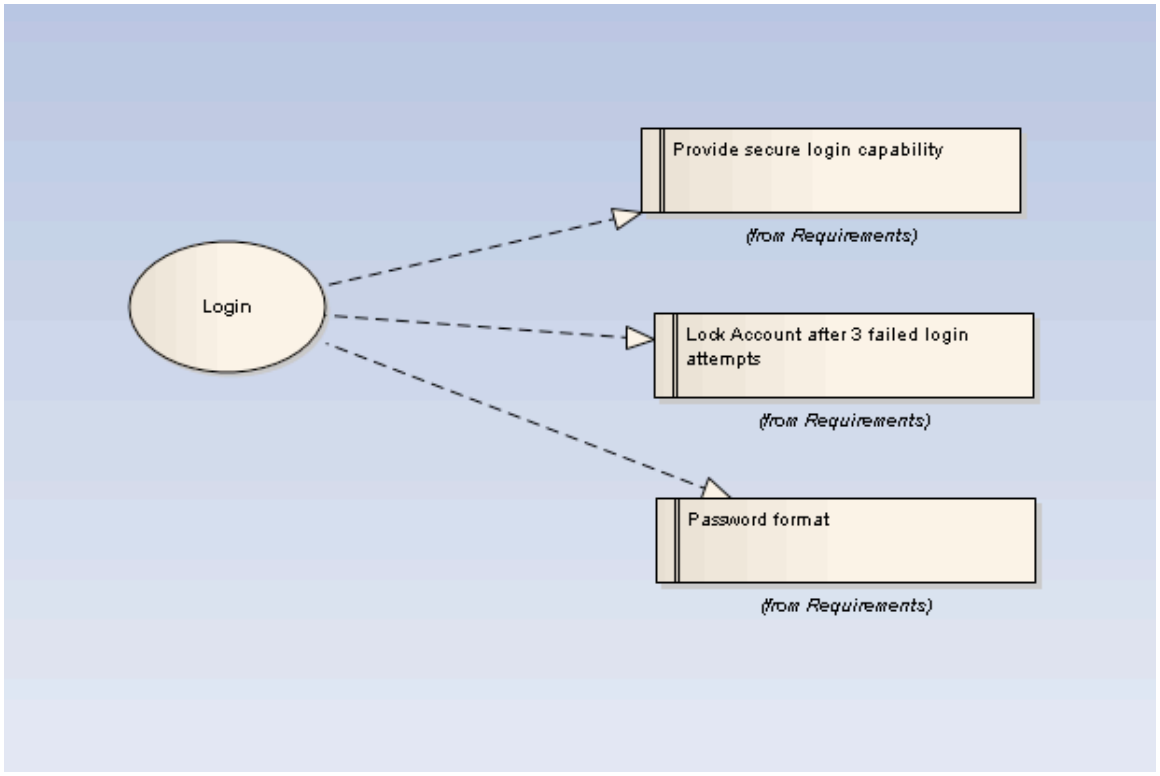
Design-Driven Testing using the Agile/ICONIX Add-In

Doug Rosenberg, ICONIX
Matt Stephens, Software Reality

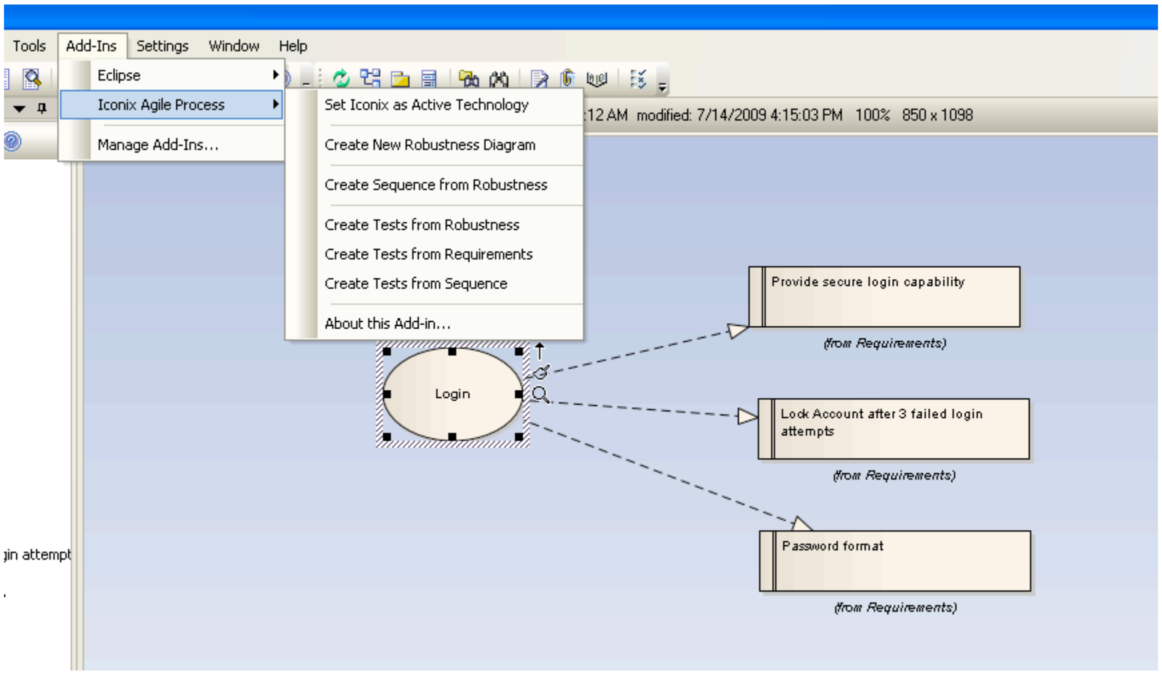
ICONIX and Sparx Systems have collaborated on the production of a new Add-In that extends the functionality of Enterprise Architect to support Design-Driven Testing (DDT). DDT (also known as **ICONIX Process for Test**) will be described more completely in an upcoming book by Doug Rosenberg and Matt Stephens. This article describes how to use the Agile/ICONIX add-in to drive test code from UML models.

Many thanks to Tom O'Reilly and Aaron Bell from Sparx Systems for their work developing the add-in.

