a good foundation of a definition of the test deliverables and the relationships between them is an incomplete description. This paper describes an entity model of testing. It offers a start at building a model that all testing follows to some degree. It provides a description of requirements, specifications, test conditions, test cases, etc and how they are related to each other. This topic may, at first glance appear to be rather academic but it has highly practical benefits. Many test teams use only a part of the entity model and are, therefore missing out on the benefits of the other parts. Certain entities take longer to build than others and so test organisations could save that most precious of commodities, time, using a more efficient approach to the test entities they build and how they use them. Having a full entity model will enable testers to see where they are and where they can improve their testing.

Introduction

The author has felt for some time that test entities – the things we use to test – are poorly defined. Software Testing standards[1] and glossaries[2, 4] provide textual descriptions of test entities such as test cases. However, these entities do not exist in isolation; they are based upon or linked to other entities. IEEE 829 does offer an entity relationship diagram[5] but the relationships are not well defined, entities are missing (most notably Test Conditions) and specifications or requirements are not included – the very thing that testing is used to verify. Indeed the definitions of testing we have imply relationships between entities in that test process must "determine that they [the software products] satisfy specified requirements"[4]. But what are these relationships? Thus purely textual descriptions of entities cannot be a complete picture without the relationships between the different entities and the best way to do this is using an entity model.

The idea of an entity model for testing was introduced at the Software Testing Retreat, a group that meets several times a year to discuss testing. This paper draws greatly on those discussions and the author is indebted to members of the Software Testing Retreat who contributed.

This is still a work in progress. The entity model offered here is incomplete and the relationships between the entities are still not fully defined but, the author hopes that the model can be taken further in the future by the testing community.

The Entities

The Test Entities in the model include:

- Specifications
- Test Conditions
- Test Cases
- Test Scripts
- Test Steps
- Test Execution Schedule
- Test Results

Notable test entities that are not yet in the model include:

- Defects
- Test Plans
- Test Strategies
- Test Coverage Items

Each of these entities are defined below where possible using current standards and glossaries.

Entity: Specification Item

The word *Specification* is used here to include requirement specifications, functional specifications and design specifications. A *Specification Item* is an indivisible part of a specification and together Specification Items form what is also called the Test Basis. Often testers use the word 'Requirements' in this context, but because the test basis can include all types of specification or some other model of system behavior the term *Specification Item* has been used.

Entity: Test Condition

A definition of the term Test Condition does not appear in either IEEE-829 or BSI 7925. The ISTQB definition is

> *An item or event of a component or system that could be verified by one or more test cases, e.g. a function, transaction, feature, quality attribute, or structural element.*

A Test Condition can be considered to be a refinement of the specification from a test perspective. It is a statement of required system behavior under certain conditions. Perhaps a best definition of what a Test Condition is can be found in test management tools such as Quality Center (Test Director) and T-Plan. In these tools Test Conditions are children of Specification Items. They are used to split the Requirement Specification Items into testable items. Very often they are used to fill in the gaps of incomplete requirements and really could be considered to be specification items as they define what the system behavior should be.

All of the Specification Items and Test Conditions could be considered to be a 'model' of required system behavior.

Entity: Test Case

Test Cases were defined in IEEE 829 and this definition has been widely accepted for many years. The ISTQB definition is:

> *A set of input values, execution preconditions, expected results and execution postconditions, developed for a particular objective or test condition, such as to exercise a particular program path or to verify compliance with a specific requirement. [After IEEE 610]*

Interestingly, this illustrates the need for an entity model because a test case '*developed for a particular objective or test condition'* would be considered to be wrong by many who use a single test case to test *many* test conditions.

Entity: Test Script

Also called Test Procedure, the ISTQB glossary defines a Test Procedure as

> *A document specifying a sequence of actions for the execution of a test. Also known as test script or manual test script. [After IEEE 829]*

Very often a test script consists of Test Steps.

Entity: Test Step

Unfortunately neither ISTQB nor BSI 7925 define what a test step is, but they do exist. In fact they are very common in testing circles. They at the very least are a way of splitting up Test Scripts and may possibly be a Test Case within a Test Script.

Entity: Test Execution Schedule

A Test Execution Schedule defines the running order for the execution of Test Scripts. ISTQB define it as:

A scheme for the execution of test procedures. The test procedures are included in the test execution schedule in their context and in the order in which they are to be executed.

Again this may not be quite as rigorous a definition as the Test Entity Model requires because, as we will see later it may not just be 'test procedures' that are executed. Certainly it is widely accepted that Test Cases can be executed stand alone.

Entity: Test Result

The ISTQB glossary defines 'Result' as:

> *The consequence/outcome of the execution of a test. It includes outputs to screens, changes to data, reports, and communication messages sent out.*

Developing the Test Entity Model

In developing the Test Entity Model an attempt has been made to describe all *possible* relationships, not just the standard way of arranging test assets. This will help to indicate the advantages of different approaches and perhaps suggest new ones.

