Using EA

Test Management in Enterprise Architect

by Dermot O’Bryan

Table of Contents

Introduction

There are many benefits to managing tests within the modeling environment. This whitepaper describes how to use Enterprise Architect to create Test Definitions and how to relate these definitions to other artifacts within the development process.

The general topics covered in this paper are as follows:

- The first section Test Tools covers the core functionality including:
    - Creating and viewing tests along with importing internal repository data into Test Cases.
    - Using Enterprise Architect's Build, Test, Run and Debug facilities for defining and executing tests during implementation.
    - Creating, storing and linking test plans and documentation.
- The second section Managing Tests in the Modeling Environment covers methods for using the Test tools in different configurations. These include:
    - Setting different layouts for storing test data
    - Connecting elements for traceability between Test Cases to implementation
    - Reporting Test Case information
    - Searching on Test Results
    - Using profiles for adding custom attributes to elements

Test Tools

Enterprise Architect supports three core functions related to testing. These can be combined and used in a variety of ways. These features are:

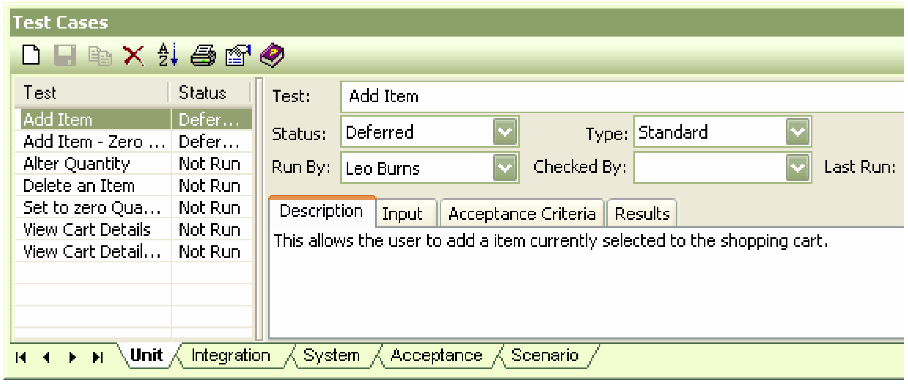
- Test case definitions using either:
    - Test Cases under any Element see: Element Test Cases.
    - A specific Element Type, see: Test Case Element.
- Build, Run, Debug and Unit Testing. This covers building xUnit tests using MDA transforms, generating tests results from these, as well as generating sequence diagrams when running the code.
- Test Plans - Attaching Test Documents This outlines the documentation and reporting of Test Plans using Linked documents and Document Artifact Elements.

Test Case Definitions:

Element Test cases

Multiple Test Cases can be assigned to any Element (from Requirements through Classes, to Nodes and Components).

Element Test Cases are defined using the Test Cases window, available from the menu: View | Testing (Alt+ 3), shown in Figure 1.


Test Cases

Test	Status
Add Item	Defer…
Add Item - Zero …	Defer…
Alter Quantity	Not Run
Delete an Item	Not Run
Set to zero Qua…	Not Run
View Cart Details	Not Run
View Cart Detail…	Not Run

Test: Add Item
Status: Deferred
Type: Standard
Run By: Leo Burns
Checked By:
Last Run:

Description | Input | Acceptance Criteria | Results

This allows the user to add a item currently selected to the shopping cart.

Unit | Integration | System | Acceptance | Scenario

Figure 1 - Test Case Window with Edit Pane.

This window can be viewed in two modes:

a) Show properties mode:
The fields are immediately editable in the right pane.

b) Hide Properties mode:
Provides a summary view of test cases. Editing is performed in an external window.

To swap between these modes use the Show/Hide Properties button:

Test Case Fields

The Test Cases window allows for any number of test cases to be defined and grouped under: **Unit, Integration, System, Acceptance** and **Scenario** tests. These groupings are selected using the tab along the base of the Test Case window (see Figure 1 above).

The Test Cases window has two panes. On the left is a short listing of test cases and on the right are fields for entering and editing data.

The following table contains descriptions of the fields in the editable section of the test cases window:

Control	**Description**	**Defined by**
Test	The name of the test.	
Status	The current status of test (passed, failed...).	
Type	The type of test.	user definable *
Run By	Test run by (person).	user definable **
Checked By	Test run checked by (person).	user definable **
Last Run	The date test last run.	
Description	A description of the test.	
Input	Definition of the data to be input.	
Acceptance Criteria	Definition of the acceptance conditions.	
Results	A listing of the results of last test.	
Defined Tests	List of defined tests associated with this element.	

* *Entries for the Type field are user definable from the menu: Settings | Maintenance | Testing*

*** Entries for Run By and Checked By are user definable, see: Settings | People | Resources*

Setting a Diagram to Show Test Scripts

All diagrams have an option to show test cases defined within the elements. To display a listing of the test definitions in a diagram's elements, use any of the following:

- Select F5 | Elements | Show Compartments
- From the main menu select: Diagram | Properties | Elements | Show Compartments
- Right-click on the diagram and from the context menu select: Properties | Elements | Show Compartments

It is common for tests to be defined against a set of classes. However the tests are best displayed within a diagram specific to testing. This diagram can be located away from the package containing the main diagram (displaying the Attributes and Operations).

Figure 2, below is an example of the Class diagram with Attributes and Operations showing. Figure 3 is an example of a test diagram set to show the Test cases for these same elements.

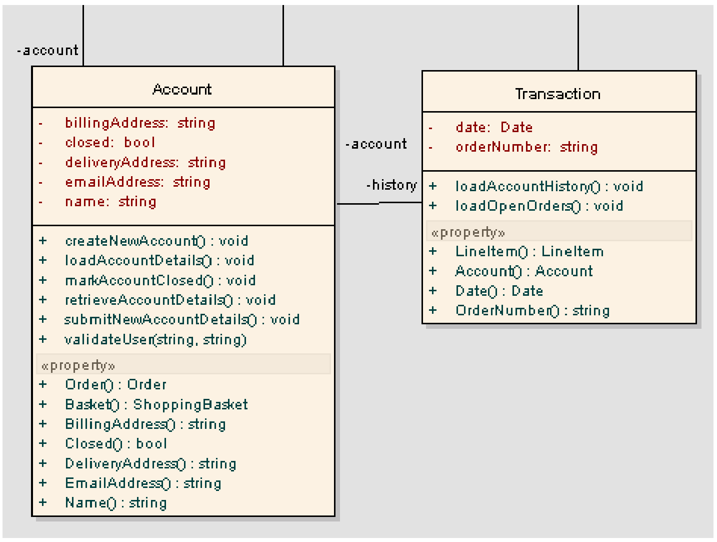

-account
Account
- billingAddress: string
- closed: bool
- deliveryAddress: string
- emailAddress: string
- name: string
+ createNewAccount() : void
+ loadAccountDetails() : void
+ markAccountClosed() : void
+ retrieveAccountDetails() : void
+ submitNewAccountDetails() : void
+ validateUser(string, string)
«property»
+ Order() : Order
+ Basket() : ShoppingBasket
+ BillingAddress() : string
+ Closed() : bool
+ DeliveryAddress() : string
+ EmailAddress() : string
+ Name() : string
-account
-history
Transaction
- date: Date
- orderNumber: string
+ loadAccountHistory() : void
+ loadOpenOrders() : void
«property»
+ LineItem() : LineItem
+ Account() : Account
+ Date() : Date
+ OrderNumber() : string

Figure 2 is an example of a standard Class diagram with Attributes and Operations showing.

Figure 2 - Original Class Diagram

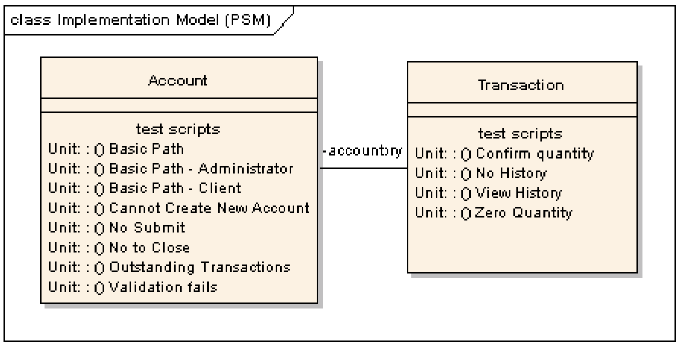

class Implementation Model (PSM)
Account
test scripts
Unit: : () Basic Path
Unit: : () Basic Path - Administrator
Unit: : () Basic Path - Client
Unit: : () Cannot Create New Account
Unit: : () No Submit
Unit: : () No to Close
Unit: : () Outstanding Transactions
Unit: : () Validation fails
-account
-history
Transaction
test scripts
Unit: : () Confirm quantity
Unit: : () No History
Unit: : () View History
Unit: : () Zero Quantity

Figure 3: is an example of a diagram containing the same Classes as are in the diagram above, but with the diagram properties set as:
- [✓]Tests
- [ ] Attributes and Operations

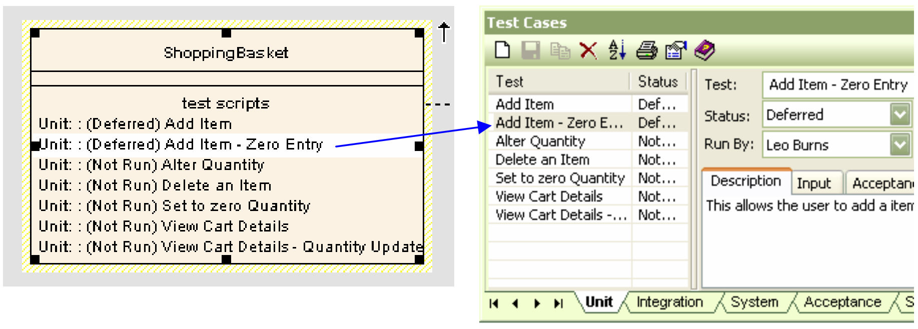
Figure 3 - A Diagram with Test View set and Atttributes and Operation unset.

In-diagram editing of Tests

With Test Cases viewable in a diagram, any Test Case can be selected for editing. This is done by:

1. Selecting the Test item using the mouse
2. Double-clicking on the item selected.

The details of the test case can then be directly accessed. If the Test Case window is not currently open, it will to open the details of the selected test item. In Figure 4 below, the Test Case selected in the Element is highlighted and opened in the Test Case window:



ShoppingBasket

test scripts
Unit: : (Deferred) Add Item
Unit: : (Deferred) Add Item - Zero Entry
Unit: : (Not Run) Alter Quantity
Unit: : (Not Run) Delete an Item
Unit: : (Not Run) Set to zero Quantity
Unit: : (Not Run) View Cart Details
Unit: : (Not Run) View Cart Details - Quantity Update

Test Cases

Test	Status
Add Item	Def...
Add Item - Zero E...	Def...
Alter Quantity	Not...
Delete an Item	Not...
Set to zero Quantity	Not...
View Cart Details	Not...
View Cart Details -...	Not...

Test: Add Item - Zero Entry
Status: Deferred
Run By: Leo Burns

Description | Input | Acceptan

This allows the user to add a item

Unit | Integration | System | Acceptance | S

Figure 4 - Test case selection in an Element.

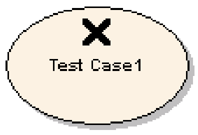
Import from Scenario, Constraints, Requirements and Other Tests

The test cases window also supports importing data from other elements. This includes importing from Scenarios, Constraints, Internal Requirements and Test Cases.

For more detailed information on using this see the appendix: Import from Scenario, Constraints, Requirements and Other Tests

Test Case Elements

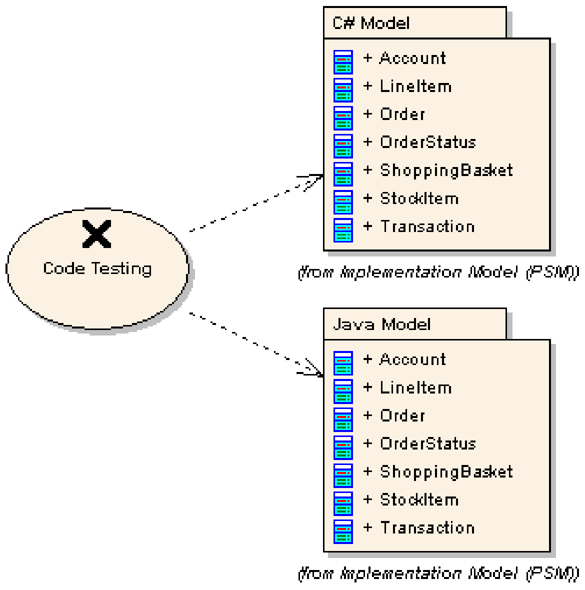
Enterprise Architect supports a custom element-type called "Test Case". This is accessible from the Toolbox under Custom. Below is an example Test Case Element:



The Test Case element is useful:

- When a test case needs to be viewed in a diagram alongside other Elements
- For defining tests that are common to multiple elements
- For containing testing policy or procedures defined in an attached document

These test elements are related to other UML elements using standard UML connectors. A good example is a Package containing Classes defined for two different platforms, but with both sharing a single outline for testing:



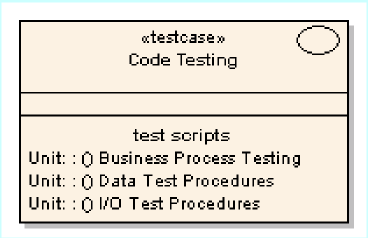
In this example, Code Testing is applicable to both packages. Therefore, related test cases are defined under one Test Case element and reused. This approach avoids duplication and ensures consistency should test case definitions change.