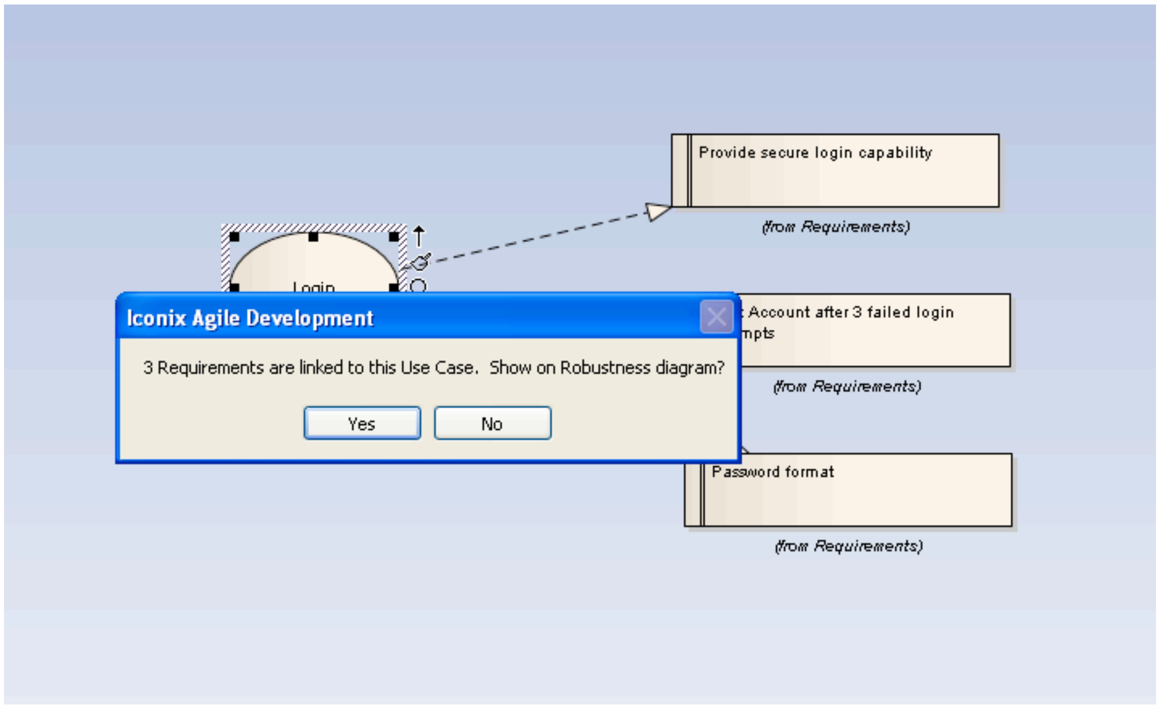
Let's consider a simple Login use case, which satisfies 3 Requirements.



We can automatically generate a Robustness Diagram using the add-in, by selecting the use case and choosing the Agile/ICONIX add-in from the Add-In menu:

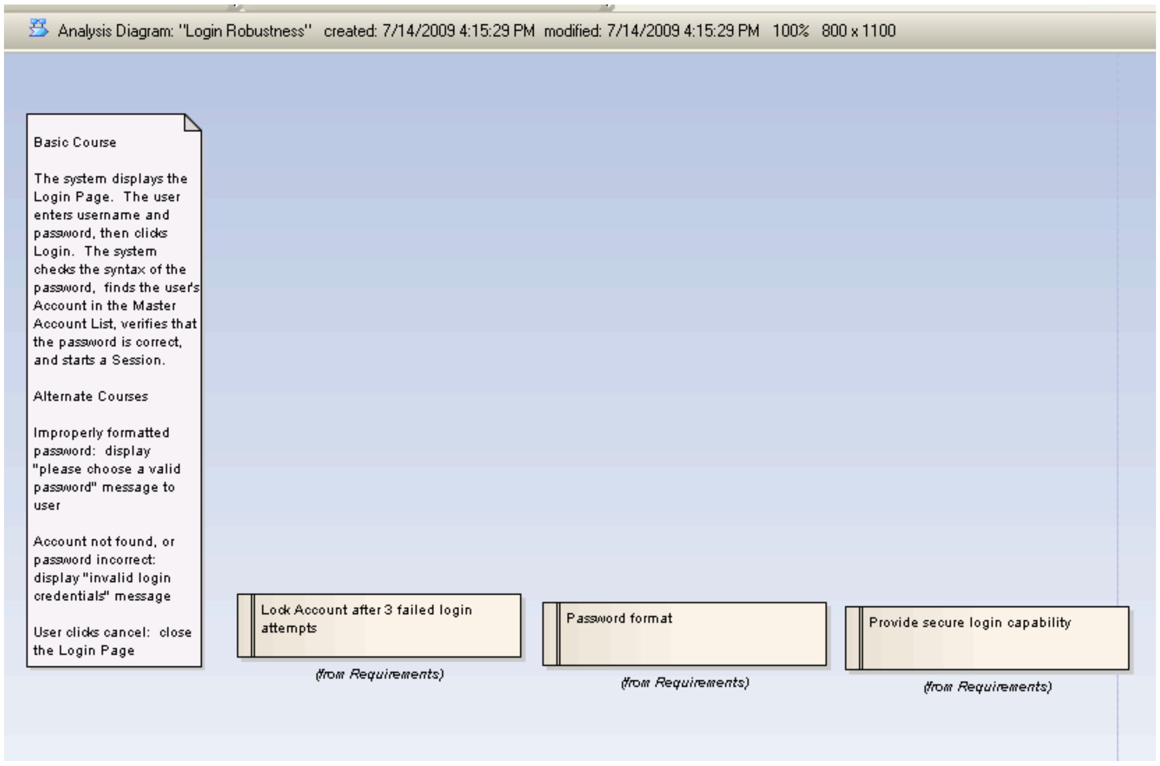

Tools
Add-Ins
Settings
Window
Help
Eclipse
Iconix Agile Process
Manage Add-Ins...
Set Iconix as Active Technology
Create New Robustness Diagram
Create Sequence from Robustness
Create Tests from Robustness
Create Tests from Requirements
Create Tests from Sequence
About this Add-in...
:12 AM modified: 7/14/2009 4:15:03 PM 100% 850 x 1098
Login
Provide secure login capability
(from Requirements)
Lock Account after 3 failed login attempts
(from Requirements)
Password format
(from Requirements)
jin attempt

The add-in creates a new Robustness diagram under the use case in the Project Browser, and propagates the Requirements (if desired) onto the Robustness Diagram.



3 Requirements are linked to this Use Case. Show on Robustness diagram?

In addition to propagating the Requirements, the add-in creates a new Note on the Robustness diagram that is hot-linked to the use case text.


Analysis Diagram: "Login Robustness" created: 7/14/2009 4:15:29 PM modified: 7/14/2009 4:15:29 PM 100% 800 x 1100

Basic Course

The system displays the Login Page. The user enters username and password, then clicks Login. The system checks the syntax of the password, finds the user's Account in the Master Account List, verifies that the password is correct, and starts a Session.

Alternate Courses

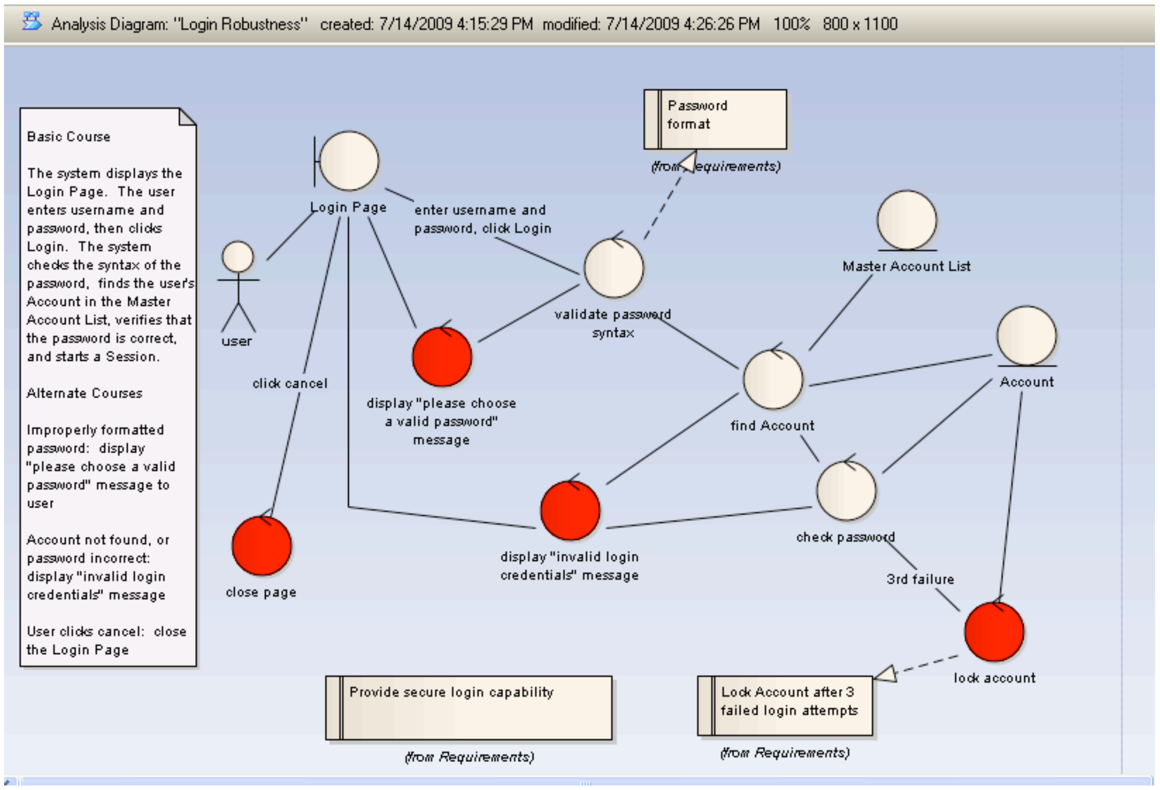
Improperly formatted password: display "please choose a valid password" message to user