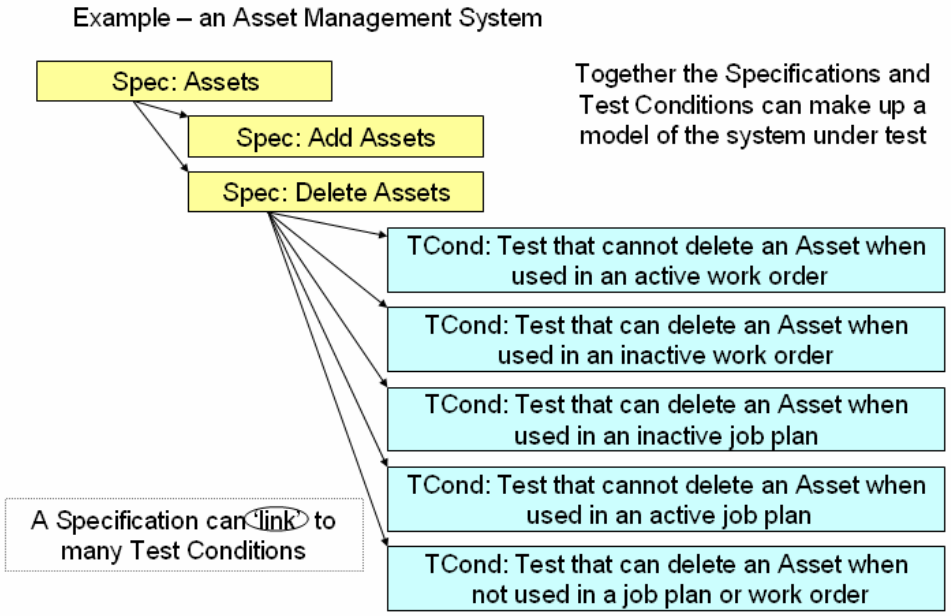
Firstly we examine the relationships individually and gradually build up the model by:

1. Using a possible relationship example
2. Making statements about the relationships

Defining the Relationship: Specifications to Test Conditions

The example below is for an asset management system. It is a simplified version of an actual test project undertaken by the author. There is a hierarchy of specification items – a concept that is very familiar to users of test management tools. At the bottom level of the hierarchy are the individual specification items. In the example below we have a specification item 'Delete Assets'.

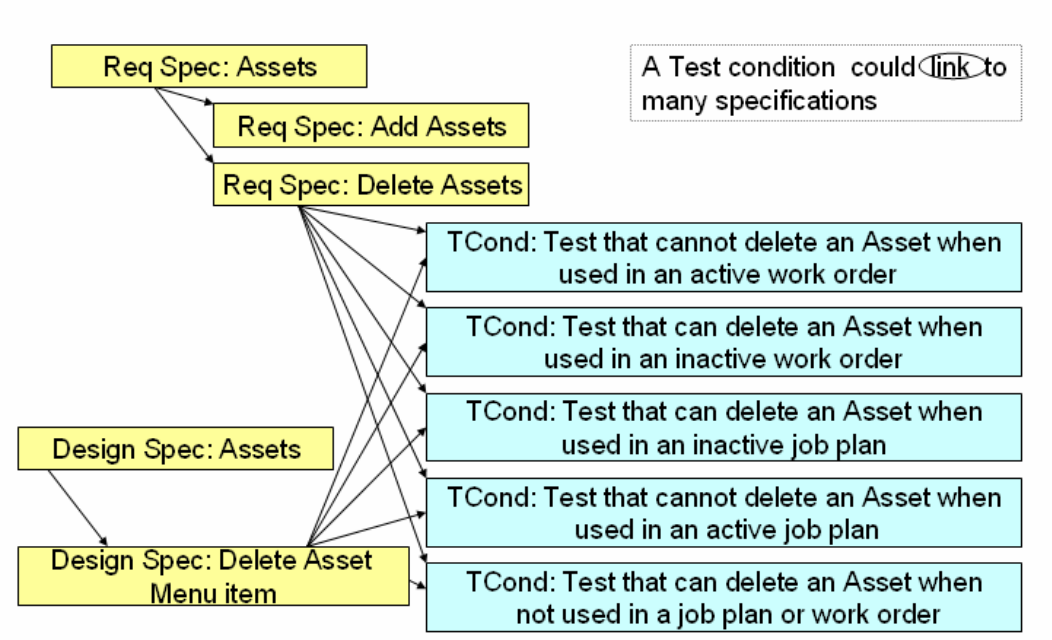
In the requirements specification document the statement was "*The system should allow the user to delete Assets*". On closer inspection of the design, it was discovered that assets could be in different states and should not be deleted for some of these states. Assets could be used in Work Orders and Job Plans and if these were Active the system should not allow the user to delete the asset. This is a typical situation for a test team to fine themselves. The requirements are incomplete and the specification item 'Delete Assets' needs to be refined. *Test Conditions* were created to fully define the system behaviour and allow fuller test coverage of the functionality.



It can be seen that the system behaviour is only fully described by both the specification items and the test conditions. Together both sets of test entities make up a model of the system. Test Conditions then can be considered to be specification items from a testing perspective. However considering the relationships we can make the statement that:

A Specification Item can be 'linked' to many Test Conditions.

This however may not be the full picture – see below:



The Test Condition "*Test that a user cannot delete an Asset when used in an active work order*" may also be linked to a specification item in the design specification – in this case the 'Delete Asset Menu Item' specification. The menu item in this situation should be grayed out. Thus we get the statement:

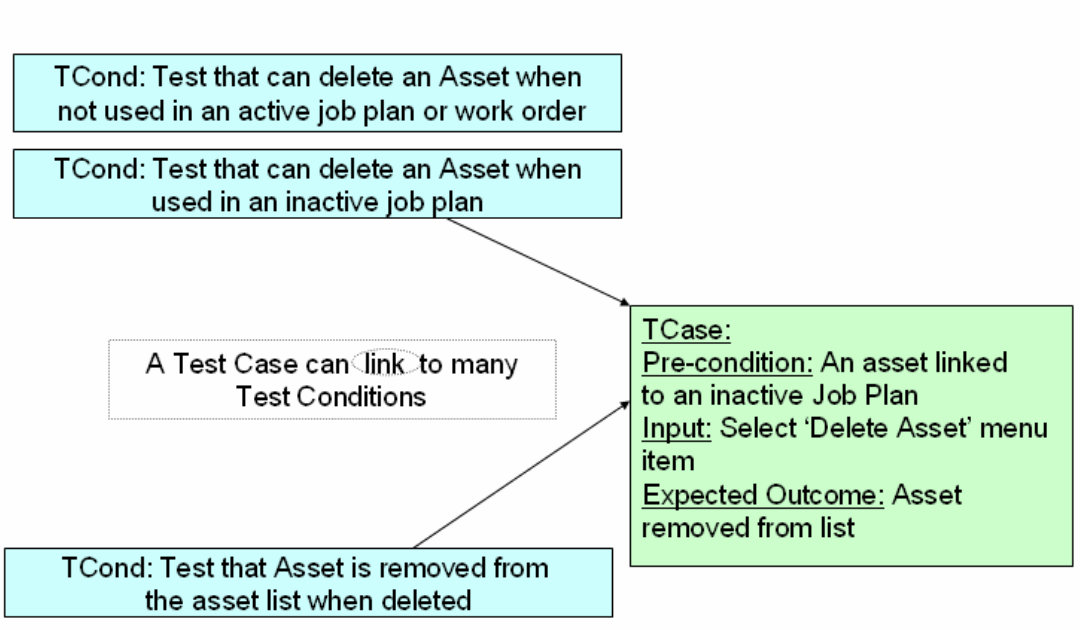
A Test Condition can be 'linked' to many Specification Items.

This relationship certainly can exist (this is dependent upon the actual definition of test condition) but whether it is of practical use and gives an overly complex situation is another thing altogether. However we are interested in all possible entity relationships so it is included in the full model.

Defining the Relationship: Test Conditions to Test Cases

Continuing with our Asset Management System example, it can be seen below that the test conditions are linked to the test cases which provide the information necessary to actually execute the test. A single test case could test many test conditions as can been seen from the example. A test case which deletes an Asset linked to a Job Plan tests not only the test condition *"Test that a user can delete an Asset when used in a Job Plan"* but also tests the test condition derived from the design specification that states that *"An Asset should be removed from the asset list when deleted".* Thus we have the statement:

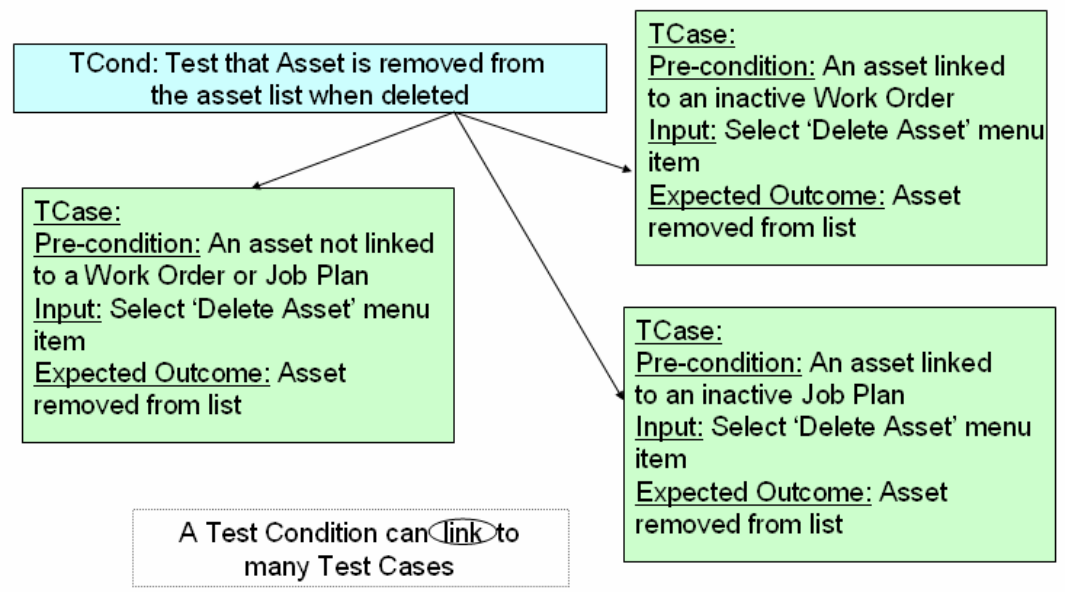
A Test Case can be 'linked' to many Test Conditions.



The *Objective* (IEEE 829) attribute of a Test Case in this definition is the linked test conditions and is really plural as in *Objectives*.

Looking at the relationship the other way around (see below) we can see that the test condition *"An Asset should be removed from the asset list when deleted"* is tested by all the test cases which delete and asset. Thus we get the relationship:

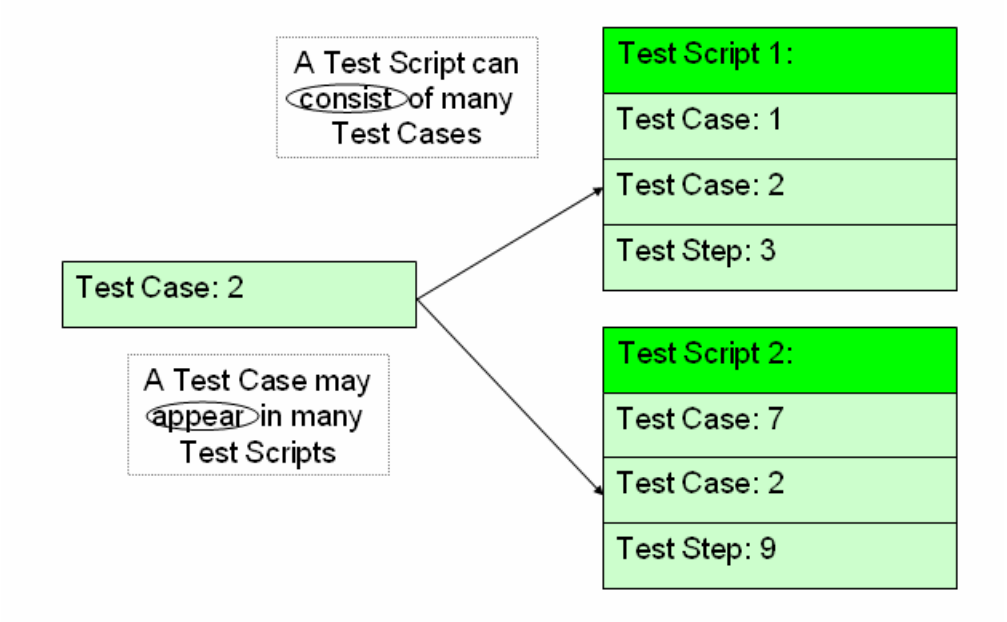
A Test Condition can be 'linked' to many Test Cases.



Defining the Relationship: Test Cases to Test Steps and Scripts

We are in slightly uncharted territory here as a Test Step is not defined in the testing literature but an assumption will be made so that can create the relationships. That assumption is that a Test Step is a Test

Case within a Test Script. This agrees with the common anatomy of a Test Step in that it usually has an *Input* attribute and an *Expected Result* attribute. This certainly holds true in almost all test management tools such as Quality Centre.



From the above diagram we can make both relationship statements:

A Test Script can 'consist of' many Test Cases or Steps.

And

A Test Case can 'appear' in many Test Scripts.

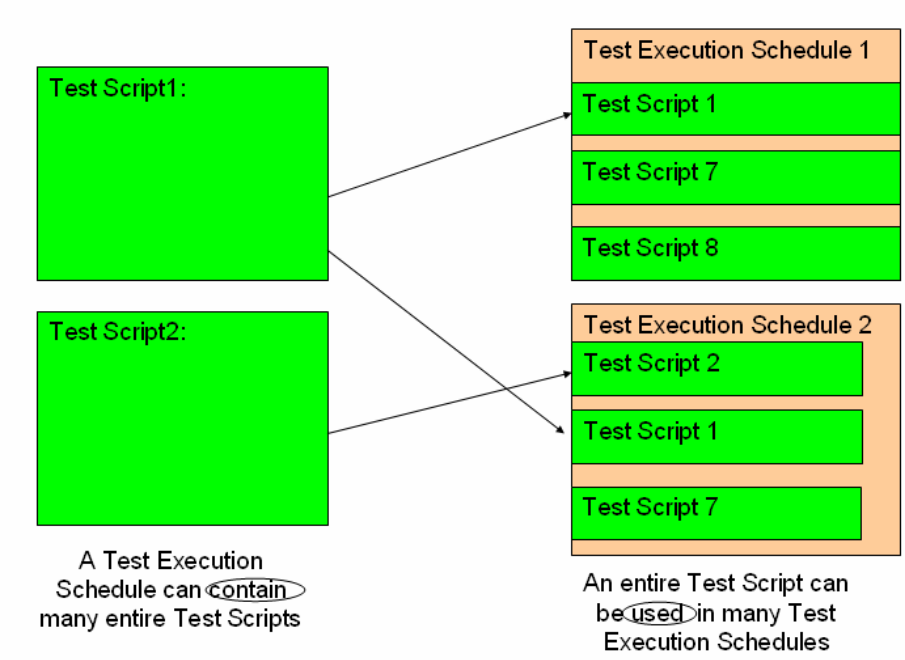
The second statement covers the situation where test cases are re-used.

Note that some test steps are not necessarily test cases. These may be simply setting up the pre-conditions for the next test step.

Also note that we have not been rigorous in defining what the relationships such as *'linked'* and *'consist of'* actually are. The relationships within test management tools (such as Quality Centre) are well defined in their database data structures but these differ slightly between the various tools and are specific to the model of testing each tool vendor has created in the database structures. Fully defining these relationships is a task for the future.

Defining the Relationship: Test Scripts to Test Execution Schedules

In the diagram below are shown the relationships between Test Scripts and Test Execution schedules:



The relationship statements are:

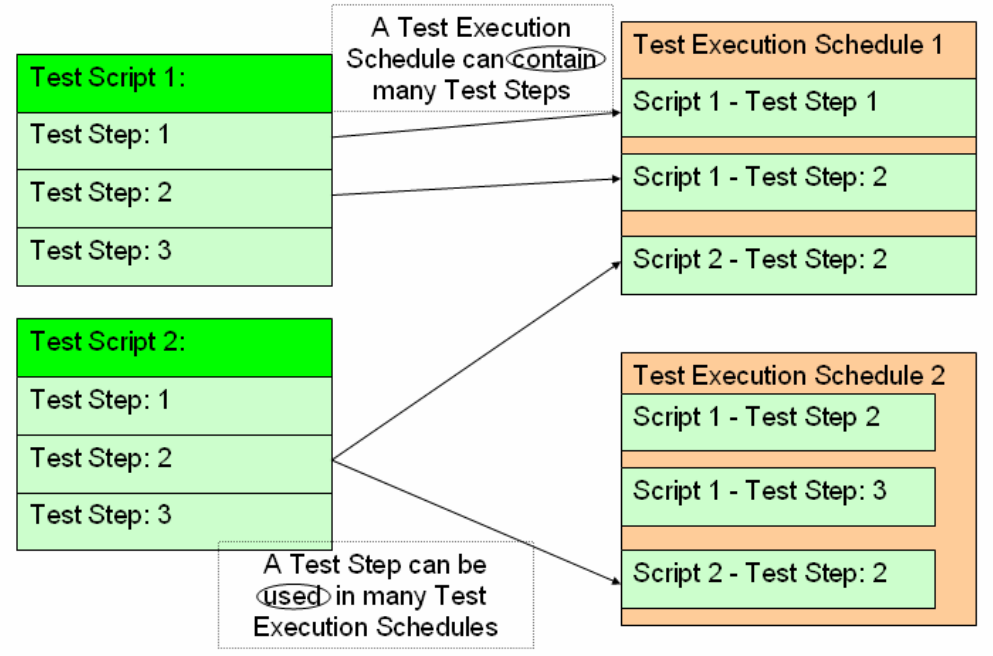
A Test Execution Schedule can 'contain' many Test Scripts.

And

A Test Script can 'appear' in many Test Execution Schedules.

Defining the Relationship: Test Steps to Test Execution Schedules

Although the most common test management tool, Quality Centre will only allow whole test scripts to be scheduled, this is a constraint of the tool and not the ideal world. The diagram below shows that Test Steps could be scheduled individually in a Test Execution Schedule.



From the diagram we get the two statements:

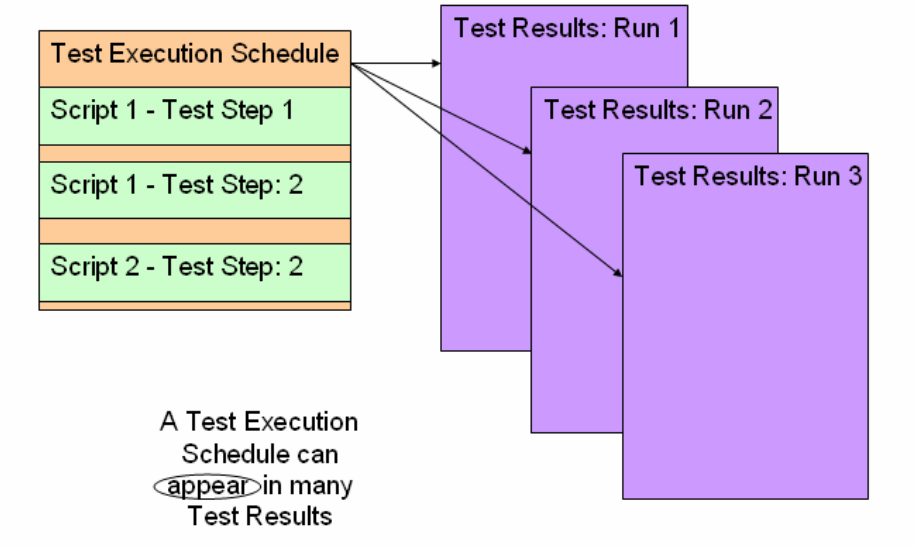
A Test Execution Schedule can 'contain' many Test Steps.

And

A Test Step can 'appear' in many Test Execution Schedules.

Defining the Relationship: Test Execution Schedules to Test Results

Once these sets of tests have been executed, the results are recorded for each run as seen below

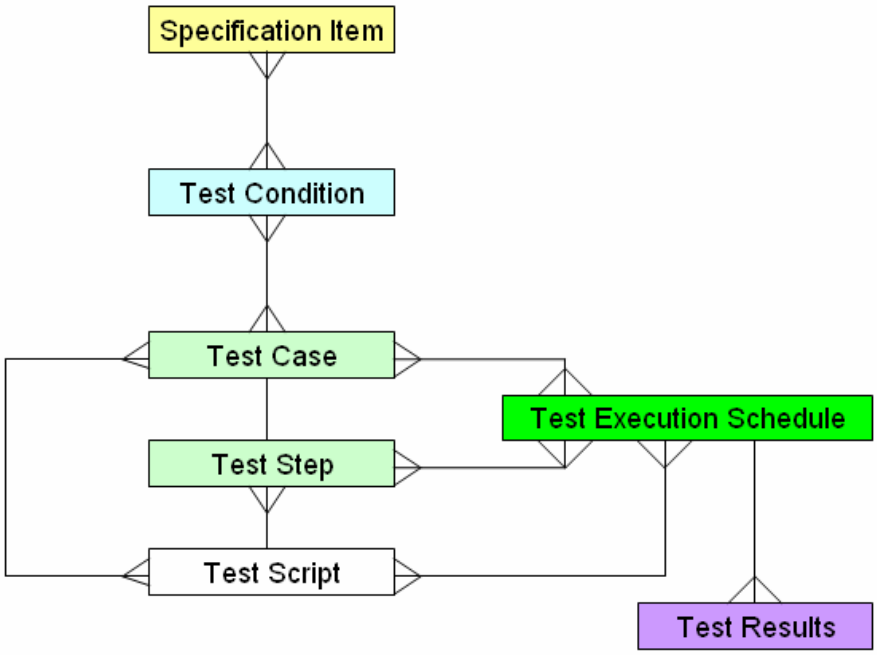


From the diagram we get the statement:

A Test Execution Schedule can 'appear' in many Test Results.

