

## **CA314 – Object Oriented Analysis & Design - 5**

**File name:** CA314\_Section\_05\_Ver01

**Author:** Heumann (Tuohey)

**No. of pages:** 11

**Table of Contents**

5. Generating Test Cases From Use Cases .....3  
5.1 Motivation.....3  
5.2. Introduction.....3  
5.2.1 Context.....3  
5.2.2 Recap of Use Cases.....4  
5.2.3 Basic and Alternate Flow of Events in Use Cases.....4  
5.2.4 Use-Case Scenarios .....8  
5.3 Generating Test Cases.....8  
5.3.0 Overview .....8  
5.3.1 Step One: Generate Scenarios .....9  
5.3.2 Step Two: Identify Test Cases .....9  
5.3.3 Step Three: Identify Data Values to Test..... 10  
5.4 Conclusion: Putting It All Together..... 11

## 5. Generating Test Cases From Use Cases<sup>1</sup>

### 5.1 Motivation

In many organizations, software testing accounts for 30 to 50 percent of software development costs. Yet most people believe that software is not well tested before it is delivered. That contradiction is rooted in two clear facts: First, testing software is a very difficult proposition; and second, testing is typically done without a clear methodology. A widely-accepted tenet in the industry – and an integral assumption in the Rational Unified

Process® (RUP®) -- is that it is better to start testing as early in the software development process as possible. Delaying the start of testing activities until all development is done is a high-risk way to proceed. If significant bugs are found at that stage (and they usually are), then schedules often slip. Haphazard methods of designing, organizing, and implementing testing activities and artifacts also frequently lead to less-than-adequate test coverage. Having a straightforward plan for how testing is done can help increase coverage, efficiency, and ultimately software quality.

### 5.2. Introduction

#### 5.2.1 Context

In this article, we will discuss how using use cases to generate test cases can help launch the testing process early in the development lifecycle and also help with testing methodology. In a software development project, use cases define system software requirements. Use case development begins early on, so real use cases for key product functionality are available in early iterations. According to the RUP, a use case "...fully describes a sequence of actions performed by a system to provide an observable result of value to a person or another system using the product under development." Use cases tell the customer what to expect, the developer what to code, the technical writer what to document, and the tester what to test.

For software testing -- which consists of many interrelated tasks, each with its own artifacts and deliverables -- creation of **test cases** is the first fundamental step. Then

---

<sup>1</sup> From Jim Heumann, Rational Software (<http://www-128.ibm.com/developerworks/rational/library/content/RationalEdge/jun01/GeneratingTestCasesFromUseCasesJune01.pdf>)

Copyright Rational Software 2001 [http://www.therationaledge.com/content/jun\\_01/m\\_cases\\_jh.html](http://www.therationaledge.com/content/jun_01/m_cases_jh.html)

**test procedures** are designed for these test cases, and finally, **test scripts** are created to implement the procedures. Test cases are key to the process because they identify and communicate the conditions that will be implemented in test and are necessary to verify successful and acceptable implementation of the product requirements. They are all about making sure that the product fulfills the requirements of the system. Although few actually do it, developers can begin creating test cases as soon as use cases are available, well before any code is written. We will discuss how to do this, and the advantages you can reap from it, below.

### 5.2.2 Recap of Use Cases

We have seen that **use cases** are based on the Unified Modeling Language (UML) and can be visually represented in use-case diagrams – such diagrams provides the big picture: Each use case represents a big chunk of functionality that will be implemented, and each actor represents someone or something outside our system that interacts with it. Each use case also requires a significant amount of text to describe it. This text is usually formatted in sections, such as the following:

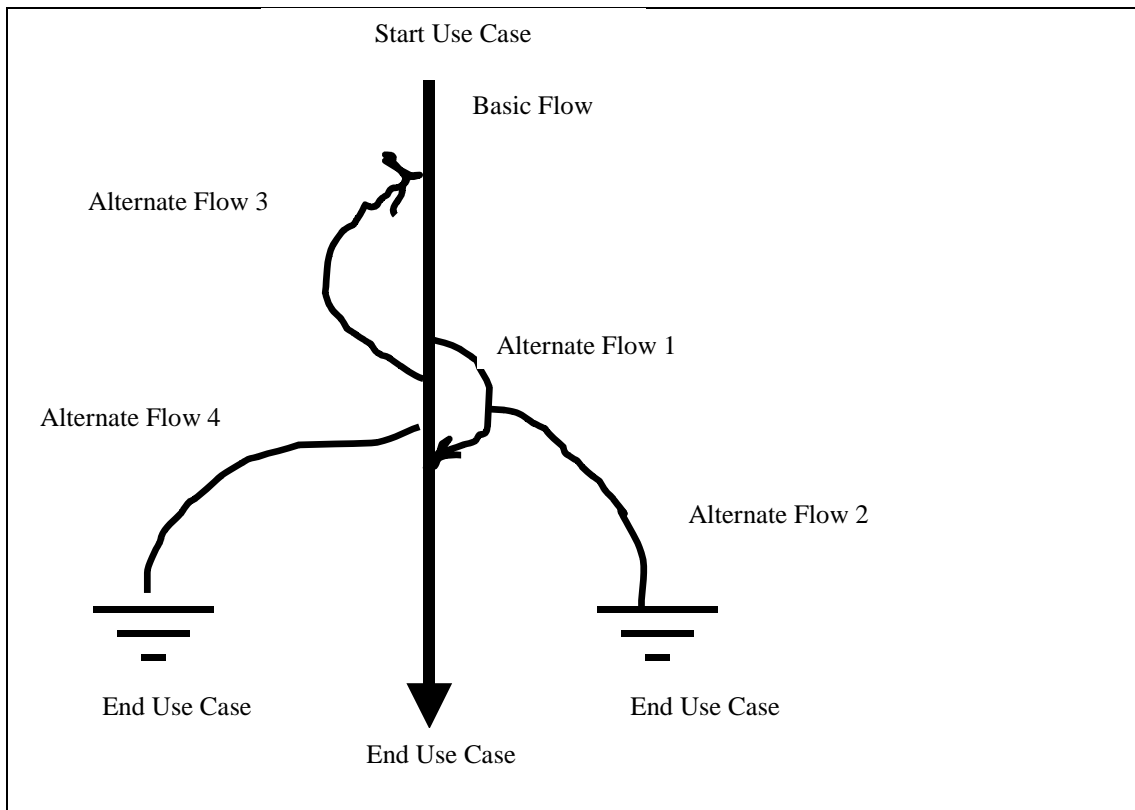
Use Case Section	Description
Name	An appropriate name for the use case
Brief Description	A brief description of the use case's role and purpose.
Flow of Events	A textual description of what the system does with regard to the use case (not how specific problems are solved by the system). The description should be understandable to the customer.
Special Requirements	A textual description that collects all requirements, such as non-functional requirements, on the use case, that are not considered in the use-case model, but that need to be taken care of during design or implementation.
Preconditions	A textual description that defines any constraints on the system at the time the use case may start.
Post conditions	A textual description that defines any constraints on the system at the time the use case will terminate.

**Table 5.2-1: Typical Format for a Use-Case Textual Description**

### 5.2.3 Basic and Alternate Flow of Events in Use Cases

The most important part of a use case *for generating test cases* is the flow of events. The two main parts of the flow of events are the *basic flow of events* and the *alternate flows of events*. The basic flow of events should cover what "normally" happens when the use case is performed. The alternate flows of events covers behavior of an optional or exceptional character relative to normal behavior, and also variations of

the normal behavior. You can think of the alternate flows of events as "detours" from the basic flow of events.



**Figure 5.2-1: Basic Flow of Events and Alternate Flows of Events for a Use Case**

Figure 5.2-1 represents the typical structure of these flows of events. The straight arrow represents the basic flow of events, and the curves represent alternate flows. Note that some alternate flows return to the basic flow of events, while others end the use case. Both the basic flow of events and the alternative flows should be further structured into *steps* or *subflows*, as illustrated in the following figures.

**Use Case: Register For Courses**

**Basic Flow**

**1. Logon**

This use case starts when a Student accesses the Wylie University Web site.

The system asks for, and the Student enters, the student ID and password.

**2. Select 'Create a Schedule'**

The system displays the functions available to the student. The student selects "Create a Schedule."

**3. Obtain Course Information**

The system retrieves a list of available course offerings from the Course Catalog System and displays the list to the Student.

**4. Select Courses**

The Student selects four primary course offerings and two alternate course offerings from the list of available course offerings.