Account not found, or password incorrect: display "invalid login credentials" message

User clicks cancel: close the Login Page

Lock Account after 3 failed login attempts
(from Requirements)
Password format
(from Requirements)
Provide secure login capability
(from Requirements)

Having the Requirements on the Robustness diagram helps us to make sure we haven't forgotten any Requirements as we analyze the use case.


Analysis Diagram: "Login Robustness" created: 7/14/2009 4:15:29 PM modified: 7/14/2009 4:26:26 PM 100% 800 x 1100

Basic Course

The system displays the Login Page. The user enters username and password, then clicks Login. The system checks the syntax of the password, finds the user's Account in the Master Account List, verifies that the password is correct, and starts a Session.

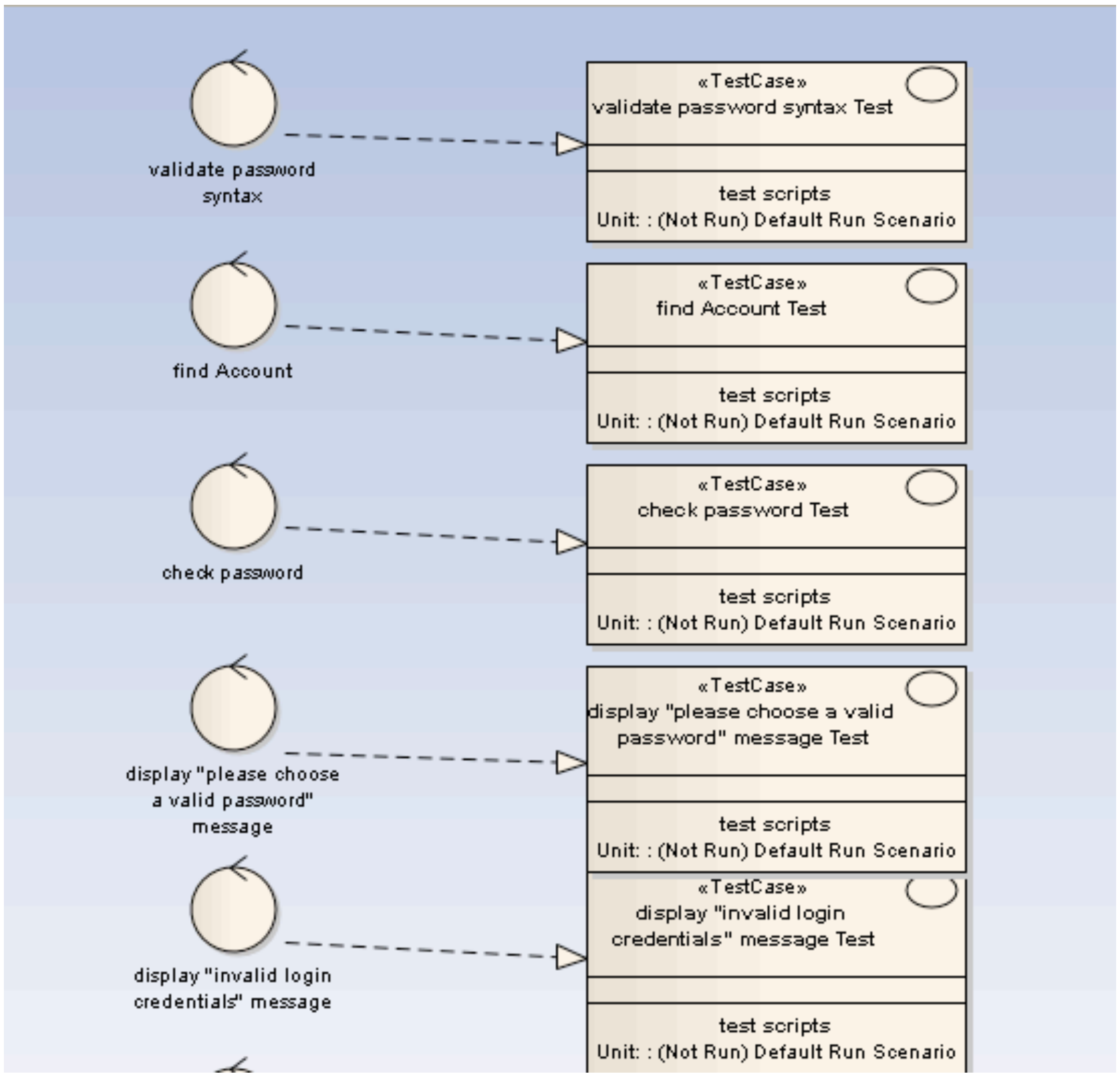
Alternate Courses

Improperly formatted password: display "please choose a valid password" message to user