The Test Entity Model

Bringing all of the relationships developed above together we get the following entity relationship diagram:



The relationships should allow for *Requirements Traceability*. This is tractability of:

1. The test coverage of the specification items through to the tests
2. The test results traced back to the specification items

This means that the Specification items can be looked at to see how many tests are used to test it and how many of those test pass and how many fail.

Short comings of the Current Test Entity Model

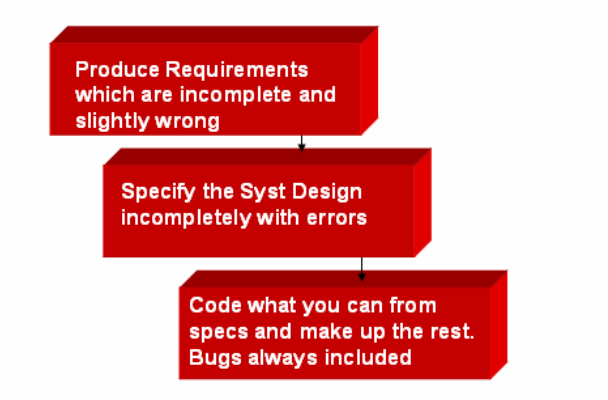
This has not been a rigorous exercise. Shortcomings in this process have been:

1. The entities are not well defined.
2. The meanings of the relationships have not been well defined –e.g. 'linked'.
3. Some relationships are missing:
    - A Test Case could be scheduled without being in a test script
    - Tests could be run (i.e. Test Results generated) without Test Execution Schedules
4. The attributes of the entities have not been defined
5. Some test entities have been left out, most notably Defects, Test Coverage Items and Test Plans

Taking all this into account, we do have a starting point to develop the model and something that can give us insights into the way we test and alternative approaches.

Testing the Model in the Real World

The major difference between theory and Real World is the Real World contains *time* and therefore the work we do is *incomplete*. There is not enough time in the universe to fully test many computer systems and this problem is merely exacerbated by testing being squeezed by project timescales. This time limitation should be a precept in all of our thinking about testing in the real world. More than this, the lack of time holds true for the *development* of the systems we test as well and this has also has a huge influence in shaping the landscape of the testing world. Perhaps the waterfall model should be modified to include this reality as below:



Requirements cannot detail the exact required behavior of a system; system behavior is too complex and there is too little time. Because Requirements are incomplete we add Test Conditions to further describe the system – we produce a list of Test Conditions which are also be incomplete but closer to what is needed to test the system. If the requirements were complete we wouldn't need any test conditions; we could link the test cases straight to requirements.

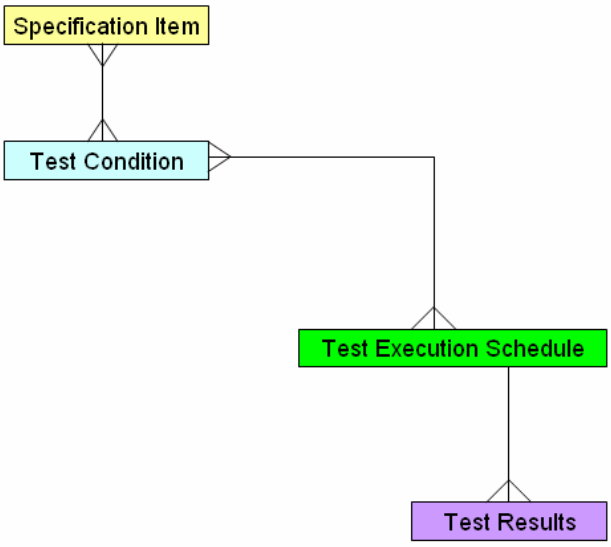
Our testing is also limited because it takes time to build the entities and it takes more time to build the relationships between them even if we use a test management tool to help us.

Executing other Entities Directly

Some entities, however take more time to create than others. It takes more time to write test scripts than it does to build the test conditions which they test so why do we script? Scripts provide important advantages:

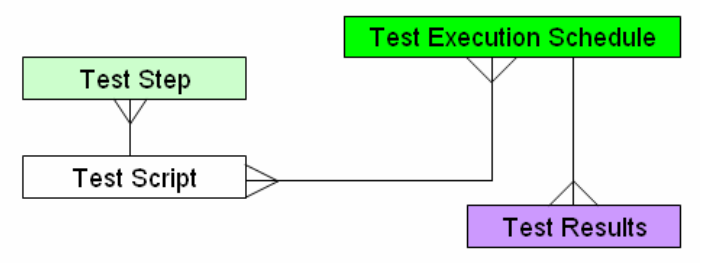
1. They provide a way of maturing data into a state ready for a test using a detailed business process. In our asset management example, we may have a script which adds an Asset to a Work Order and then de-activates the Work Order ready to delete the Asset. Each activity would be a separate test step.
2. They provide instructions for testers who are not familiar with the system or the business activities implemented by the system.