**5. Submit Schedule**

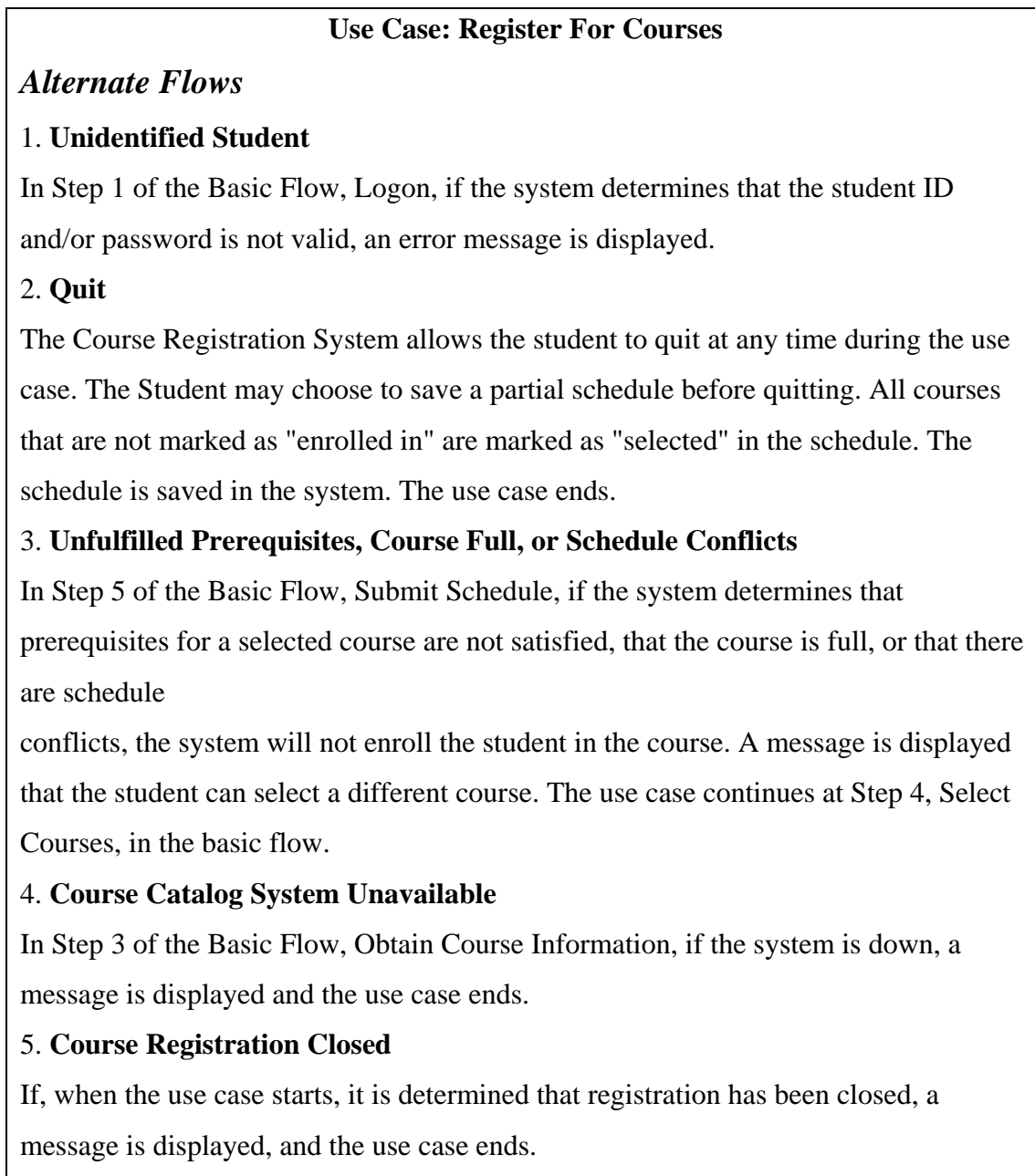
The student indicates that the schedule is complete. For each selected course offering on the schedule, the system verifies that the Student has the necessary prerequisites.

**6. Display Completed Schedule**

The system displays the schedule containing the selected course offerings for the Student and the confirmation number for the schedule.

**Figure 5.2-2: Textual Description for "Register For Courses" Basic Flow of Events**

Figure 5.2-3 shows a few alternate flows.