Figure 5 - Test Case Element

In the above case, the general testing process is defined in one document and applied to both platforms. This can be included in the model in the form of a Linked Document in the Test Case Element, or an Element of type Document Artifact. For details on including a linked Document with Testing Procedures see “Linked Documents and Document Artifacts” below.

Note: Test Case elements may contain internal tests viewable via the Testing window (Alt-3).

To view tests on a diagram for non-rectangular element shapes, such as Test Case elements, first set rectangle notation: right-click on the element and select from the context menu: Advanced | Use Rectangle Notation. Then follow the procedure described under: Setting a Diagram to Show Test Scripts.


«testcase»
Code Testing
test scripts
Unit: : () Business Process Testing
Unit: : () Data Test Procedures
Unit: : () I/O Test Procedures

Build, Run, Debug & Unit Testing

Enterprise Architect can interface with the Build, Run and Debug facilities of the major third party development platforms. This feature includes the ability to:

- Build Tests: Use MDA Transforms for creating NUnit & JUnit class stubs.
- Record Unit Test results against the Unit Test classes.
- Immediate build, test and run from EA: Set up package Build Scripts for Builds, Unit Tests, Running and Debugging.
- Testing & Visualization: Using EA's debug workbench facilities
- Profiling and visualization: Generating Sequence diagrams from run-time debugging.

The Build Script feature maintains the runtime components of a package. Here you configure how a package is built (compiled), assign any debugger, configure tests and detail how the package should be deployed.

The Build, Test & Run options are defined in the main menu option: Project | Build and Run:

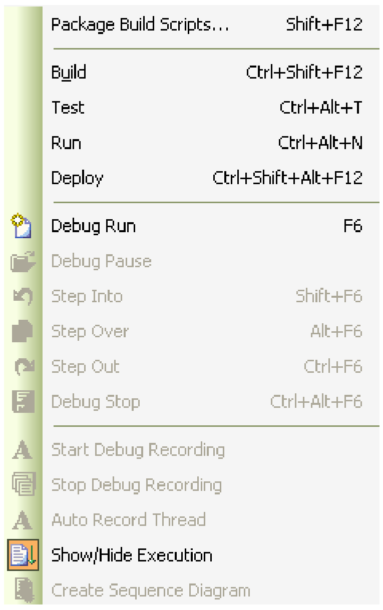


Figure 6 - Project - Build and Run Submenu

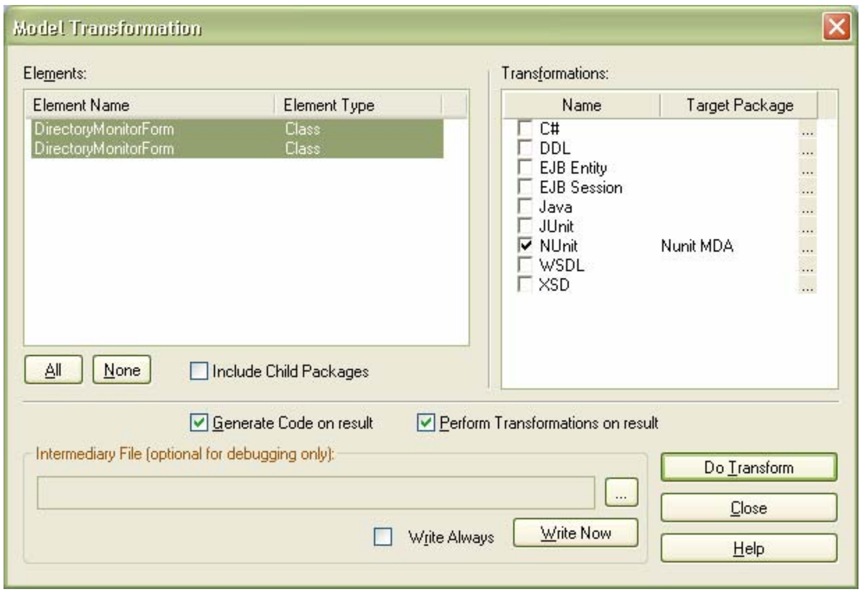
The Build Scripts, Builds, Test & Run entries in this menu are also directly available in the Debug Workbench space. When interacting with the code this can give a more rapid access to update this data. For general information on this, search on the help-index for: "Debug and Profile"

The following is a brief description (and Help-references) of the testing features associated with Build, Test & Run.

MDA Transform NUnit or JUnit test Classes

Where NUnit or JUnit is used for testing code, the Unit Test Classes can be created manually or using MDA transforms to automate test case generation as part of a repeatable process.

For a simple generation of class structures using an MDA transform of the existing Class structure; select Project | Transformations | Transform Current Package:


Model Transformation

Elements:

Element Name	Element Type
DirectoryMonitorForm	Class
DirectoryMonitorForm	Class

Transformations:

	Name	Target Package
☐	C#	
☐	DDL	
☐	EJB Entity	
☐	EJB Session	
☐	Java	
☐	JUnit	
✓	NUnit	Nunit MDA
☐	WSDL	
☐	XSD	

☐ Include Child Packages
✓ Generate Code on result
✓ Perform Transformations on result
Intermediary File (optional for debugging only):
☐ Write Always

Figure 7 - MDA Generation of NUnit Testing

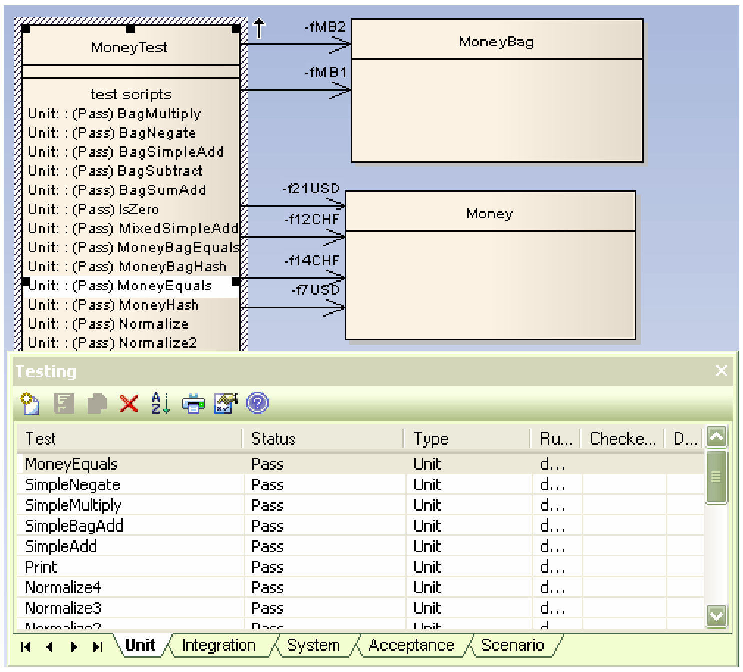
For more information on using MDA transforms, from the help-index; search for the following topics: "Built-in Transformations" "NUnit" "JUnit" and "Unit Testing"

Tracing Compiler errors to the Modeling environment

Using Enterprise Architect to initiate the compilation process allows for capturing compiler warnings and errors in the Output view (Ctr-Shift+F8). These results can be copied from the Output view for further processing. See the help index on: "Build Commands", for further information.

xUnit Test Results Recorded in Test Cases

After inputting code for testing in the xUnit operations then running the compiled xUnit class, Test Cases for each of the operations are added to the Class Element. These record the timing and status of these test operations. They are viewable in the Test Case Window or in a diagram with the diagram-properties set to: [✓] Show Tests;



Test	Status	Type	Ru...	Checke...	D...
MoneyEquals	Pass	Unit	d...		
SimpleNegate	Pass	Unit	d...		
SimpleMultiply	Pass	Unit	d...		
SimpleBagAdd	Pass	Unit	d...		
SimpleAdd	Pass	Unit	d...		
Print	Pass	Unit	d...		
Normalize4	Pass	Unit	d...		
Normalize3	Pass	Unit	d...		

Figure 8: On execution of xUnit tests in EA, the test results are automatically used to generate test cases in the Unit classes.

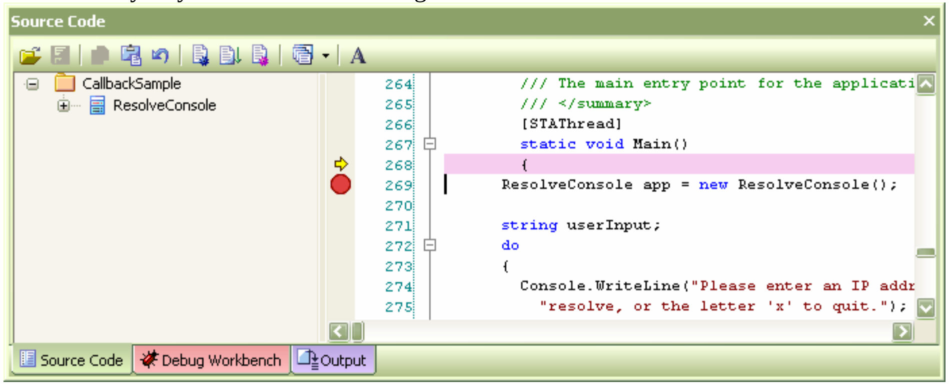
This image shows a diagram with the Diagram-Properties set to show Test Cases. These test cases are also shown below, via the Testing Window.

Figure 8 - Test Cases Generated by xUnit Tests

For more information, see the Enterprise Architect Help Index: “Unit Testing”

Testing & Visualization – using Debug Workbench

Using the Enterprise Architect Debug Workbench, you can walk through the execution of code to verify a system’s behavior for a given test scenario.

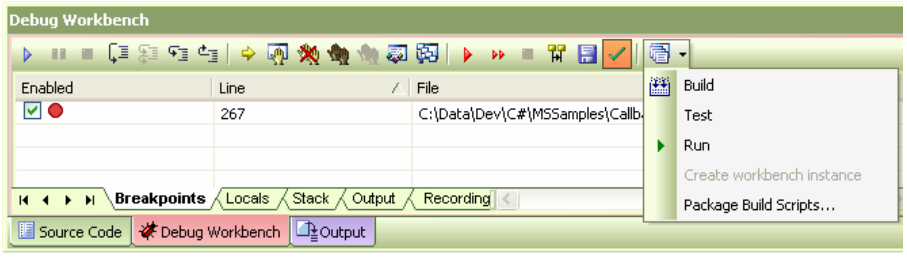


```
        /// The main entry point for the applicati
        /// </summary>
        [STAThread]
        static void Main()
        {
      ResolveConsole app = new ResolveConsole();

      string userInput;
      do
      {
        Console.WriteLine("Please enter an IP addr
          "resolve, or the letter 'x' to quit.");
```

Figure 9 - Source Code View with Current Line of Execution and a Break Point set

Enterprise Architect not only allows you to set up break points in the code, view local variables and stack trace details, but it also allows you to visualize these details by capturing the execution as a Sequence diagram. The Debug Workbench works in conjunction with Enterprise Architect’s Source Code view to do this.



Enabled	Line	File
✓ ●	267	C:\Data\Dev\C#\MSSamples\Callb

Figure 10 - Debug Workbench

For more information see the Enterprise Architect Help Index: "Profiling and Debugging".

Sequence Diagrams of Code Execution

Sequence diagrams can be generated when executing code from the debugger. There are two methods available: Selecting the Main() operation in a class and from the context menu generate the Sequence Diagram, or by setting up break points in the code and generating the diagram between two break points.