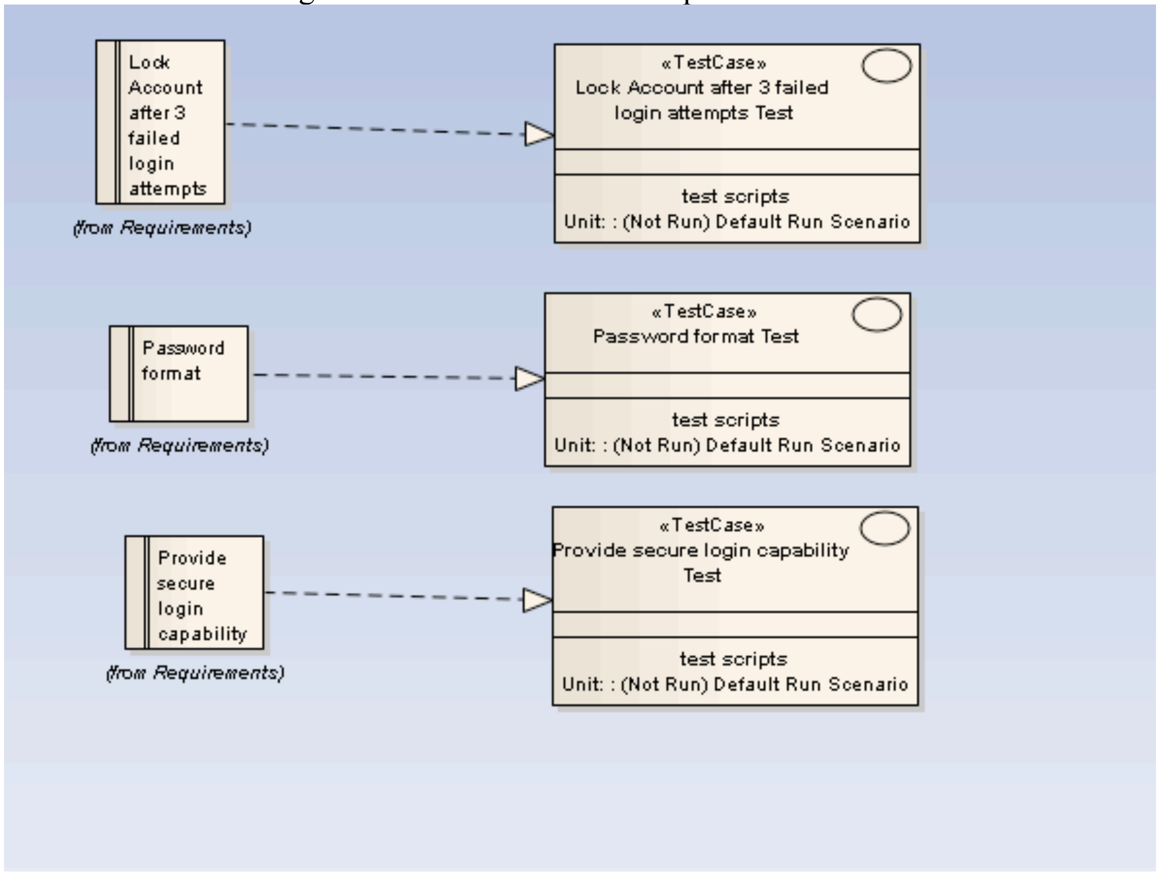
Account not found, or password incorrect: display "invalid login credentials" message

User clicks cancel: close the Login Page

By analyzing our use case using the Robustness Analysis technique, we've added a lot of information to the model that can be used to drive the testing of the application. To begin with, our add-in can automatically generate test cases for each "controller" on the Robustness diagram.

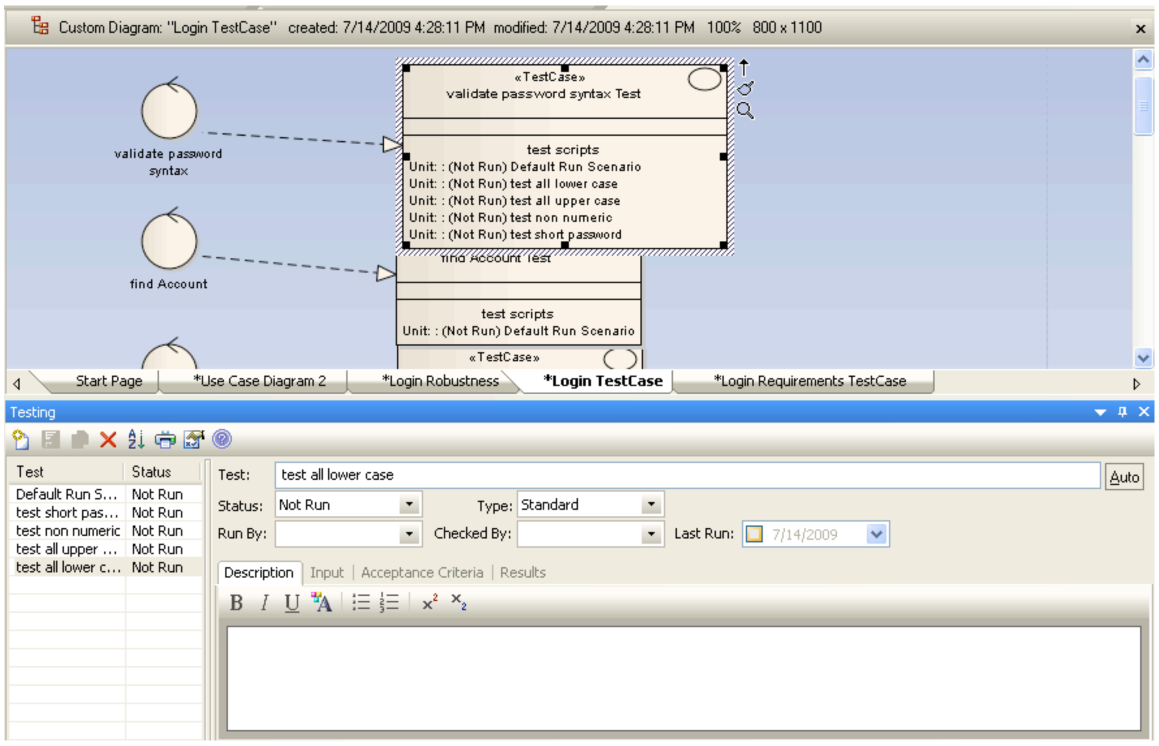

validate password syntax
«TestCase» validate password syntax Test
test scripts
Unit: : (Not Run) Default Run Scenario
find Account
«TestCase» find Account Test
test scripts
Unit: : (Not Run) Default Run Scenario
check password
«TestCase» check password Test
test scripts
Unit: : (Not Run) Default Run Scenario
display "please choose a valid password" message
«TestCase» display "please choose a valid password" message Test
test scripts
Unit: : (Not Run) Default Run Scenario
display "invalid login credentials" message
«TestCase» display "invalid login credentials" message Test
test scripts
Unit: : (Not Run) Default Run Scenario

We can also choose to generate test cases for our Requirements.


Lock Account after 3 failed login attempts
(from Requirements)
«TestCase» Lock Account after 3 failed login attempts Test
test scripts
Unit: : (Not Run) Default Run Scenario
Password format
(from Requirements)
«TestCase» Password format Test
test scripts
Unit: : (Not Run) Default Run Scenario
Provide secure login capability
(from Requirements)
«TestCase» Provide secure login capability Test
test scripts
Unit: : (Not Run) Default Run Scenario

In some organizations, the development team might own the “controller tests” while an independent QA team might own the “requirement tests”. We can also generate test cases for messages on Sequence diagrams.

Once we’ve generated the test cases, we can add Scenarios using Enterprise Architect’s “Testing View”. In this example shown, we’re adding scenarios to the test case that validates the format of a password. We need to check that the password is of the required length, contains a mix of alpha and numeric characters, and is a mix of upper and lower case.