There are however situations where neither of these points are real benefits. Taking the second point first; if the tester has enough systems and business knowledge, they do not need detailed instructions on how to perform a test. As for the first point, the author can think of several systems he has tested where each test could be executed without requiring maturing of the data with preceding test steps. In these situations the test conditions could be executed directly without scripting as shown below:

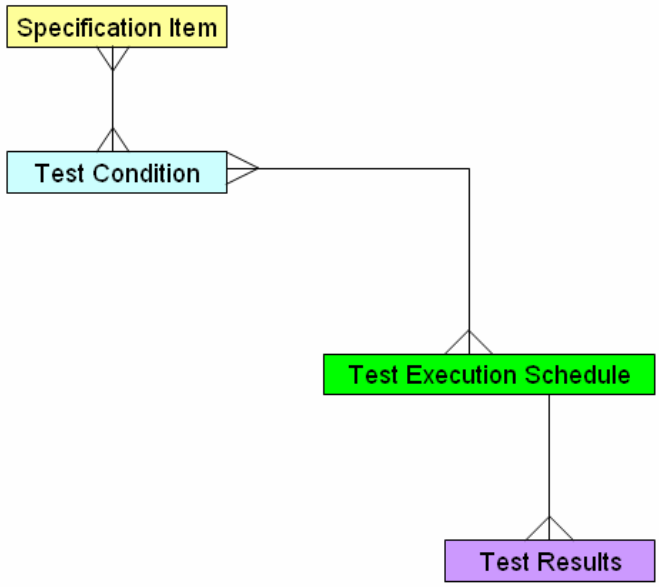


Another reason testers use scripts is simply because many think that is what they should do in order to test. There is an over-emphasis on test scripts because it is generally accepted that testers write scripts. Even the most popular test management tool we use, Quality Centre is built around scripting and you cannot execute a test *unless* it is a script. Creating a Test Condition however, takes much less time than writing a Test Script. Often we could get greater test coverage by producing more test conditions, gaining greater test coverage, and then executing them directly.

To summarise this is what many test teams do:



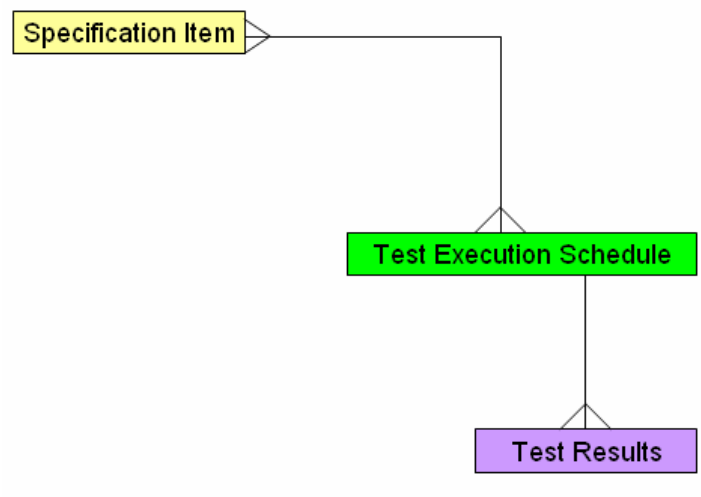
But they could get greater test coverage by using their time more efficiently by doing this:



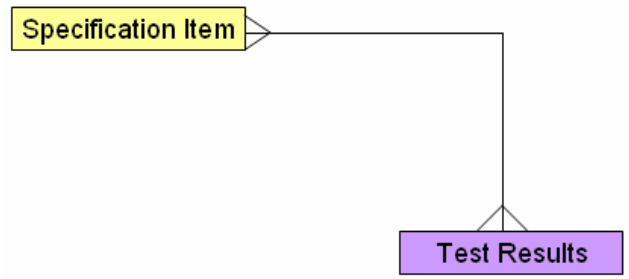
As Test Conditions take less time to create than test scripts, the test team gets more test coverage for the same amount of time and money. They could then execute the test conditions directly. This is of course provided each test condition did not have to be preceded by actions and also that the testers had enough knowledge of the system to execute the tests.

Exploratory Testing's Entity Model

Exploratory Testing also does not follow the Test Entity Model completely either and looks more like the diagram below:



Exploratory Testing may even look like this:



Specifications of some sort must exist, even if only in the heads of the testers. If there is no specification of the system then exploratory testing can only be exploration, not testing. The testers execute the specified behavior directly without a script or test conditions. This takes advantage of the tester's ability to create tests intuitively without the need for formalized test design techniques such as Boundary Value Analysis or Equivalence Partitioning.

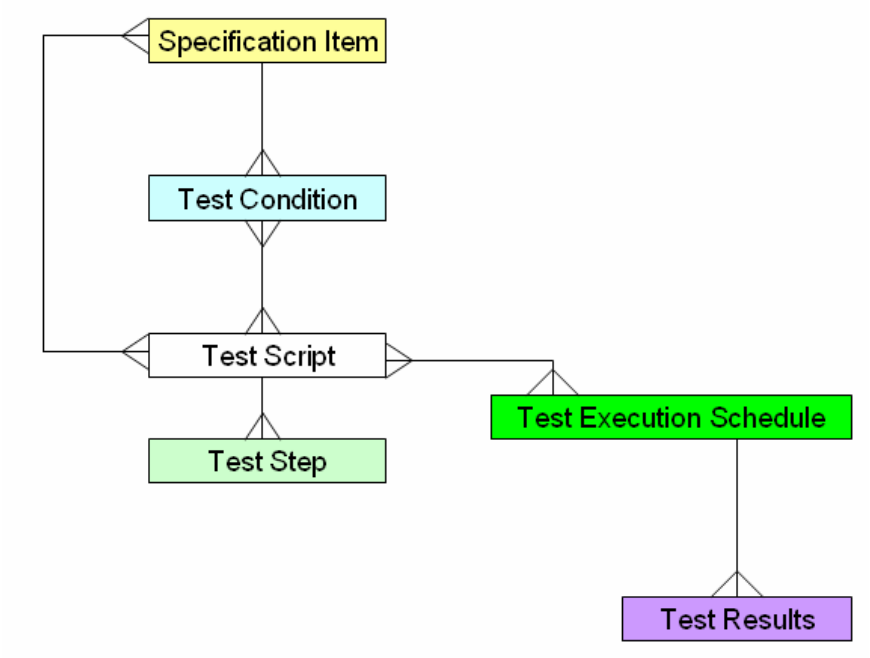
Quality Centre's Entity Model

Quality Centre (formerly Test Director) dominates the test management tool market. It has a database at the back-end and is therefore constrained by what is feasible to implement in a data model. It has all of the entities we have been discussing; however, it is based heavily around scripting.