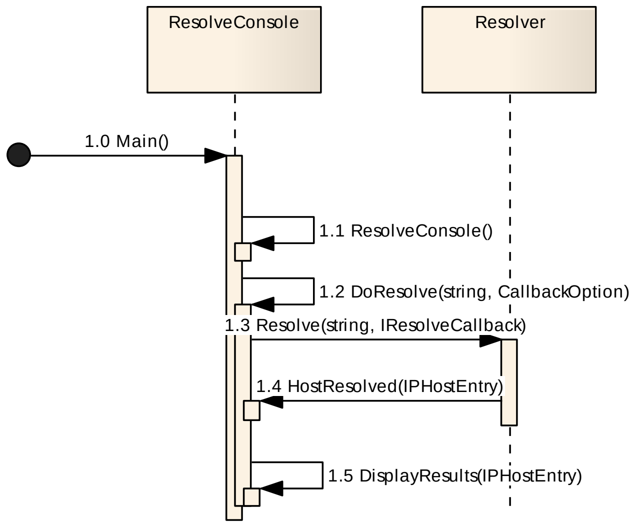

ResolveConsole
Resolver
1.0 Main()
1.1 ResolveConsole()
1.2 DoResolve(string, CallbackOption)
1.3 Resolve(string, IResolveCallback)
1.4 HostResolved(IPHostEntry)
1.5 DisplayResults(IPHostEntry)

Figure 11 - Example Sequence Diagram Generated from Execution of Code.

For more information on generating Sequence Diagrams - see help: "Generate Sequence Diagrams".

Test Plans – Attaching Test Documents

There are often numerous documents associated with the testing process. Enterprise Architect provides a number of methods for attaching test related documents to Elements. These are as follows:

- Using links to external documents.

- Internally stored RTF Documents. In Enterprise Architect there are two types of internally stored RTF documents:
  - Linked Documents
  - Document Artifacts Element.

External Documents

Any Enterprise Architect element can contain a set of external file references using the File Tab under the Element Properties dialog. Only the file path is stored for such references, not the content, which means any file format may be referenced ("Launched"). Similarly, generated reports only contain the meta-information for these references, such as file path and description, not the file content itself.

Linked Documents

RTF documents can be added to any element as internally stored documents (Linked Documents). These are accessible by selecting an element and using either:

- Right –click | Linked Document
- Ctrl-Alt-D

This will open the RTF Editor with an option to start from a template or a blank document. There is a default set of templates available. You can define (or import) your own company templates in the resources section: Resources View | Templates | Linked Document Templates.

Document Artifacts Elements

Document Artifacts Elements are accessible from:

- Toolbox | Deployment | Document Artifact.

On creation of one of these elements, the Element Properties window is displayed by default. On closing this, double-clicking on the element will open the RTF Editor.

RTF Editor

The RTF editor provides for full editing of RTF documents. All options are available using the context menu in the editable region. Below is a view of the context menu selection:

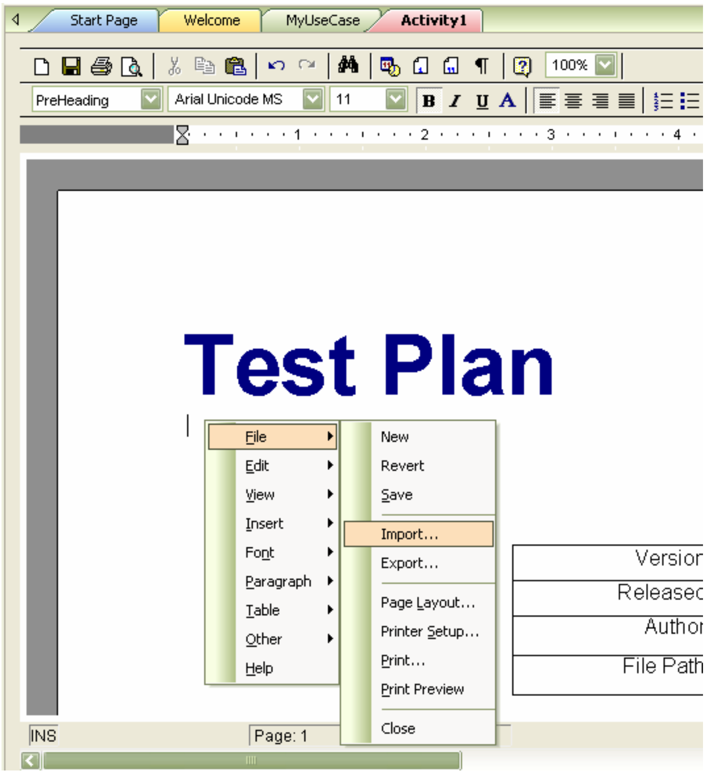


Figure 12: The RTF editor provides a full menu accessible by right-clicking anywhere in the document text area.

Figure 12 - Internal RTF Editor Menu

To import a document into the RTF editor use the context menu option: File | Import. For example, you can import an 'RTF saved' copy from an external document, such as a .doc file.

Outputting RTF Documents

When in the RTF editor; these documents can be output directly by selecting: File | Print from the context menu.

They can also be output using the RTF report generator by including in the RTF report-template the following Sections:

- Element::Model Document
- Package::Package Element::Model Document.

Below is a view of the RTF editors Sections area, with the Element::Model Document section selected:

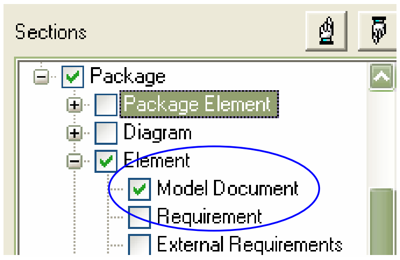


Figure 13: This is the section used in RTF report templates for outputting:

- Element.Linked Documents
- Document Artifacts

Figure 13 - RTF report generator Section for Documents

For more information, see the help section: "Creating Documents | RTF Documents" & "Modeling with Enterprise Architect | Working with Elements | Linked Documents"

Templates

When creating a new document using the "Copy Template" drop-down, there is a default 'Test Plan' template available. User defined templates (i.e. a corporate standard test plan template) can be added in Enterprise Architect's Resources section under:

Resources | Templates | RTF Templates | Linked Document | Templates.

Managing Test Cases in the Modeling Environment

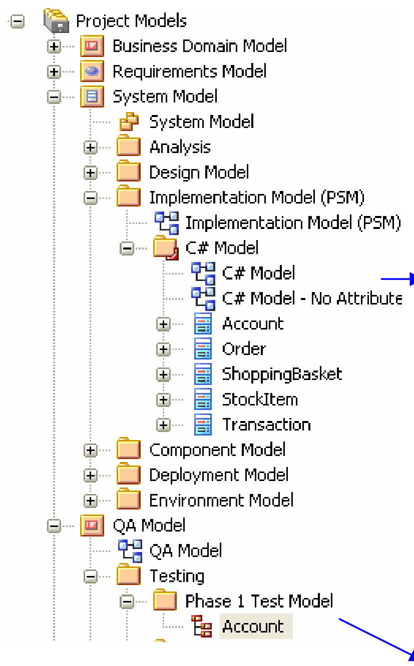
While the approach to modeling test cases will vary according to development methodology, in this section we present some specific approaches that can be readily adapted to suit different methodologies.

Our first approach is most applicable to "white-box testing", where tests are defined according to specific aspects of the system's underlying implementation. The second approach is useful under "black-box testing" conditions, where we test directly against functional system requirements, without regard to implementation details.

In both cases, we give consideration to how the Test cases are organized with respect to the model hierarchy and what implications this has for reporting, traceability and other analysis activities.

White-Box Testing: Implementation Elements Define the Tests

Following is an excerpt from the Enterprise Architect Example model, which shows an example of elements containing tests in two different diagram layouts. Figure 14 shows the standard class diagram format, without test information displayed, while Figure 15 uses the test view diagram format.



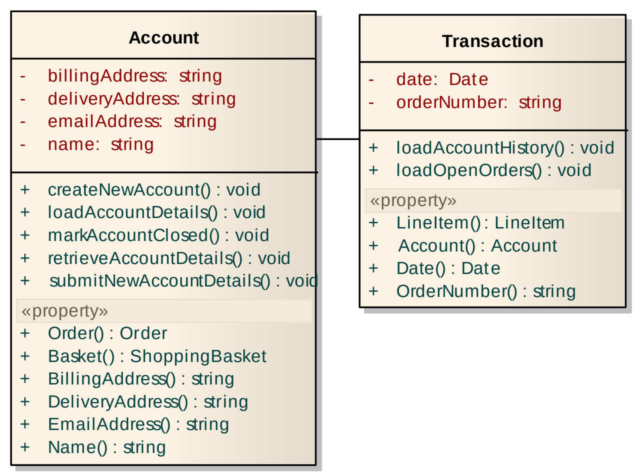

Account
- billingAddress: string
- deliveryAddress: string
- emailAddress: string
- name: string
+ createNewAccount() : void
+ loadAccountDetails() : void
+ markAccountClosed() : void
+ retrieveAccountDetails() : void
+ submitNewAccountDetails() : void
«property»
+ Order() : Order
+ Basket() : ShoppingBasket
+ BillingAddress() : string
+ DeliveryAddress() : string
+ EmailAddress() : string
+ Name() : string
Transaction
- date: Date
- orderNumber: string
+ loadAccountHistory() : void
+ loadOpenOrders() : void
«property»
+ LineItem() : LineItem
+ Account() : Account
+ Date() : Date
+ OrderNumber() : string

Figure 14 - Class Diagram – Attributes and Operations Visible.

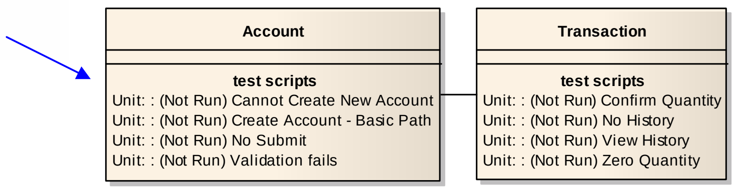

Account
test scripts
Unit: : (Not Run) Cannot Create New Account
Unit: : (Not Run) Create Account - Basic Path
Unit: : (Not Run) No Submit
Unit: : (Not Run) Validation fails
Transaction
test scripts
Unit: : (Not Run) Confirm Quantity
Unit: : (Not Run) No History
Unit: : (Not Run) View History
Unit: : (Not Run) Zero Quantity

Figure 15 - Class Diagram – Test Cases Visible

Defining tests directly against implementation elements, as above, is useful for initial tests on a newly created Class or as a permanent set of ‘regression tests’. It may also be useful when relating tests to elements that specify system behavior, such as Requirement or Use Case elements.

Note: At higher levels of abstraction, the above approach may be equally applied to black-box testing, where Requirement elements for example, contain internal tests.

Black-Box Testing: Using Maintenance Elements to Establish Traceability

During the development and maintenance phases of a system, issues will be reported against specific components and enhancements will be made over time. Enterprise Architect's Issue and Change elements can contain test definitions, and thus provide traceability from a specific maintenance event to the affected system component and its subsequent testing. The following is a simple example, using the Account class (depicted earlier):

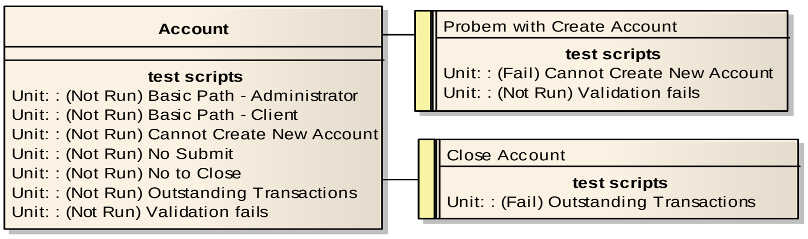
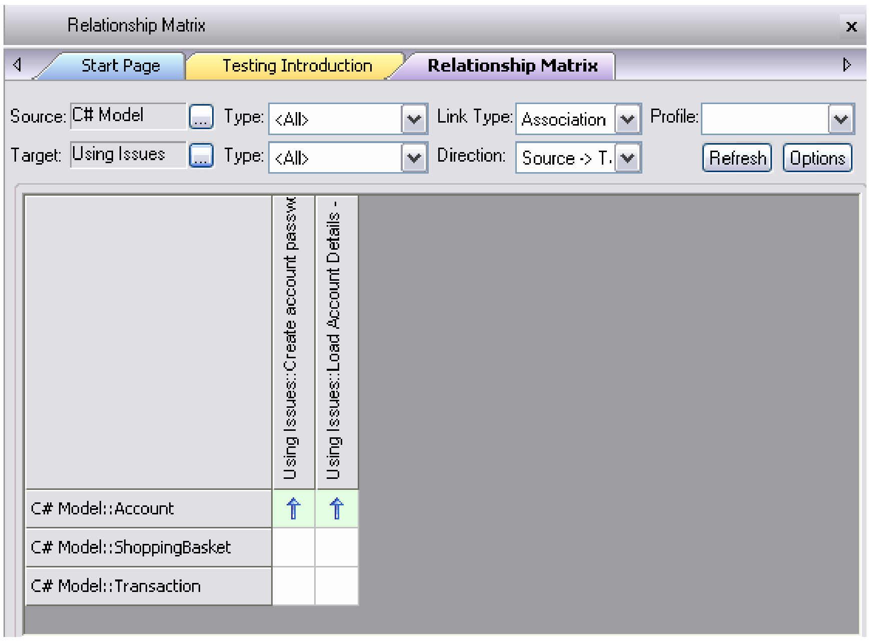

Account
test scripts
Unit: : (Not Run) Basic Path - Administrator
Unit: : (Not Run) Basic Path - Client
Unit: : (Not Run) Cannot Create New Account
Unit: : (Not Run) No Submit
Unit: : (Not Run) No to Close
Unit: : (Not Run) Outstanding Transactions
Unit: : (Not Run) Validation fails
Probem with Create Account
test scripts
Unit: : (Fail) Cannot Create New Account
Unit: : (Not Run) Validation fails
Close Account
test scripts
Unit: : (Fail) Outstanding Transactions

Figure 16 - Class and Related Issues Containing Tests

Where tests are captured using a separate element, a relationship such as a Dependency connector provides direct traceability from a system element to its testing.