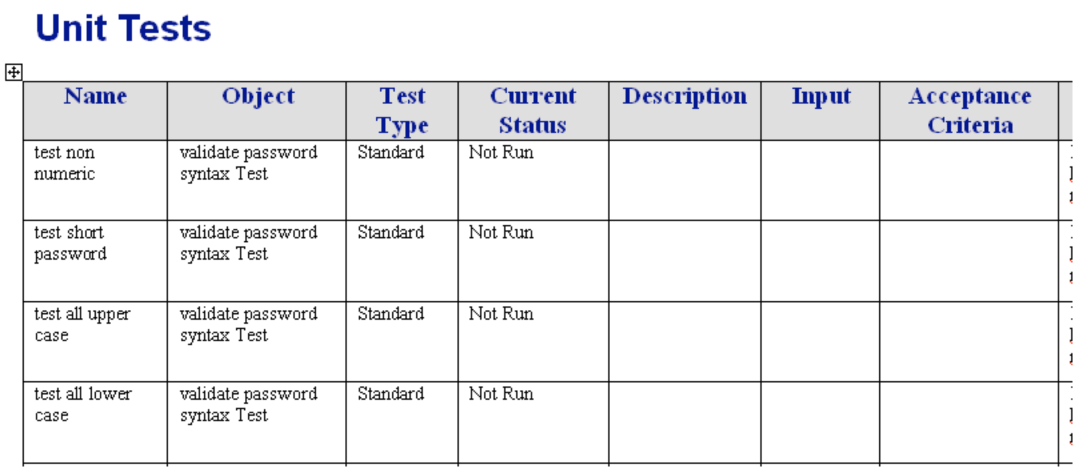
test scripts
Unit: : (Not Run) Default Run Scenario
Unit: : (Not Run) test all lower case
Unit: : (Not Run) test all upper case
Unit: : (Not Run) test non numeric
Unit: : (Not Run) test short password

test scripts
Unit: : (Not Run) Default Run Scenario

Test	Status
Default Run S...	Not Run
test short pas...	Not Run
test non numeric	Not Run
test all upper ...	Not Run
test all lower c...	Not Run

Test: test all lower case
Status: Not Run Type: Standard
Run By: Checked By: Last Run: 7/14/2009

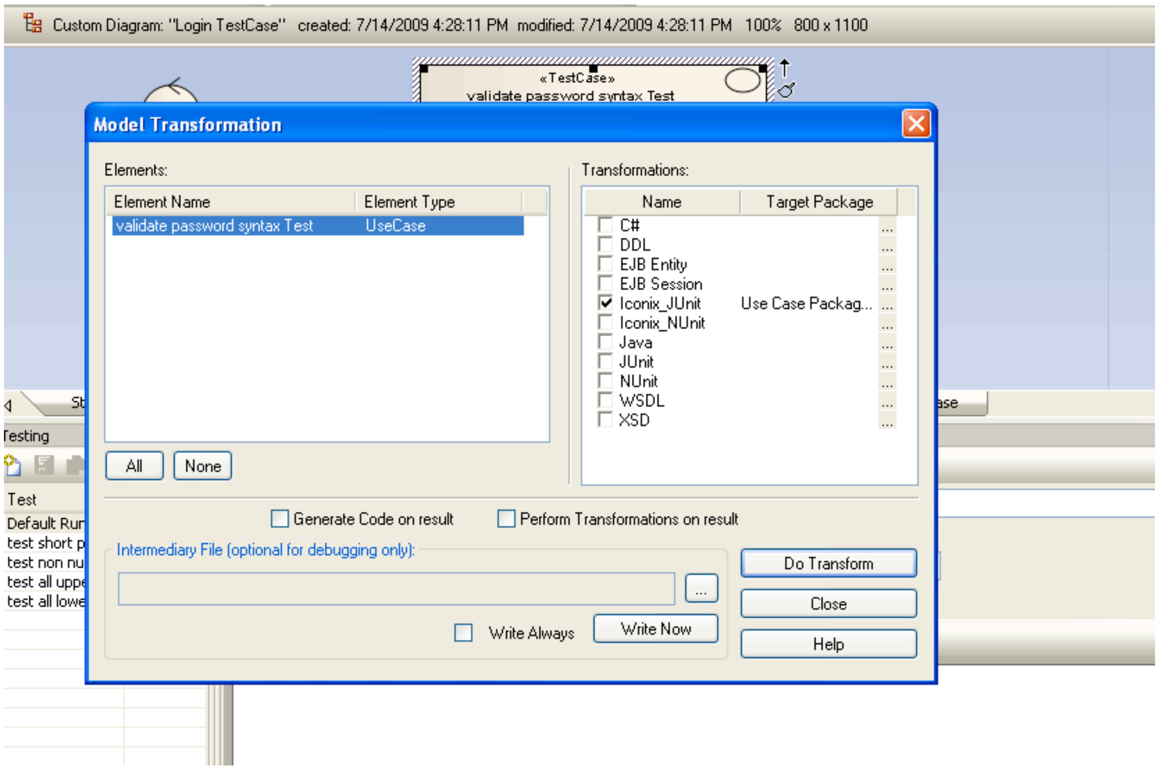
We can easily generate test plan reports using Enterprise Architect's built-in report generation capability. So driving our testing activity from the model is very straightforward.



Unit Tests

Name	Object	Test Type	Current Status	Description	Input	Acceptance Criteria
test non numeric	validate password syntax Test	Standard	Not Run			
test short password	validate password syntax Test	Standard	Not Run			
test all upper case	validate password syntax Test	Standard	Not Run			
test all lower case	validate password syntax Test	Standard	Not Run			

Even if the add-in did nothing more than what's been shown so far, it would still be extremely useful. But there's significantly more capability ahead. Sparx Systems has developed automatic Transforms between ***test cases*** and ***test classes***. Transforms exist for widely used unit testing frameworks JUnit (for Java) and NUnit (for .Net).


Custom Diagram: "Login TestCase" created: 7/14/2009 4:28:11 PM modified: 7/14/2009 4:28:11 PM 100% 800 x 1100

Model Transformation

Elements:

Element Name	Element Type
validate password syntax Test	UseCase

Transformations:

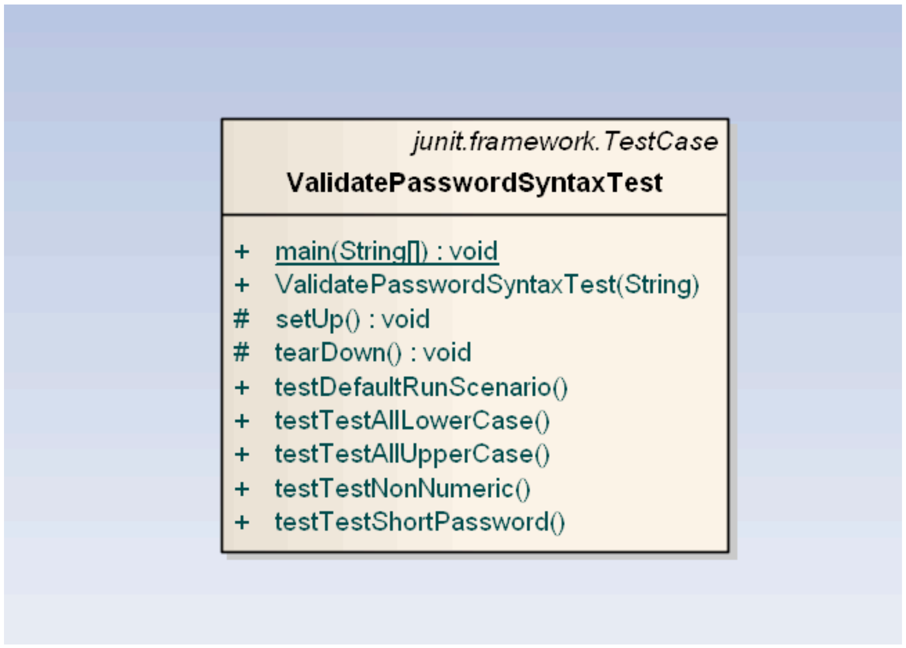
Name	Target Package
☐ C#	
☐ DDL	
☐ EJB Entity	
☐ EJB Session	
☑ Iconix_JUnit	Use Case Packag...
☐ Iconix_NUnit	
☐ Java	
☐ JUnit	
☐ NUnit	
☐ WSDL	
☐ XSD	