**Figure 5.2-3: Textual Description for “Register For Courses” Alternate Flows**

As you can see, a significant amount of detail goes into fully specifying a use case. Ideally, the flows should be written as "dialogs" between the system and the actors. Each step should explain what the actor does and what the system does in response; it should also be numbered and have a title. Alternate flows always specify where they start in the basic flow and where they go when they end.

### 5.2.4 Use-Case Scenarios

Recall that a use-case scenario is an instance of a use case, or a complete "path" through the use case. End users of the completed system can go down many paths as they execute the functionality specified in the use case. Following the basic flow would be one scenario. Following the basic flow plus alternate flow 1A would be another. The basic flow plus alternate flow 2A would be a third, and so on.

Table 5.2-2 lists all possible scenarios for the diagram shown in Figure 5.2-1, beginning with the basic flow and then combining the basic flow with alternate flows.

Scenario 1	Basic Flow			
Scenario 2	Basic Flow	Alternate Flow 1		
Scenario 3	Basic Flow	Alternate Flow 1	Alternate Flow 2	
Scenario 4	Basic Flow	Alternate Flow 3		
Scenario 5	Basic Flow	Alternate Flow 3	Alternate Flow 1	
Scenario 6	Basic Flow	Alternate Flow 3	Alternate Flow 1	Alternate Flow 2
Scenario 7	Basic Flow	Alternate Flow 4		
Scenario 8	Basic Flow	Alternate Flow 3	Alternate Flow 4	

**Table 5.2-2: Scenarios for the Use Case Shown in Figure 5.2-2**

These scenarios will be used as the basis for creating test cases.

## 5.3 Generating Test Cases

### 5.3.0 Overview

A test case is a set of test inputs, execution conditions, and expected results developed for a particular objective: for example, to exercise a particular program path or verify compliance with a specific requirement. The purpose of a test case is to identify and communicate conditions that will be implemented in test. Test cases are necessary to verify successful and acceptable implementation of the product requirements (e.g. use cases). We will describe a 3-step process for generating test cases from a fully detailed use case:

1. **For each use case**, generate a full set of use-case scenarios.
2. **For each scenario**, identify at least one test case and the conditions that will make it "execute."
3. **For each test case**, identify the data values with which to test.

### 5.3.1 Step One: Generate Scenarios

Read the use-case textual description and identify each combination of main and alternate flows -- the scenarios -- and create a *scenario matrix*. Table 5.3-1 shows a partial scenario matrix for the “Register for Courses” use case. This is a simple example with no nested alternate flows.

Scenario Name	Starting Flow	Alternate
Scenario 1 - Successful registration	Basic Flow	
Scenario 2 - Unidentified student Basic	Basic Flow	A1
Scenario 3 - User quits	Basic Flow	A2
Scenario 4 - Course catalog system unavailable	Basic Flow	A4
Scenario 5 - Registration closed	Basic Flow	A5
Scenario 6 – Cannot enroll	Basic Flow	A3

**Table 5.3-1: Partial Scenario Matrix for the Register for Courses Use Case**

### 5.3.2 Step Two: Identify Test Cases

Once the full set of scenarios has been identified, the next step is to identify the test cases. We can do this by analyzing the scenarios and reviewing the use case textual description as well. There should be at least one test case for each scenario, but there will probably be more. For example, if the textual description for an alternate flow is written in a very cursory way, like the description below,

3A. Unfulfilled Prerequisites, Course Full, or Schedule Conflicts

then additional test cases may be required to test all the possibilities. In addition, we may wish to add test cases to test boundary conditions.

The next step in fleshing out the test cases is to reread the use-case textual description and find the conditions or data elements required to execute the various scenarios. For the “Register for Course” use case, conditions would be student ID, password, courses selected, etc.

To clearly document the test cases, once again, a matrix format is useful, like the one in Table 5.3-2. Notice the top (header) row: The 1<sup>st</sup> column contains the test case ID, the 2<sup>nd</sup> column has a brief description of the test case, including the scenario being tested, and all other columns except the last one contain data elements that will be used in implementing the tests. The last column contains a description of the test case's expected output.