Such relationships can be constructed diagrammatically as in the above example, or using the Relationship Matrix as in Figure 17.



Source: C# Model Type: <All> Link Type: Association Profile:
Target: Using Issues Type: <All> Direction: Source -> T

	Using Issues::Create account passw	Using Issues::Load Account Details -
C# Model::Account	⇧	⇧
C# Model::ShoppingBasket		
C# Model::Transaction		

Figure 17 - Association of the Elements containing Test Cases to the Classes.

The Hierarchy View (Ctrl+Shift+4) provides an alternate view of the traceability between the Issue elements and the Class elements.

Organizing Testing by Release

When developing multiple releases it is more preferable to separate the system and test model elements and store the testing analysis in a separate package grouped by iteration such as build or release. These test cases can be related by Package structure, or by Connectors.

Below is a classic example of setting up Issue elements, containing test cases, in a set of release related packages:

Project View **Diagram**

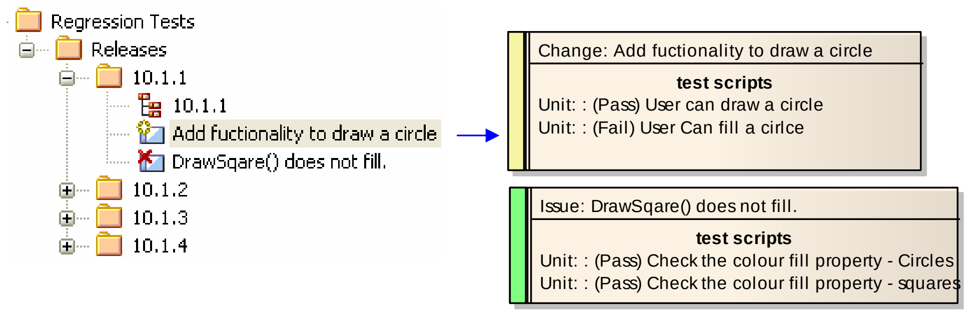

Regression Tests
Releases
10.1.1
10.1.1
Add fuctionality to draw a circle
DrawSqare() does not fill.
10.1.2
10.1.3
10.1.4
Change: Add fuctionality to draw a circle
test scripts
Unit: : (Pass) User can draw a circle
Unit: : (Fail) User Can fill a cirlce
Issue: DrawSqare() does not fill.
test scripts
Unit: : (Pass) Check the colour fill property - Circles
Unit: : (Pass) Check the colour fill property - squares

Figure 18 - Release Specific Packages with Elements containing Test Scripts

Where connectors are required, these can be set up using the relationship matrix (see Figure 17).

There are many different methods that can be used for grouping and arranging correction tasks and testing in a release. They can be grouped by functional areas, by developers etc. These groupings can also vary; they can be by Packages, Swim-lanes and/or Boundaries etc.

Below is an example of grouping by Release/Developer. The diagram uses swim-lanes to define a column for the original issues and a column for the associated corrections.

Project View **Diagram**

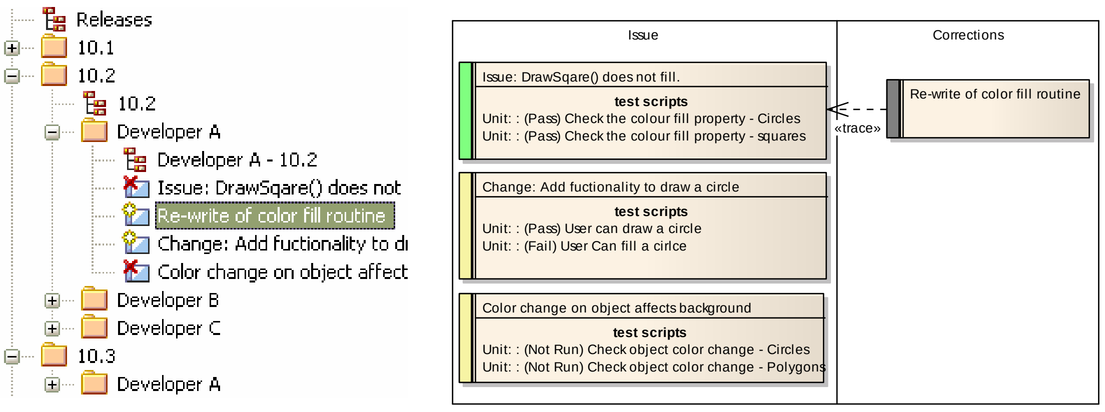

Releases
10.1
10.2
10.2
Developer A
Developer A - 10.2
Issue: DrawSqare() does not
Re-write of color fill routine
Change: Add fuctionality to dı
Color change on object affect
Developer B
Developer C
10.3
Developer A
Issue
Corrections
Issue: DrawSqare() does not fill.
test scripts
Unit: : (Pass) Check the colour fill property - Circles
Unit: : (Pass) Check the colour fill property - squares
«trace»
Re-write of color fill routine
Change: Add fuctionality to draw a circle
test scripts
Unit: : (Pass) User can draw a circle
Unit: : (Fail) User Can fill a cirlce
Color change on object affects background
test scripts
Unit: : (Not Run) Check object color change - Circles
Unit: : (Not Run) Check object color change - Polygons

Figure 19 - Issues with Tests defined by Developer in 'Releases' Packages

An alternative is to group changes into sets based on functional area (Figure 20), but including relationships to the human resources used (e.g. developers and testers etc. – see Figure 21).

Project View **Diagram**

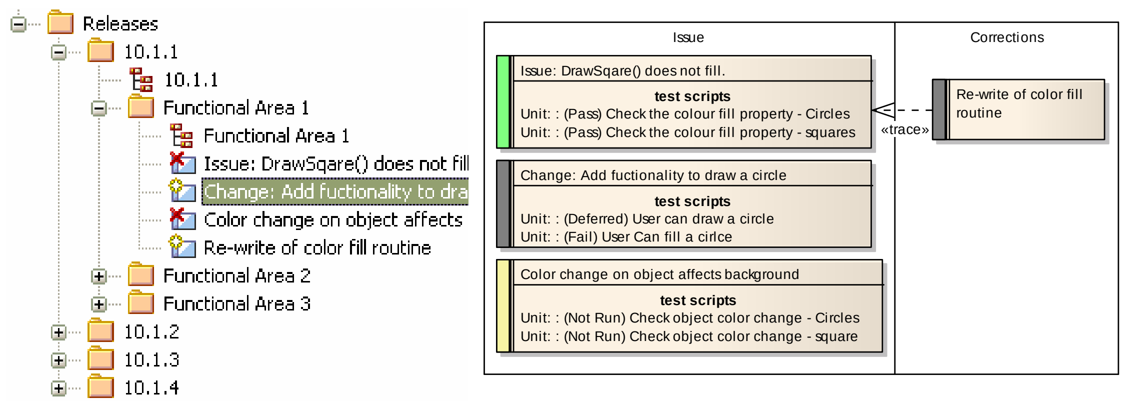

Releases
10.1.1
10.1.1
Functional Area 1
Functional Area 1
Issue: DrawSqare() does not fill
Change: Add fuctionality to dra
Color change on object affects
Re-write of color fill routine
Functional Area 2
Functional Area 3
10.1.2
10.1.3
10.1.4
Issue
Corrections
Issue: DrawSqare() does not fill.
test scripts
Unit: : (Pass) Check the colour fill property - Circles
Unit: : (Pass) Check the colour fill property - squares
«trace»
Re-write of color fill routine
Change: Add fuctionality to draw a circle
test scripts
Unit: : (Deferred) User can draw a circle
Unit: : (Fail) User Can fill a cirlce
Color change on object affects background
test scripts
Unit: : (Not Run) Check object color change - Circles
Unit: : (Not Run) Check object color change - square

Figure 20 - Grouping Issues by functional area

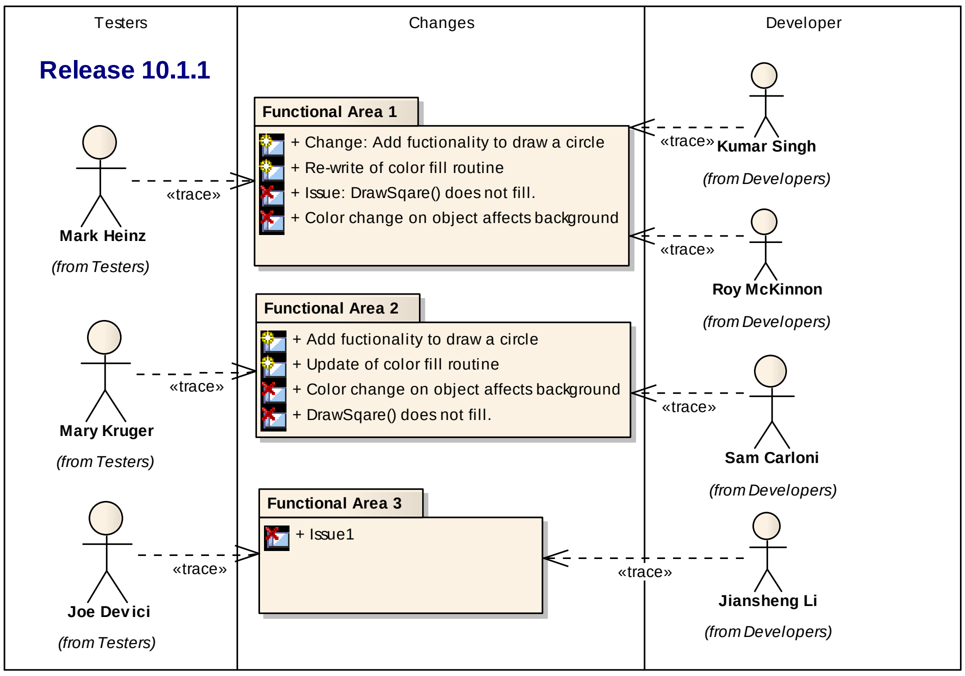


Figure 21 - Diagram of Testers and Developers Assigned to Packages

Typically relationships to testers and developers can be defined in the main diagram of the release.

Another alternative means of linking back to the code modified is to use partitions that are 'instances' of the classes or the components being tested. These can contain the Issue elements as children:

Project View **Diagram**

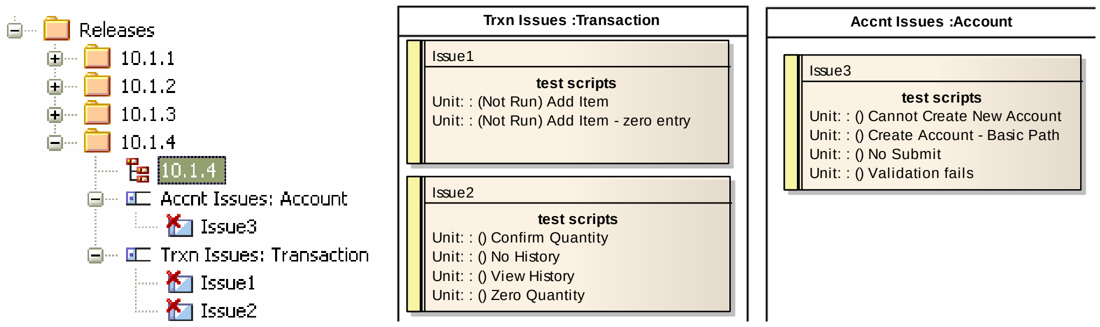


Figure 22 - Using Partitions as 'Instances' of classes to contain Issues

Hint: The partitions can be set as instances of the related class using: Advanced | Instance Classifier, or Ctrl+L.

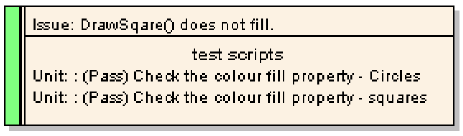
Enterprise Architect offers many other alternatives for using this type of grouping.