☐ Generate Code on result ☐ Perform Transformations on result

Intermediary File (optional for debugging only):

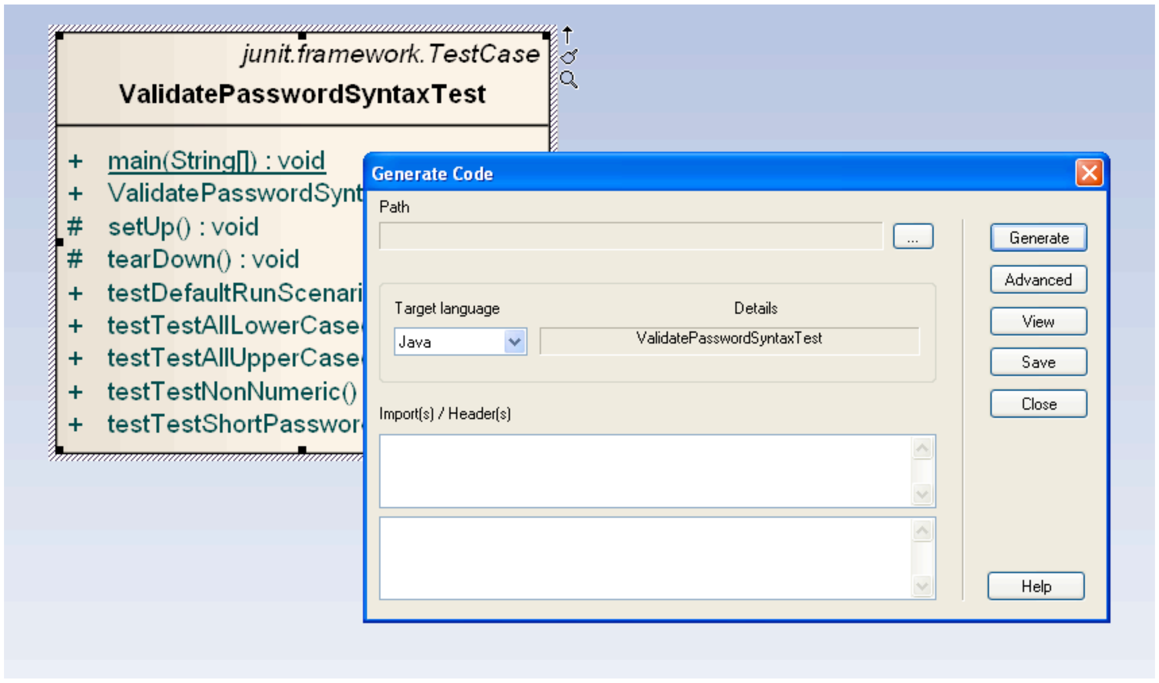
☐ Write Always

When we transform our test case using the ICONIX JUnit transform, “setup()” and “tearDown()” methods are added automatically, and a test method is created for each scenario. This transformation, originally envisioned by Matt Stephens while we were writing the book “Agile Development with ICONIX Process[1]”, and implemented by Sparx Systems, is the essence of the Design-Driven Testing approach.


```
junit.framework.TestCase
ValidatePasswordSyntaxTest

+ main(String[]) : void
+ ValidatePasswordSyntaxTest(String)
# setUp() : void
# tearDown() : void
+ testDefaultRunScenario()
+ testTestAllLowerCase()
+ testTestAllUpperCase()
+ testTestNonNumeric()
+ testTestShortPassword()
```

[1] http://www.iconixsw.com/Books.html

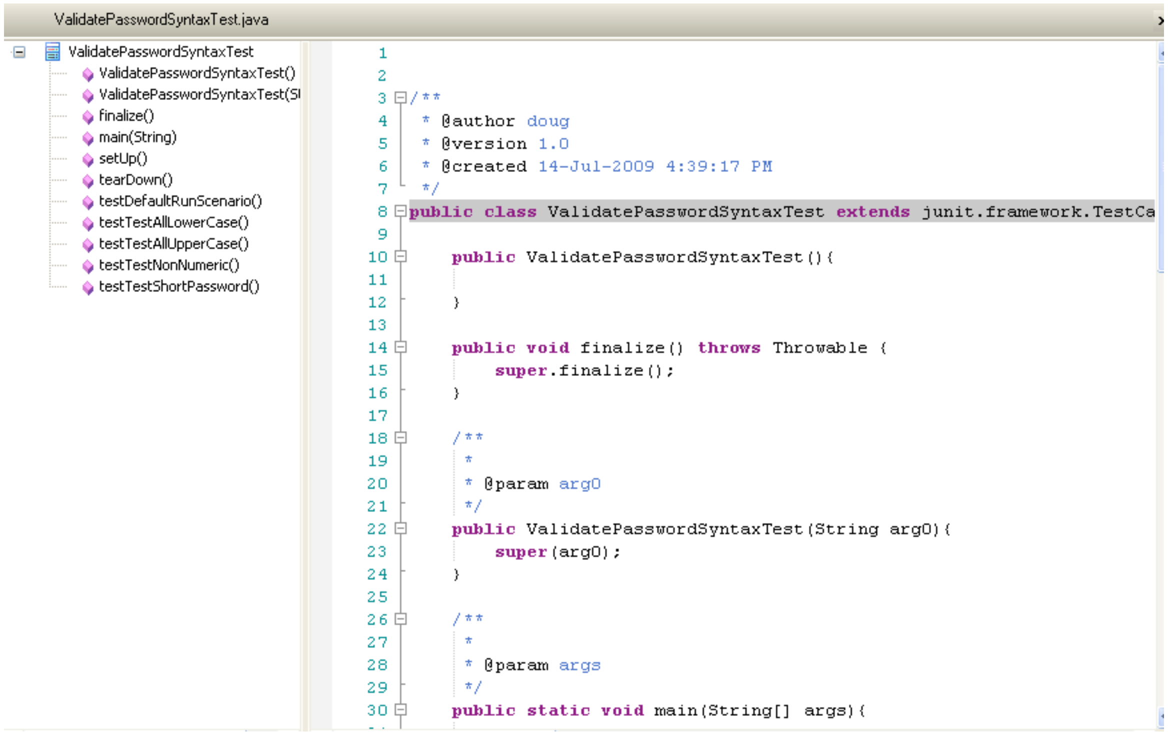
Once the test cases have been transformed to test classes, they can be code-generated, using Enterprise Architect’s standard code-generation techniques.