Test Case ID	Scenario/Condition	Student ID	Password	Courses selected	Pre-requisites fulfilled	Course Open	Schedule Open	Expected Result
RC 1	Scenario 1- successful registration	V	V	V	V	V	V	Schedule & confirmation no. displayed
RC 2	Scenario 2- unidentified student	I	N/A	N/A	N/A	N/A	N/A	Error message; back to login screen
RC 3	Scenario 3- valid user quits	V	V	N/A	N/A	N/A	N/A	Login screen appears
RC 4	Scenario 4- course registration system unavailable	V	V	N/A	N/A	N/A	N/A	Error message; back to step 2
RC 5	Scenario 5- registration closed	V	V	N/A	N/A	N/A	N/A	Error message; back to step 2
RC 6	Scenario 6- cannot enroll -- course full	V	V	V	V	I	V	Error message; back to step 3
RC 7	Scenario 6- cannot enroll -- prerequisite not fulfilled	V	V	V	I	V	V	Error message; back to step 4
RC 8	Scenario 6- cannot enroll -- schedule conflict	V	V	V	V	V	I	Error message; back to step 4

**Table 5.3-2: Test Case Matrix for the Register for Courses Use Case**

Notice that in this matrix *no data values* have actually been entered. The cells of the table contain a *V*, *I*, or *N/A*, where *V* indicates *valid*, *I* is for *invalid*, and *N/A* means that it is not necessary to supply a data value in this case.

This specific matrix is a good *intermediate step*; it clearly shows what conditions are being tested for each test case. It is also very easy to determine by looking at the *Vs* and *Is* whether you have identified a sufficient number of test cases. In addition to the "happy day" scenarios in which everything works fine, each row in the matrix should have at least one *I* indicating an invalid condition being tested. In the test case matrix in Table 5.3-2, some conditions are obviously missing -- e.g., Registration Closed -- because RC3, RC4, and RC5 each has the same combination of *Is* and *Vs*.

### 5.3.3 Step Three: Identify Data Values to Test

Once all of the test cases have been identified, they should be reviewed and validated to ensure accuracy and to identify redundant or missing test cases. Then, once they are approved, the final step is to substitute actual data values for the *Is* and *Vs*. Without test data, test cases (or test procedures) can't be implemented or executed; they are just descriptions of conditions, scenarios, and paths. Therefore, it is necessary to

identify actual values to be used in implementing the final tests. Table 5.3-3 shows a *test case matrix* with values substituted for the *Is* and *Vs* in the previous matrix. A number of techniques can be used for identifying data values – see later for a cursory outline.

Test Case ID	Scenario/Condition	Student ID	Password	Courses selected	Pre-requisites fulfilled	Course Open	Schedule Open	Expected Result
RC 1	Scenario 1- successful registration	jheumann	abc123	M101 E201 S101	Yes	Yes	Yes	Schedule & confirmation no. displayed
RC 2	Scenario 2- unidentified student	jheumann1	N/A	N/A	N/A	N/A	N/A	Error message; back to login screen
RC 3	Scenario 3- valid user quits	jheumann	abc123	N/A	N/A	N/A	N/A	Login screen appears
RC 4	Scenario 4- course registration system unavailable	jheumann	abc123	N/A	N/A	N/A	N/A	Error message; back to step 2
RC 5	Scenario 5- registration closed	jheumann	abc123	N/A	N/A	N/A	N/A	Error message; back to step 2
RC 6	Scenario 6- cannot enroll -- course full	jheumann	abc123	M101 E201 S101	Yes	M101 full	Yes	Error message; back to step 3
RC 7	Scenario 6- cannot enroll -- prerequisite not fulfilled	jheumann	abc123	M101 E201 S101	No for E201	Yes	Yes	Error message; back to step 4
RC 8	Scenario 6- cannot enroll -- schedule conflict	jheumann	abc123	M101 E201 S101	Yes	Yes	E201 and S101 conflict	Error message; back to step 4

**Table 5.3-3: Test Case Matrix with Data Values**

## 5.4 Conclusion: Putting It All Together

In [much] current practice, use cases are associated with the front end of the software development lifecycle and test cases are typically associated with the latter part of the lifecycle. By leveraging use cases to generate test cases, however, testing teams can get started much earlier in the lifecycle, allowing them to identify and repair defects that would be very costly to fix later, ship on time, and ensure that the system will work reliably.

Using the clearly-defined methodology outlined above for generating test cases, developers can simplify the testing process, increase efficiency, and help ensure complete test coverage.