Color Coding Elements Based on Test Status

Custom Elements such as Issues, Changes and Requirements may be color coded to enable quick visual cues indicating their status. To enable color coding for Issues and Changes:

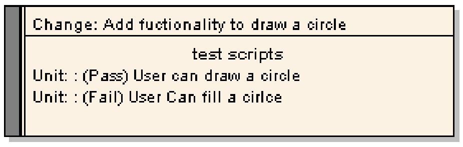
1. From the menu, select Tools | Options | Objects
2. Check the [] Show Status colors on diagrams Checkbox

Once color coding has been enabled, it can be applied by selecting an element and in the elements properties setting the Status. Below are examples of Issues, set from top down, to 'Validated', 'Implements' and 'Proposed':



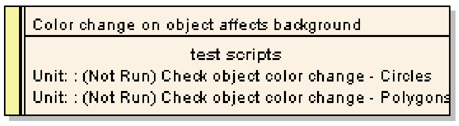
Issue: DrawSqare() does not fill.
test scripts
Unit: : (Pass) Check the colour fill property - Circles
Unit: : (Pass) Check the colour fill property - squares

Issue Status: Validated



Change: Add fuctionality to draw a circle
test scripts
Unit: : (Pass) User can draw a circle
Unit: : (Fail) User Can fill a cirlce

Issue Status: Implemented



Color change on object affects background
test scripts
Unit: : (Not Run) Check object color change - Circles
Unit: : (Not Run) Check object color change - Polygons

Issue Status: Proposed

This gives a clear view of the final status of the set of tests against one Issue. On completion of the testing, all Elements will be displayed with a green (validated) status coloring.

Hint: Status Types and the associated colors can be user-defined using the main menu option: Settings | General Types | Status Types

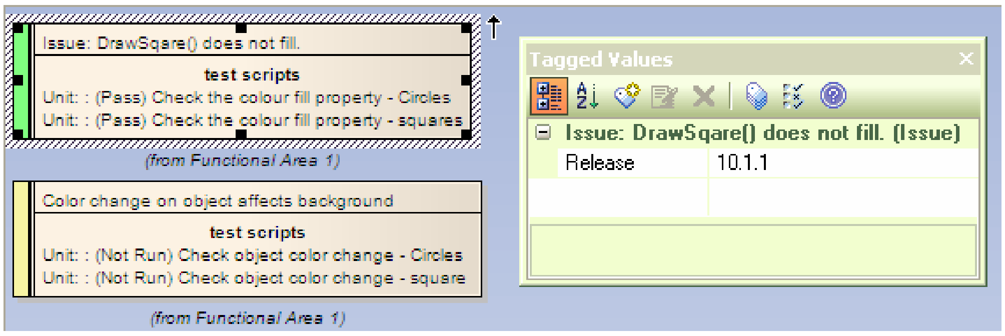
Using Profiles to Add User-defined Testing Fields

Using Tagged Values, you can enter any number of additional attributes such as the Email of a user pointing out a bug, the release affected, etc.

Tagged Values can be defined on a one-off basis for any element, or predefined to be included on creation of a new element.

Tagged value data for an Element is available as a separate window, which is accessed using Ctrl+Shift+6 (or from the main menu View | Tagged Values).

See Figure 23 for a diagram showing a one-off addition of a Tagged Value.


Issue: DrawSqare() does not fill.
test scripts
Unit: : (Pass) Check the colour fill property - Circles
Unit: : (Pass) Check the colour fill property - squares
(from Functional Area 1)

Color change on object affects background
test scripts
Unit: : (Not Run) Check object color change - Circles
Unit: : (Not Run) Check object color change - square
(from Functional Area 1)

Tagged Values
Issue: DrawSqare() does not fill. (Issue)

Release	10.1.1

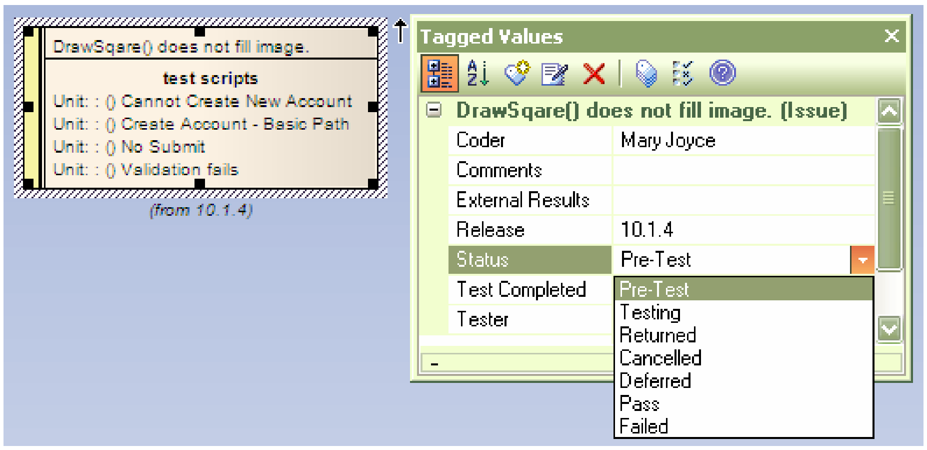
Figure 23 - Issues with the Tagged Value Sheet allowing the Assignment of Attributes

How to Add One-off Tagged Values to Requirements

1. Right-click the Requirement
2. Select Add | Add Tagged Value…
3. Enter the name of the new Attribute. (eg. "Release")
4. Enter the value for the new Attribute. (eg. "10.1.4")
5. Click OK to set the Attribute.

Predefining Tagged Values Types for Test Cases

Any Element in Enterprise Architect, including Issue and Change Elements, can have an extended set of attributes defined for a project. The Element attributes can be predefined using either a *UML Profile* or a *Template*. See Figure 24 for an example of an element using a predefined set of Tagged Values for a project's Issue Elements.


DrawSqare() does not fill image.
test scripts
Unit: : () Cannot Create New Account
Unit: : () Create Account - Basic Path
Unit: : () No Submit
Unit: : () Validation fails
(from 10.1.4)

Tagged Values
DrawSqare() does not fill image. (Issue)

Coder	Mary Joyce
Comments	
External Results	
Release	10.1.4
Status	Pre-Test
Test Completed	
Tester	

Figure 24 - Using Predefined Tagged Values

The predefined Tagged Values types can include a number of standard formats, such as date/time, calendar view, and drop-down lists, etc.

These extended attributes can also be viewed directly on the Element in the diagram. To set this mode for a diagram, right-click on the diagram, in the context menu select: Diagram Properties | Visible Compartments [✓] Tags. Below is the same element in Figure 24, viewed in this mode.

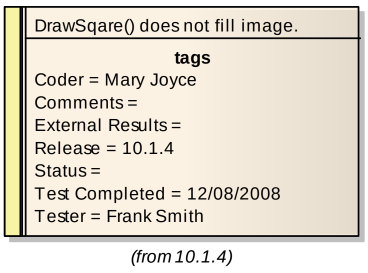


Figure 25 - Tagged Values Visible on Elements.

For more information on extending Issue and Change Element attributes using Tagged Values see the Appendix - Defining Attributes Using a Profile or a Template.

Viewing and Reporting

Aside from opening the Testing Workspace, there are several facilities for viewing the test scripts and reporting on test scripts. These are:

- Reporting using :
  - Standard test reports
  - RTF report generator
  - Search results reports
  - HTML reporting

Built-in Test Reports

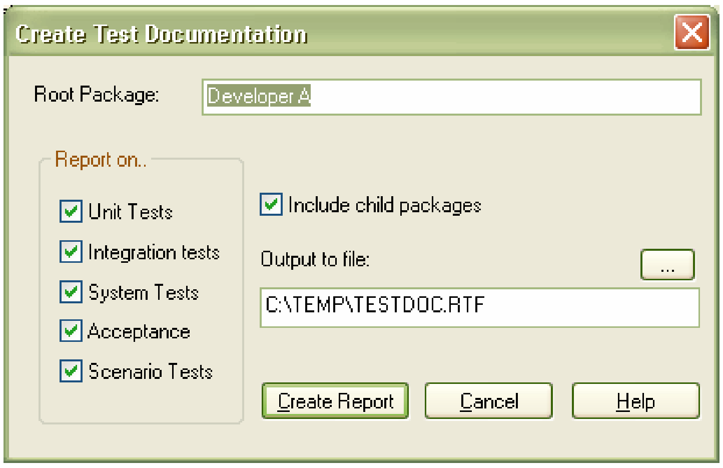
Enterprise Architect supports two standard test reports:

- Testing Report
- Testing Details

Testing Report

The Testing Report allows for filtering on Test Case Type only. To access this report; from the main menu select: Project | Documentation | Testing Report…

Following is the dialog for creating a Testing report;



Create Test Documentation

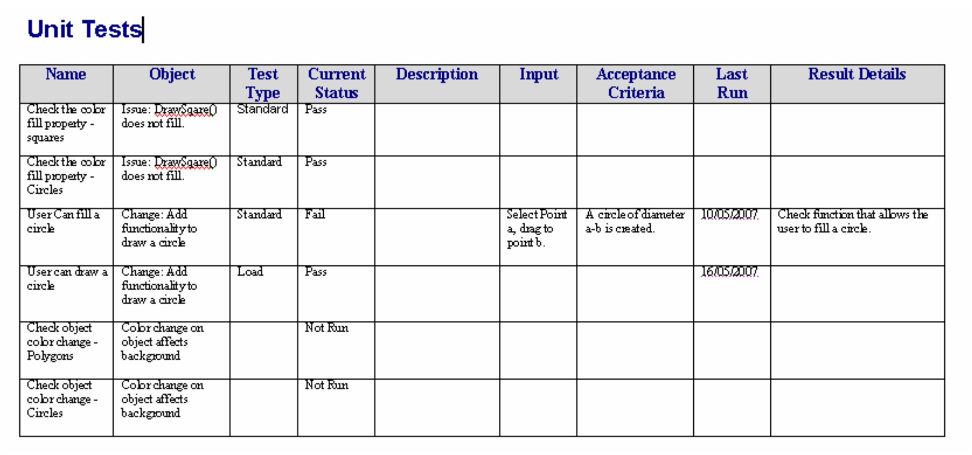
Root Package: Developer A

Report on..
- [x] Unit Tests
- [x] Integration tests
- [x] System Tests
- [x] Acceptance
- [x] Scenario Tests

- [x] Include child packages

Output to file: C:\TEMP\TESTDOC.RTF

This generates documents with a format as follows:



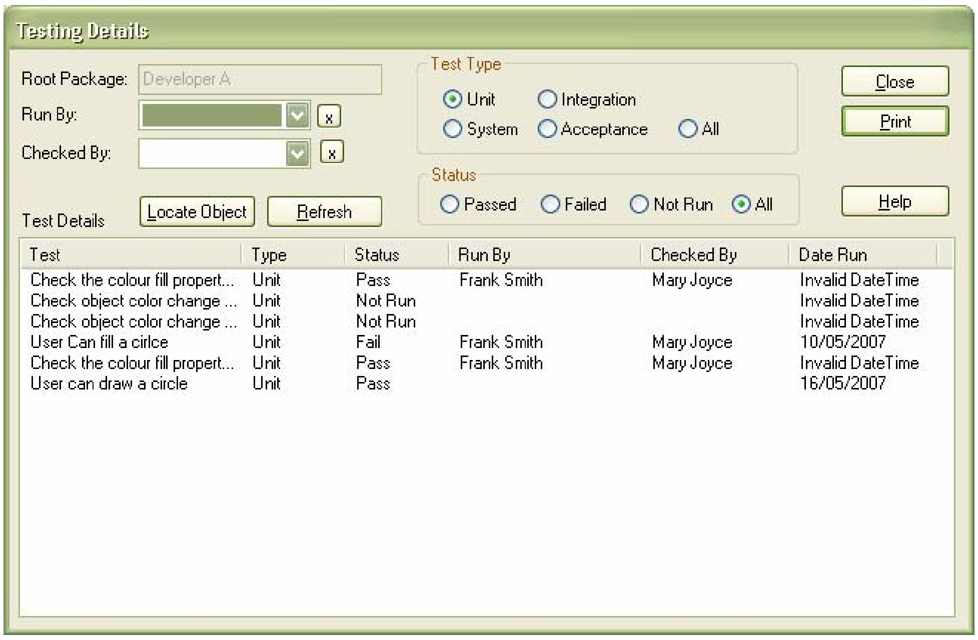
Unit Tests

Name	Object	Test Type	Current Status	Description	Input	Acceptance Criteria	Last Run	Result Details
Check the color fill property - squares	Issue: DrawSqare() does not fill.	Standard	Pass					
Check the color fill property - Circles	Issue: DrawSqare() does not fill.	Standard	Pass					
User Can fill a circle	Change: Add functionality to draw a circle	Standard	Fail		Select Point a, drag to point b.	A circle of diameter a-b is created.	10/05/2007	Check function that allows the user to fill a circle.
User can draw a circle	Change: Add functionality to draw a circle	Load	Pass				16/05/2007	
Check object color change - Polygons	Color change on object affects background		Not Run					
Check object color change - Circles	Color change on object affects background		Not Run					