Target language: Java

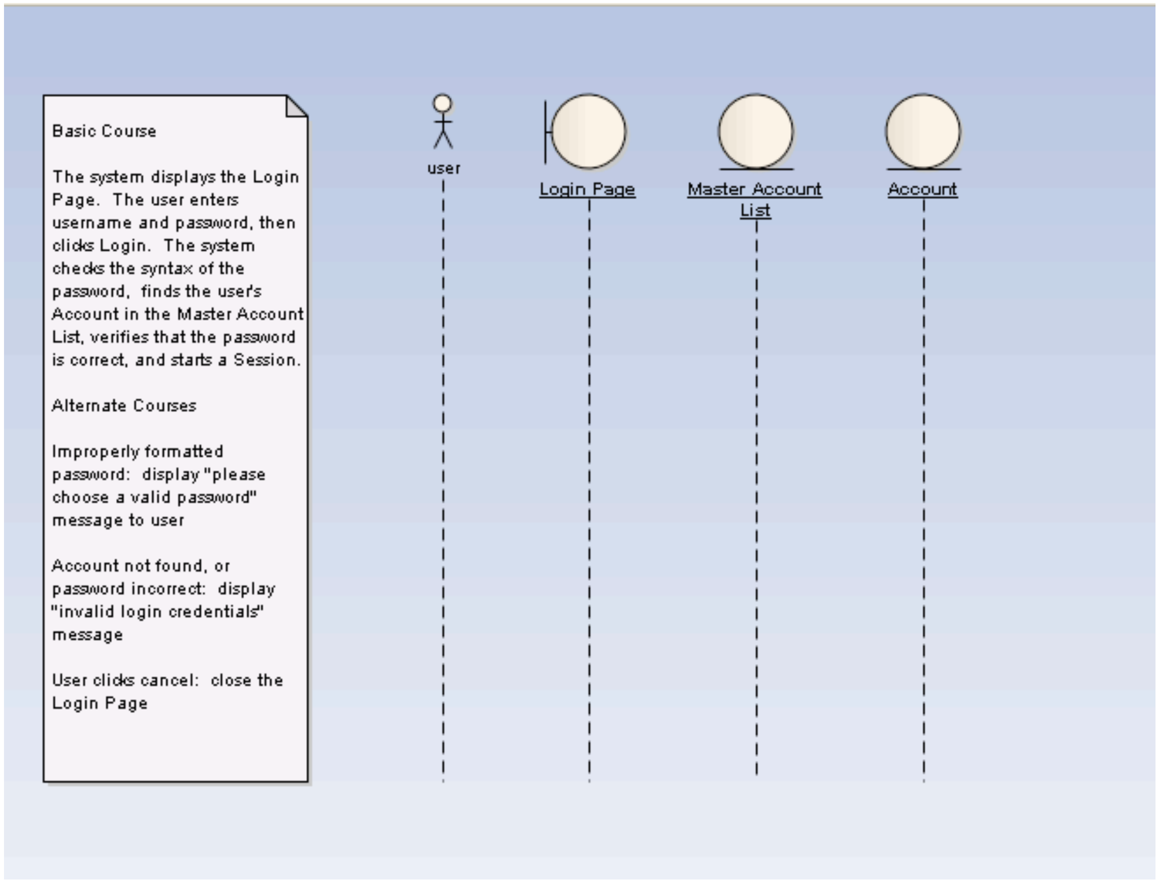
Details: ValidatePasswordSyntaxTest

NUnit test classes can be generated in a variety of languages; the example shown is a JUnit test class, code generated in Java.



```


/**
 * @author doug
 * @version 1.0
 * @created 14-Jul-2009 4:39:17 PM
 */
public class ValidatePasswordSyntaxTest extends junit.framework.TestCa

    public ValidatePasswordSyntaxTest(){

    }

    public void finalize() throws Throwable {
        super.finalize();
    }

    /**
     *
     * @param arg0
     */
    public ValidatePasswordSyntaxTest(String arg0){
        super(arg0);
    }

    /**
     *
     * @param args
     */
    public static void main(String[] args){
```

In addition to being useful during Requirements definition and Analysis/Conceptual Design phases of a project, the Agile/ICONIX add-in also delivers value when transitioning from the Analysis phase to Detailed Design, by automatically generating skeleton Sequence diagrams from Robustness Diagrams.



Basic Course

The system displays the Login Page. The user enters username and password, then clicks Login. The system checks the syntax of the password, finds the user's Account in the Master Account List, verifies that the password is correct, and starts a Session.

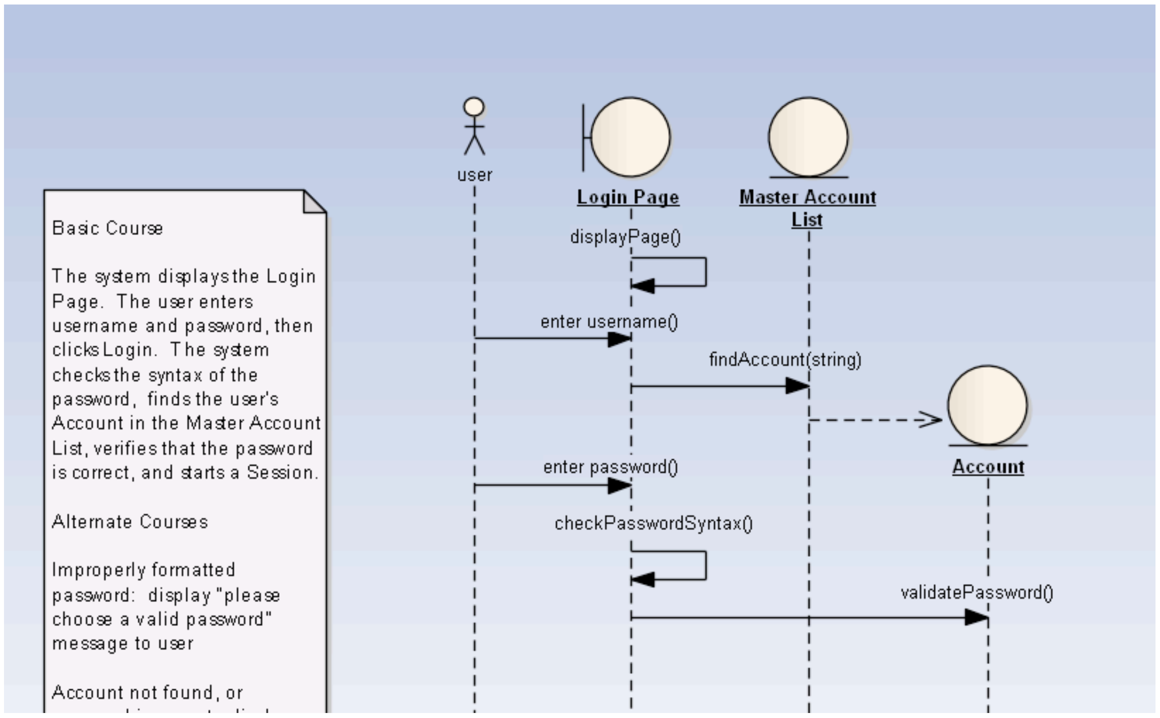
Alternate Courses

Improperly formatted password: display "please choose a valid password" message to user