See the diagram below for Quality Centre's entity model. It differs from the entity model in several ways:

1. It introduces an additional relationship to the Entity Model: Test Conditions to Test Scripts. Test Conditions cannot be linked to Test Steps, only to whole Test Scripts.
2. Only entire Test Scripts can be scheduled in a Test Execution Schedule.
3. Test Scripts can be linked to Specification Items directly not necessarily via Test Conditions.

The first two points are limitations in the way the tool allows testers to work. A Test Script does consist of Test Steps but all external relationships with other entities are at the Test Script level.

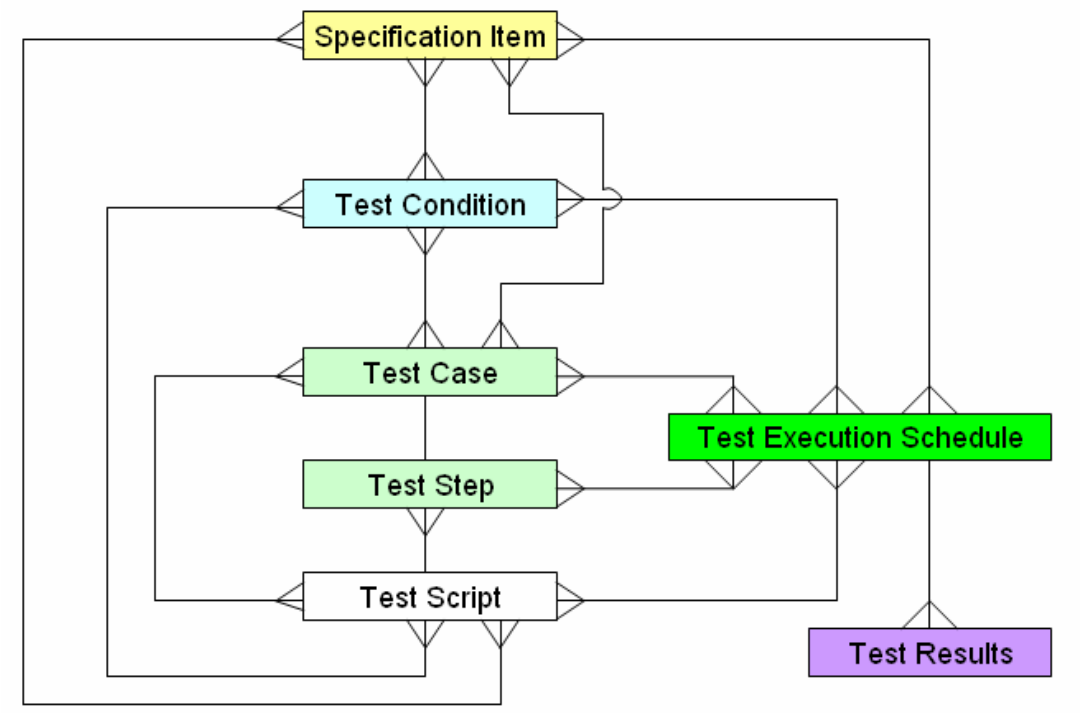


Additions to the Test Entity Model

The new relationships discussed above are:

1. Test Conditions to Test Scripts
2. Specifications Items to Test Cases (or Steps)

Also we could link Specifications Items to Test Scripts. If we add these to the Test Entity Model we get:



This looks rather overly complex but it should not be seen as a single approach, rather it can be used to highlight the advantages and disadvantages of different approaches.

Further Work on the Model

The test entity model is by no means complete. Further work is required to:

- Define all possible relationships between the entities
- Define the attributes of the entities (e.g. a test case has input, pre-conditions and expected outcome attributes)
- Include other entities such as:
    - Defects
    - Test Plans
    - Test Coverage Items

In doing this we may arrive at complete definitions for each of the testing entities – something which has proven to be difficult for those defining glossaries (such as the ISTQB), perhaps because there was no definitive entity model.

Conclusions

The model of testing in the current testing literature, standards and glossaries is not fully defined. The definition of the test entities leaves much to be desired and there is poor definition of the relationships between entities. The entities cannot, however be defined without an entity relationship model because part of the definition of an entity is its relationships with other entities.

The model seeks to be a full description of all possible relationships between entities. In the real world we are likely to use simplified versions of the Test Entity Model and consider the impact of time upon test development and execution. This paper has offered a start at building a full entity model which describes all possible ways of using test entities. The model may lead test organisations into more efficient ways of testing. For example, currently in most test organisations, scripting dominates approaches to System Test and UAT. This may not necessarily the most efficient way to organise the test assets because of time constraints. More streamlined approaches are offered by the model such as executing Test Conditions and Specification Items directly.

References

1. IEE 829-1998: *IEEE Standard for Software Test Documentation*

2. BSI 7925: BSI Standard for Software Component Testing

3. ISEB sylabus

4. ISTQB glossary

5. IEE 829-1998: *IEEE Standard for Software Test Documentation,* Introduction, page iv

6. *Holistic Test Analysis & Design: Paper delivered to SoftTest Ireland Sept 2007, Neil Thompson and Mike Smith*