Testing Details

The Testing Details Report allows for filtering to be set up to narrow the output generated. To access this report; from the main menu select: Project | Documentation | Testing Details…

The following shows the dialog for creating a Testing report along with the filter options available;



Root Package: Developer A
Run By:
Checked By:

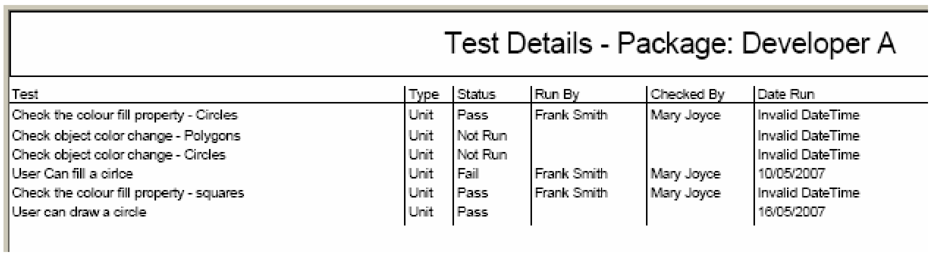
Test Type: [✓ Unit] Integration System Acceptance All

Status: Passed Failed Not Run [✓ All]

Test Details

Test	Type	Status	Run By	Checked By	Date Run
Check the colour fill propert...	Unit	Pass	Frank Smith	Mary Joyce	Invalid DateTime
Check object color change ...	Unit	Not Run			Invalid DateTime
Check object color change ...	Unit	Not Run			Invalid DateTime
User Can fill a cirlce	Unit	Fail	Frank Smith	Mary Joyce	10/05/2007
Check the colour fill propert...	Unit	Pass	Frank Smith	Mary Joyce	Invalid DateTime
User can draw a circle	Unit	Pass			16/05/2007

This generates documents with a simple format as follows:



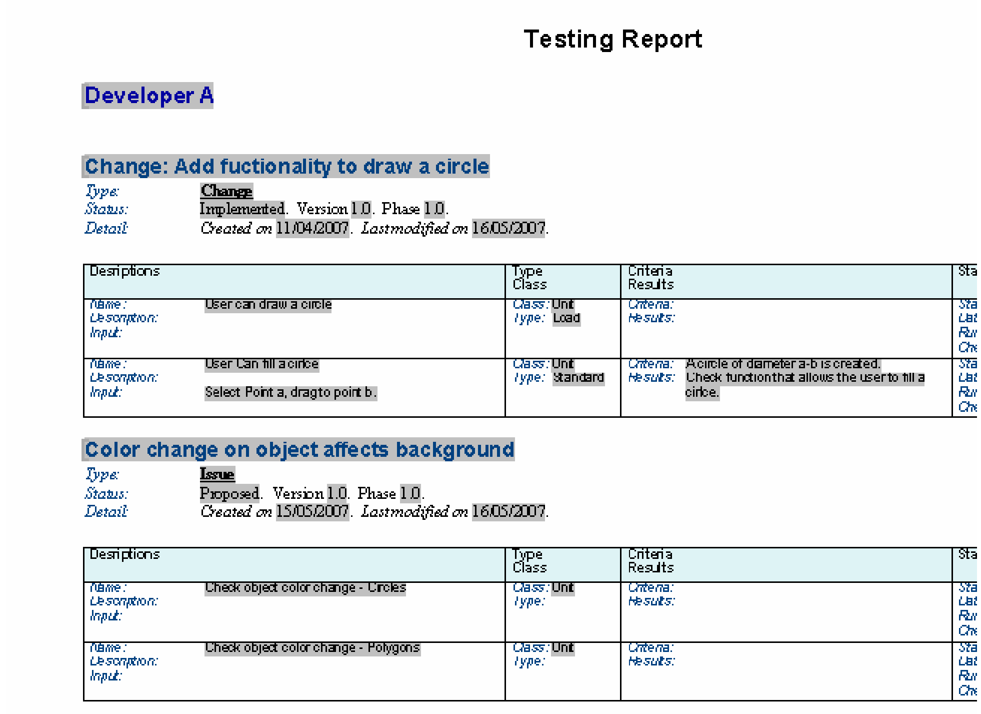
Test Details - Package: Developer A

Test	Type	Status	Run By	Checked By	Date Run
Check the colour fill property - Circles	Unit	Pass	Frank Smith	Mary Joyce	Invalid DateTime
Check object color change - Polygons	Unit	Not Run			Invalid DateTime
Check object color change - Circles	Unit	Not Run			Invalid DateTime
User Can fill a cirlce	Unit	Fail	Frank Smith	Mary Joyce	10/05/2007
Check the colour fill property - squares	Unit	Pass	Frank Smith	Mary Joyce	Invalid DateTime
User can draw a circle	Unit	Pass			16/05/2007

RTF Test Reports

The RTF template report generator includes a report template '{testing template}' that can be used as a starter for creating user-defined test documents.

The RTF report generator is accessible using Project | Documentation | Rich Text Format (RTF) Report or F8. Below is output generated using the default {testing template};



Testing Report

Developer A

Change: Add fuctionality to draw a circle

Type:	**Change**
Status:	Implemented. Version 1.0. Phase 1.0.
Detail:	*Created on* 11/04/2007. *Lastmodified on* 16/05/2007.

Desriptions		Type Class	Criteria Results	Sta
Name: *Description:* *Input:*	User can draw a circle	*Class:* Unit *Type:* Load	*Criteria:* *Results:*	*Sta* *Lat* *Ru* *Ch*
Name: *Description:* *Input:*	User Can fill a cirlce Select Point a, dragto point b.	*Class:* Unit *Type:* Standard	*Criteria:* A circle of diameter a-b is created. *Results:* Check functionthat allows the user to fill a cirlce.	*Sta* *Lat* *Ru* *Ch*

Color change on object affects background

Type:	**Issue**
Status:	Proposed. Version 1.0. Phase 1.0.
Detail:	*Created on* 15/05/2007. *Lastmodified on* 16/05/2007.

Desriptions		Type Class	Criteria Results	Sta
Name: *Description:* *Input:*	Check object color change - Circles	*Class:* Unit *Type:*	*Criteria:* *Results:*	*Sta* *Lat* *Ru* *Ch*
Name: *Description:* *Input:*	Check object color change - Polygons	*Class:* Unit *Type:*	*Criteria:* *Results:*	*Sta* *Lat* *Ru* *Ch*

For more information on defining templates in the RTF format see Help | Creating Documents | RTF Documents or see the RTF document generation whitepaper on: http://www.sparxsystems.com.au/resources/whitepapers/index.html

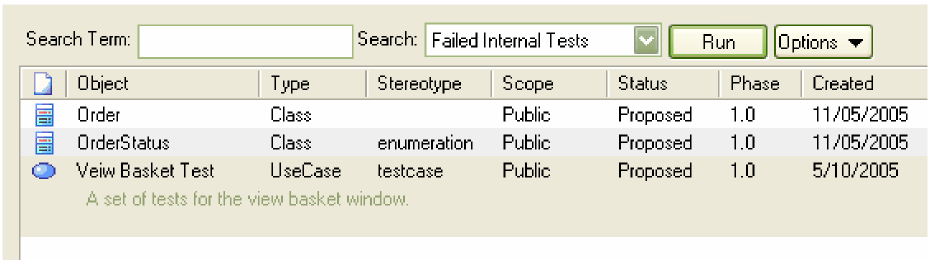
Search Reports

The Enterprise Architect model search facility (Ctrl-F) offers two options for defining search filters. These are by using either the Query Builder or a SQL Search.

Query Builder Searches

The standard search using the Query Builder returns results by Element (e.g. any Element that has one or many failed Tests). The output can be used to generate simple reports.

Of the predefined searches there is a search: *Failed Internal Tests*. This searches for elements containing internal Test Cases where the search term is in any common Test Case field and the Status value is 'Fail'. The following shows the results page after running this search.



Search: Failed Internal Tests

Object	Type	Stereotype	Scope	Status	Phase	Created
Order	Class		Public	Proposed	1.0	11/05/2005
OrderStatus	Class	enumeration	Public	Proposed	1.0	11/05/2005
Veiw Basket Test	UseCase	testcase	Public	Proposed	1.0	5/10/2005

A set of tests for the view basket window.

Figure 26 - List of Elements with Tests Cases that have 'Status' set to 'Fail'.

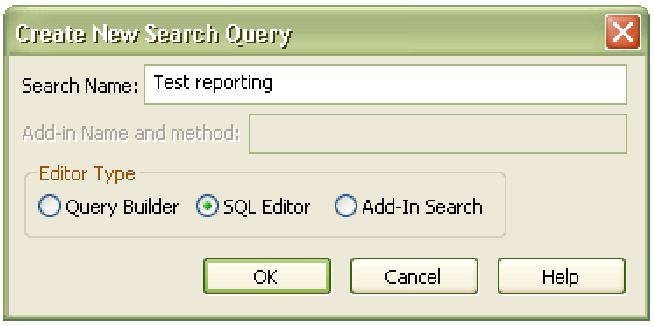
User defined searches can be easily created from within the Advanced Search window.

SQL Editor Searches

For filtering and reporting testing on a Sub-element level, the Search facility has an option that allows users to define sub-element filters (e.g. a set of Element::TestCases). This requires using a SQL statement that acts on the whole model; they cannot be set to search a part of the model hierarchy.

The following example is for a search of Test data:

1. Create an advanced search using: Ctrl-F | Advanced | New Search
2. In the Create New Search Query dialog:
   - In the field: Search Name - type in a name for the search.
   - Select: ⦿ SQL Editor.
   - Press OK.



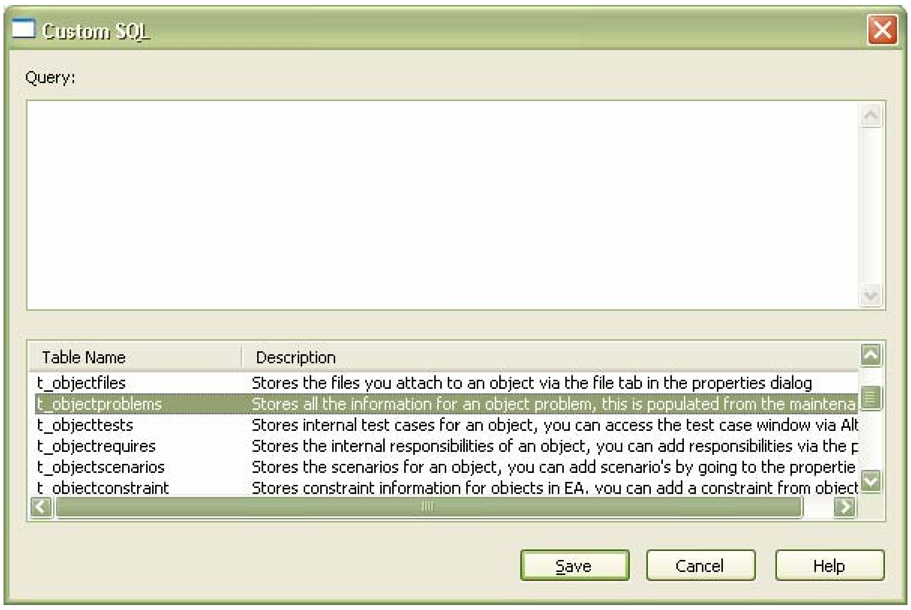
Create New Search Query

Search Name: Test reporting

Add-in Name and method:

Editor Type
Query Builder [✓ SQL Editor] Add-In Search

This will open the SQL dialog box:



Custom SQL

Query:

Table Name	Description
t_objectfiles	Stores the files you attach to an object via the file tab in the properties dialog
t_objectproblems	Stores all the information for an object problem, this is populated from the maintena
t_objecttests	Stores internal test cases for an object, you can access the test case window via Alt
t_objectrequires	Stores the internal responsibilities of an object, you can add responsibilities via the p
t_objectscenarios	Stores the scenarios for an object, you can add scenario's by going to the propertie
t_objectconstraint	Stores constraint information for objects in EA, you can add a constraint from object

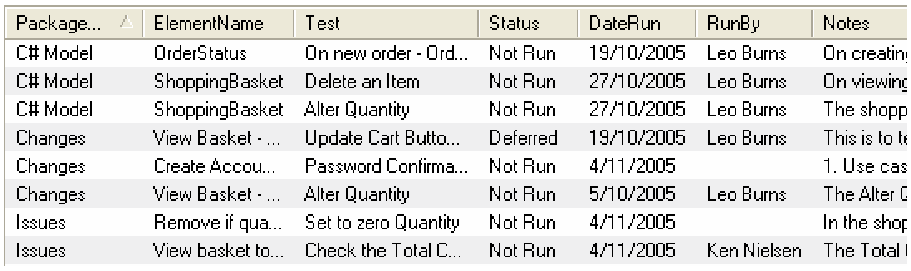
In the Query: text box, add a SQL statement:

- For example: SELECT * FROM t_objecttests where status <> "Pass"
- Press Save
- In the Find In Model dialog press: Run Search

Tip: The following SQL statement will give the Package name, Element name and test details:

```
SELECT t_package.Name AS PackageName, t_object.Name AS
ElementName, t_objecttests.Test, t_objecttests.TestClass,
t_objecttests.TestType, t_objecttests.Status, t_objecttests.DateRun,
t_objecttests.RunBy, t_objecttests.CheckBy, t_objecttests.Notes,
t_package.Package_ID, t_object.Object_ID
FROM (t_package INNER JOIN t_object ON t_package.Package_ID =
t_object.Package_ID) INNER JOIN t_objecttests ON t_object.Object_ID =
t_objecttests.Object_ID;
```

Below is a partial view of the output from the SQL statement in the tip above:



Package...	ElementName	Test	Status	DateRun	RunBy	Notes
C# Model	OrderStatus	On new order - Ord...	Not Run	19/10/2005	Leo Burns	On creatin
C# Model	ShoppingBasket	Delete an Item	Not Run	27/10/2005	Leo Burns	On viewing
C# Model	ShoppingBasket	Alter Quantity	Not Run	27/10/2005	Leo Burns	The shopp
Changes	View Basket - ...	Update Cart Butto...	Deferred	19/10/2005	Leo Burns	This is to te
Changes	Create Accou...	Password Confirma...	Not Run	4/11/2005		1. Use cas
Changes	View Basket - ...	Alter Quantity	Not Run	5/10/2005	Leo Burns	The Alter C
Issues	Remove if qua...	Set to zero Quantity	Not Run	4/11/2005		In the shop
Issues	View basket to...	Check the Total C...	Not Run	4/11/2005	Ken Nielsen	The Total

This can be printed by clicking on the above, and selecting the Print option from the context menu.

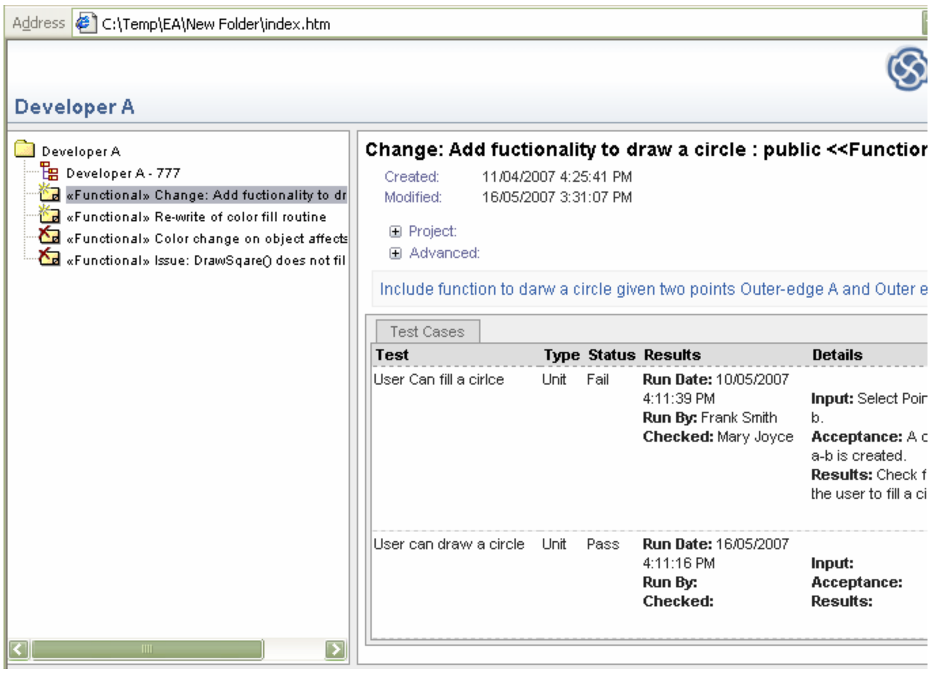
The SQL query builder only allows searches on the whole model, however it can used to generate a clear set of Test Cases using any conditions defined in the SQL statement.

Note: Class defines the Test Case Class (unit, Integration, etc.). These are defined numerically by the order displayed in the Test View window tab. Unit = 1, Integration=2, etc.

HTML reports

HTML reports can also be generated from EA. The HTML report generator is accessible using Project | Documentation | HTML Report or Shift-F8.

Below is an example of the data displayed for one of the above Issues with test cases:



Address C:\Temp\EA\New Folder\index.htm

Developer A

- Developer A
  - Developer A - 777
  - «Functional» Change: Add fuctionality to dr
  - «Functional» Re-write of color fill routine
  - «Functional» Color change on object affects
  - «Functional» Issue: DrawSqare() does not fil

Change: Add fuctionality to draw a circle : public <<Functior

Created:	11/04/2007 4:25:41 PM
Modified:	16/05/2007 3:31:07 PM

⊞ Project:
⊞ Advanced:

Include function to darw a circle given two points Outer-edge A and Outer e

Test Cases

Test	Type	Status	Results	Details
User Can fill a cirlce	Unit	Fail	**Run Date:** 10/05/2007 4:11:39 PM **Run By:** Frank Smith **Checked:** Mary Joyce	**Input:** Select Poir b. **Acceptance:** A c a-b is created. **Results:** Check f the user to fill a ci
User can draw a circle	Unit	Pass	**Run Date:** 16/05/2007 4:11:16 PM **Run By:** **Checked:**	**Input:** **Acceptance:** **Results:**

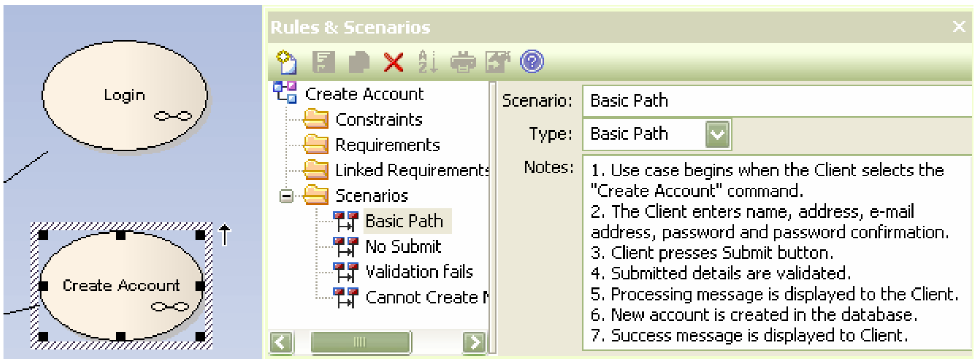
Figure 27 - Generated HTML viewed in a browser

Appendix

Import from Scenario, Constraints, Requirements and Other Tests

The test cases window supports importing data from other elements. This includes Scenarios, Constraints, Internal Requirements and Test Cases.

Common uses of this feature are importing Test Cases for a Class or an Issue from a Use Case. Below is an example of a Use Case showing a set of scenarios:

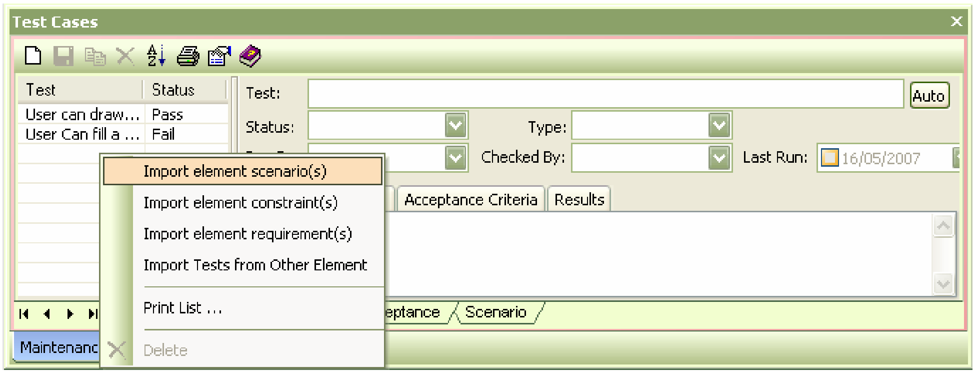


Scenario: Basic Path
Type: Basic Path
Notes: 1. Use case begins when the Client selects the "Create Account" command.
2. The Client enters name, address, e-mail address, password and password confirmation.
3. Client presses Submit button.
4. Submitted details are validated.
5. Processing message is displayed to the Client.
6. New account is created in the database.
7. Success message is displayed to Client.

Figure 28 - A Use Case Scenario

To start an import process:

1) Select the Element where the data is to be imported.
2) Open the Testing Window (Alt+3)
3) Right-click on the list of tests, then from the context menu, select an import type from the list of options. The following image shows the context menu;

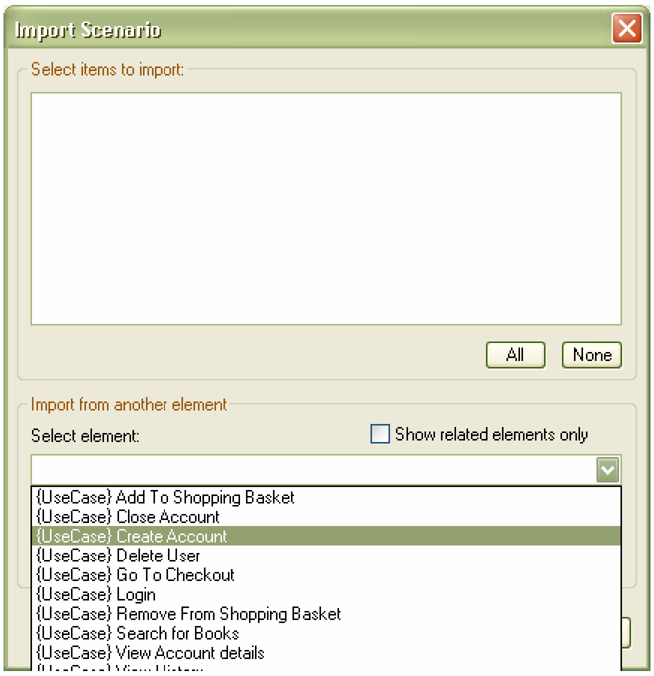


Test	Status
User can draw...	Pass
User Can fill a ...	Fail

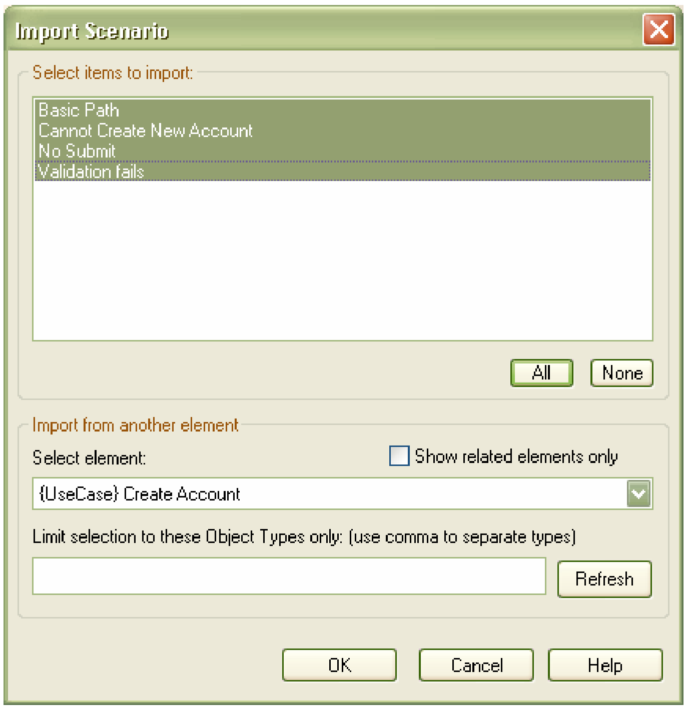
Last Run: 16/05/2007

Figure 29 - Option to Import Scenarios to Test Cases

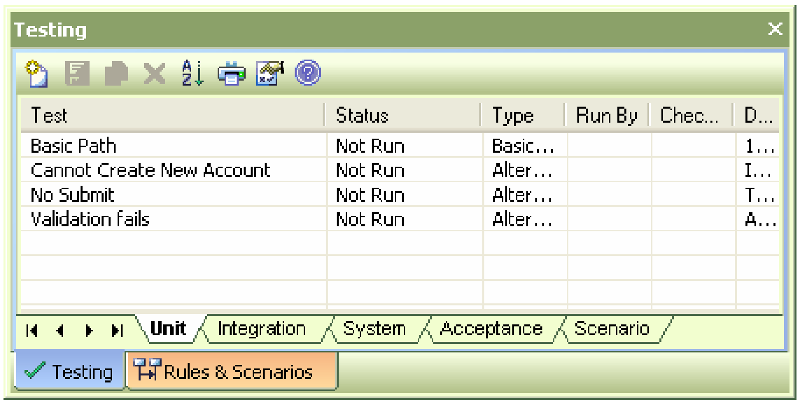
This opens the import dialog. Use the Select Element drop-down to retrieve a list of possible Elements that contain the scenarios:


Import Scenario
Select items to import:
All
None
Import from another element
Select element:
Show related elements only
{UseCase} Add To Shopping Basket
{UseCase} Close Account
{UseCase} Create Account
{UseCase} Delete User
{UseCase} Go To Checkout
{UseCase} Login
{UseCase} Remove From Shopping Basket
{UseCase} Search for Books
{UseCase} View Account details

On selection of an element, the scenarios available are displayed in Selected Items to Import. A set of entries can then be selected. The button: All - selects all entries to import.