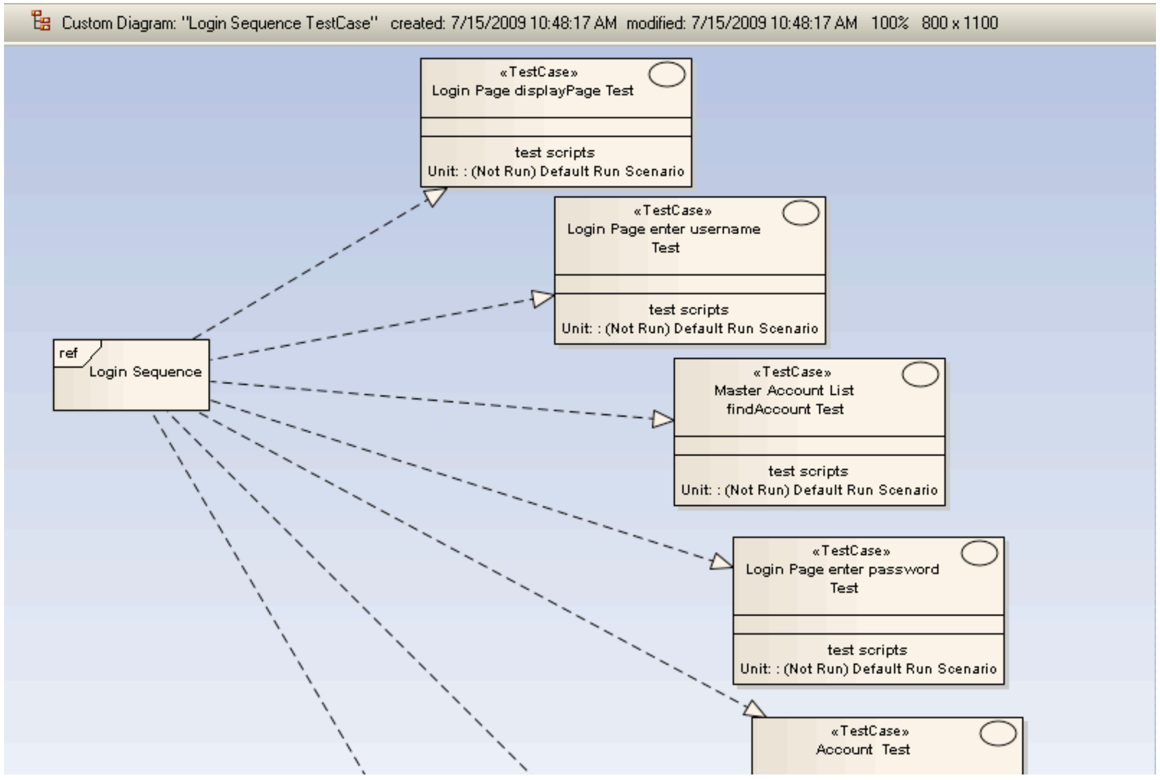
Account not found, or password incorrect: display "invalid login credentials" message

User clicks cancel: close the Login Page

The add-in automatically creates a new sequence diagram, and brings a "hot-linked" note, and all Boundary and Entity objects from the Robustness diagram onto the Sequence diagram. Developers then complete the design of the use case on the Sequence diagram.

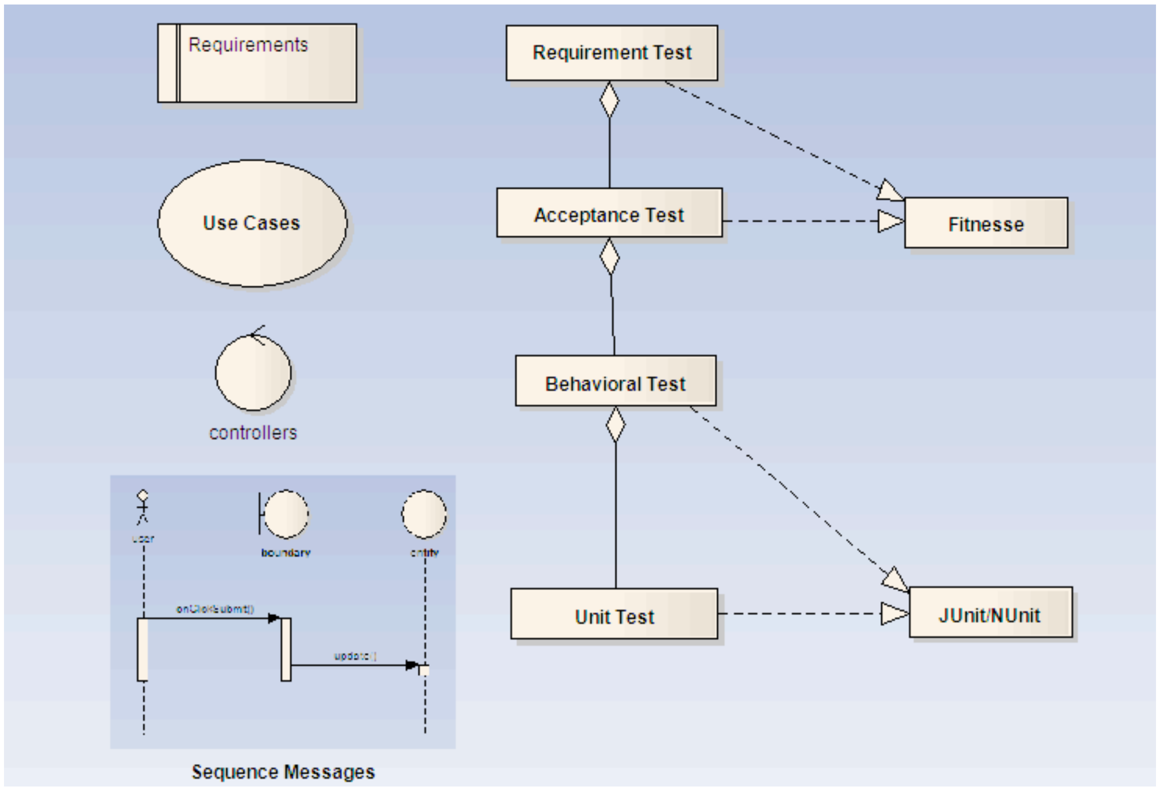

Basic Course
The system displays the Login Page. The user enters username and password, then clicks Login. The system checks the syntax of the password, finds the user's Account in the Master Account List, verifies that the password is correct, and starts a Session.
Alternate Courses
Improperly formatted password: display "please choose a valid password" message to user
Account not found, or
user
Login Page
Master Account List
Account
displayPage()
enter username()
findAccount(string)
enter password()
checkPasswordSyntax()
validatePassword()

Once the Sequence diagram is completed, the Add-in is used to generate unit tests from the messages on the Sequence Diagram. Note that "controllers" on Robustness diagrams tend to reflect a slightly higher abstraction level than Sequence messages; that is, a controller may be implemented as multiple messages on a Sequence diagram.


Custom Diagram: "Login Sequence TestCase" created: 7/15/2009 10:48:17 AM modified: 7/15/2009 10:48:17 AM 100% 800 x 1100
ref
Login Sequence
«TestCase»
Login Page displayPage Test
test scripts
Unit: : (Not Run) Default Run Scenario
«TestCase»
Login Page enter username Test
test scripts
Unit: : (Not Run) Default Run Scenario
«TestCase»
Master Account List findAccount Test
test scripts
Unit: : (Not Run) Default Run Scenario
«TestCase»
Login Page enter password Test
test scripts
Unit: : (Not Run) Default Run Scenario
«TestCase»
Account Test

Using the DDT process, we have the flexibility to test against Requirements, against Controllers ("logical functions") and to do true unit testing at the Sequence message level. All of these abstraction levels should be verified by our testing.

Thus the Add-in delivers value during the Requirements, Analysis, and Design phases of the lifecycle, and automates the process of driving test plans and test code directly from the models.



The Agile/ICONIX add-in enables a systematic approach to software development that drives tests from a UML-based design. The authors believe this approach offers significant advantages over "Test-Driven Development", where the "design" is driven by unit-testing.