Import Scenario
Select items to import:
Basic Path
Cannot Create New Account
No Submit
Validation fails
All
None
Import from another element
Select element:
Show related elements only
{UseCase} Create Account
Limit selection to these Object Types only: (use comma to separate types)
Refresh
OK
Cancel
Help

On selecting OK, the selected items will be added to the current set of Test Cases:



Test	Status	Type	Run By	Chec...	D...
Basic Path	Not Run	Basic...			1...
Cannot Create New Account	Not Run	Alter...			I...
No Submit	Not Run	Alter...			T...
Validation fails	Not Run	Alter...			A...

Defining Element Attributes Using a Profile

Elements can be pre-defined to include a set of user-defined attributes. These are used to document user specific qualities. Although these attributes cannot be added to the Element::Test Cases, they can be added to the element to define attributes related to the set of test cases. For example, attributes could store the date, product release number and contact information associated with an error report.

The additional attributes can be defined using either a *Model Template* or a *Profile Definition*.

A short comparison of these two options is as follows:

1. The Template definition is the simplest to set up, however:
    a. It pre-sets qualities of the default Element type used (e.g. An Issue Element defined in a template package dictates how all future Issue elements will be created for that model).
    b. Only one template definition can be made for any given element type for a given model (only one Issue type can be pre-set).
2. A Profile while more complex to set up, allows for multiple extension types for a given Element type.

Both of these require Attributes to be defined using the Tagged Values definition.

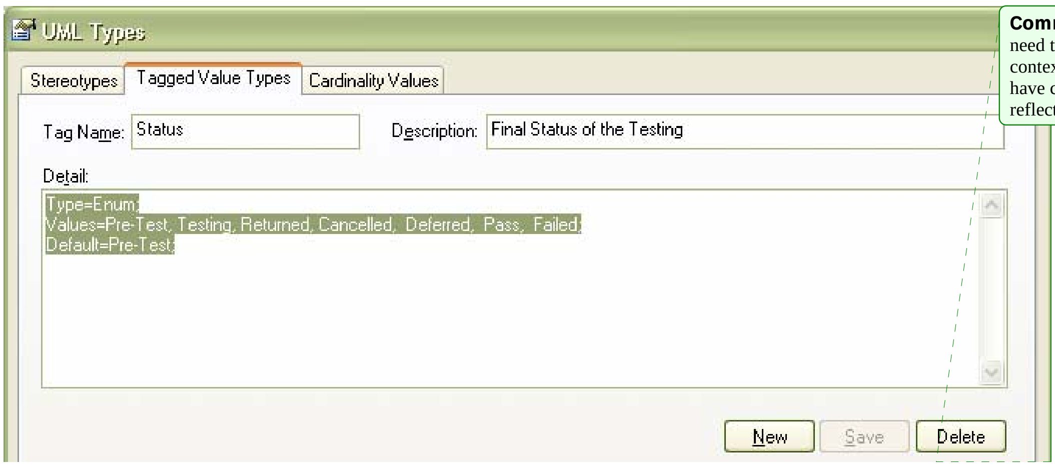
Note: If the project contains a number of Elements of the same type that contain different attributes, (i.e. Issue-Pre-Test, Issue-Post-Test) then the Profile approach, although more complex to set up, is the better one to use.

For more information on using a Template, see: Help | Index: Template, Package, Settings.

Defining Tagged Values

Tagged values allow users to define any number of fields with a wide variety of predefined or user-defined data types.

To set up a Tagged Value - select from the main menu Settings | UML | Tagged Value Types. This will bring up the definition window as shown below.



Tag Name: Status
Description: Final Status of the Testing
Detail:
Type=Enum;
Values=Pre-Test, Testing, Returned, Cancelled, Deferred, Pass, Failed;
Default=Pre-Test;

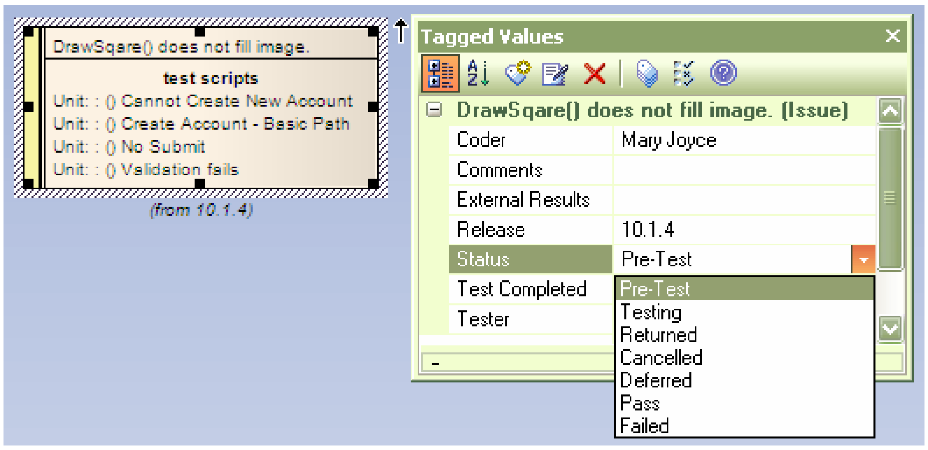
Comment: Ben I think this does need to stay as is – it sets the context for the examples below. I have cut it down and updated to reflect the latest build

Figure 30 - Tagged Values definition window.

In the example above, the Tagged Value selected, called "Status", uses a predefined type to display a drop-down list of selectable options. In the Detail area it contains:

Type=Enum;
Values=Pre-Test, Testing, Returned, Cancelled, Deferred, Pass, Failed;
Default=Pre-Test;

When viewed in the Tagged Values window this is presented as a drop-down option box:


DrawSqare() does not fill image.
test scripts
Unit: : () Cannot Create New Account
Unit: : () Create Account - Basic Path
Unit: : () No Submit
Unit: : () Validation fails
(from 10.1.4)
Tagged Values

DrawSqare() does not fill image. (Issue)

Coder	Mary Joyce
Comments	
External Results	
Release	10.1.4
Status	Pre-Test
Test Completed	
Tester	

Pre-Test
Testing
Returned
Cancelled
Deferred
Pass
Failed

Figure 31 - Elements definition with associated Tagged Values.

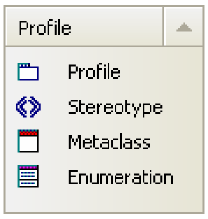
There are numerous standard types available such as numeric and string types; Enumerated lists (see above), Date-Time, Boolean, Memo, etc. For more information on setting up the standard types and a list of types available in Enterprise Architect's help, use the Index tab to locate 'Predefined Tagged Value Types'

Defining Additional Attributes Using a Profile

Enterprise Architect supports the creation of Profiles. Profiles allow the user to define a set of extensions to standard Elements using Tagged Values. See Defining Tagged Values above.

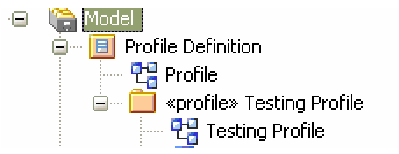
To create a Profile:

1. In the Project Browser, set up a specific package that will contain the Profile
2. Create a diagram for the Profile under this Package
3. In the UML toolbox open the Profile section



5. Drag the Profile element onto the diagram.

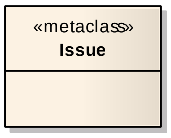
Once you have entered a name for the profile, a package will be created with the «profile» stereotype and a child diagram beneath it. This child diagram (eg. 'Testing Profile' in the image below) will be used to add stereotypes to the profile.



To add a new Element definition:

1. Open the system created diagram contained in the «profile» package
2. Create a Metaclass element by dragging the Metaclass tool from the Profile Toolbox. This will bring up a dialog box to select the type of Element that is to be created
3. For an Issue select the Issue Element type
4. Press OK.

The Metaclass will appear as:

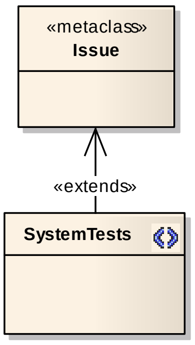


To define where the Tagged Values will be included in the profile:

1. Drag the Stereotype tool, from the Profiles toolbox, onto the diagram

2. In the Properties dialog box, select the Name field. Type in your preferred name for the element in the new Toolbox, i.e. 'SystemTests'
3. Press OK in the Properties dialog
4. From the Profile Toolbox click on the Extension Connector, select the new Stereotype element 'Bug' and drag the mouse to the Requirement Metaclass. This should create an Extension connector between these two elements.

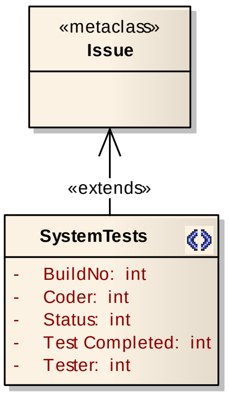
Below is a view of the elements and connection created:



To add the Tagged Values

1. Select the Stereotype (i.e. 'SystemTests')
2. From the context menu, select Attributes
3. In the Attributes dialog box, add as a new entry in the Name field, for each of the Tagged Values created above, you want included in this 'SystemTests' Element.

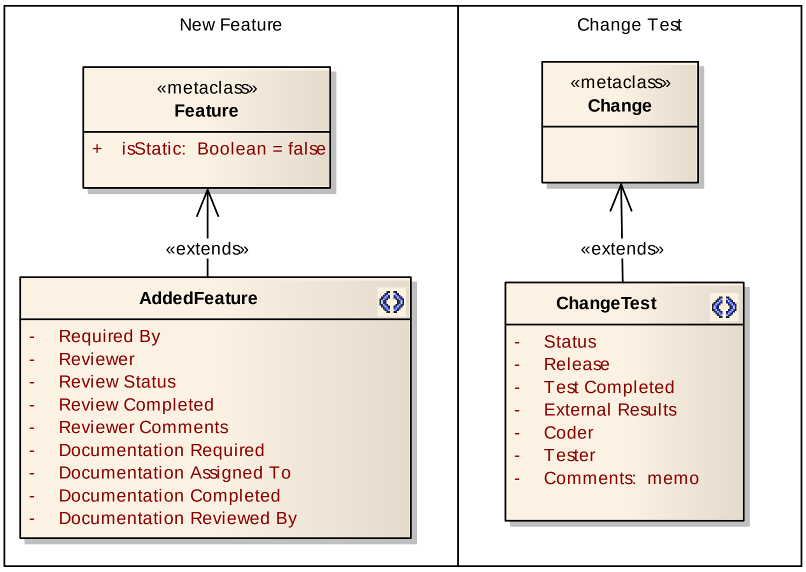
Below is an example of a the above 'SystemTests' populated with some of the Tagged Values defined in Figure 30 - Tagged Values definition window.



Note: In the case of creating simple String or Boolean Tagged Values, these do not need to be defined in the Tagged Values definition, but can be entered directly in the Attributes of the 'SystemTests' Element.

Any number of these definitions can be set up in one profile. Each Stereotype Element would need to have a unique name.

Below is an example of profiles for two user-defined elements (in swimlanes) – one for logging new features and one for logging changes with related test details. See Figure 31 for an example of the ChangeTest.



To set the new Element types to be viewed in the Toolbox

1. Select the «profile» package
2. Right-click and from the context menu, select Save Package to UML Profile.
3. Set the filename to save the XMI file
4. Select Save
5. Open the Resources View
6. From the Resources tree, select UML Profiles
7. Right-click and from the context menu, select Import Profile.

Once this has been imported, it can be included on the Toolbox using the following:

1. Select the newly imported profile
2. Right-click on the new profile and from the context menu select: Show Profile in UML Toolbox.

A new Toolbox with the name of your Profile package will be added to the Toolbox. Now you can create your custom Feature and Change Elements in any package by dragging these from the Toolbox.