Test Case Generation from UML Models

Yiwen Wang and Mao Zheng
Department of Computer Science
University of Wisconsin - La Crosse
La Crosse, WI 54601
mzheng@uwlax.edu

Abstract

Software Testing is one of the important phases in the software development life cycle. The cost of software testing is approximately 50% of the total development cost. In order to test software in an effective and an efficient manner, the test case should be generated systematically.

Unified Modeling Language (UML) is the current industrial standard used to assist software development. It is widely used to describe the requirements and the design of the software in a number of diagrams. Each diagram provides a view of the system. The class diagram provides the static configuration, while the interaction diagrams describes the dynamics of the system behavior.

This paper presents a methodology to generate the test case from a design level class diagram and an interaction diagram. It is in the category of model-based testing. It is assumed that the given model is correct and the goal of the testing is to check if the implementation conforms to the given model. A car rental example is used to illustrate the test case generation. The test adequacy criteria used in this paper are the coverage for the model elements, also called the building blocks in the class diagrams and the interaction diagrams. These criteria are based on the same premise for the underlying code testing criteria to help define testing objectives. The class diagram defines the object configuration and the interaction diagram determines the method sequence in the testing. The generated test cases are compared with a few other UML model-based test case generation methodologies; they are able to meet the required test adequacy criteria better.

1 Introduction

Software testing refers to execute a program with a set of test cases and compare the actual output with the expected results. It is one of the most widely used techniques to improve software quality. It is also considered to be one of the important phases in the software development life cycle. Today the cost on software testing is approximately 50% of the total development cost. In order to test software in an effective and an efficient manner, the test case should be generated systematically.

A *testing criterion* is a rule or a collection of rules that impose testing on a set of test cases. A *testing technique* guides the tester through the testing process by including a testing criterion and a process for creating test cases values [5]. Testers measure the extent to which a criterion is satisfied in terms of coverage, which is the percent of testing requirements that are satisfied. Testing criteria can also be used to determine when testing should stop: testing can stop when tests that satisfy all the criteria have been carried out successfully. There are various ways to classify adequacy criteria. One of the most common is by the source of information used to specify testing requirements and in the measurement of test adequacy. Hence, an adequacy criterion can be specification-based, design-based, or program-based. The test adequacy criteria used in this paper are the coverage for the design model elements, also called the building blocks in the class diagrams and the interaction diagrams.

This paper discusses the technique using the test adequacy criteria mentioned above to generate a test case from the UML design class and interaction diagrams. The assumption is that the design is a valid representation of the system's desired behavior. The tests generated will primarily evaluate whether the implementation correctly reflects the design.

The Unified Modeling Language (UML) [8] is an Object Management Group (OMG) Object-Oriented (OO) modeling language standard that is gaining widespread use in the software development industry. Modeling a large, complex system can result in a system model that consists of a variety of diagrams presenting different views of the model. The class diagram provides the static configuration, while the interaction diagrams describes the dynamics of the system behavior.

This paper presents a methodology to generate the test case from a design level class diagram and an interaction diagram. A car rental example is used to illustrate the test case generation.

2 UML Class and Interaction Diagrams

2.1 Class Diagram

A UML class diagram consists of classes and the relationship among the classes. There are three types of relationships: association, generalization/specialization and aggregation. Classes represent the problem concepts; associations model the semantic relationships between problem concepts. Generalization/Specialization, at this concept class diagram level, describes a categorization from the bottom up approach. The class that defines common concepts will be the generalization of subclasses. The aggregation is one kind of association.

Paper [1] extracts three building blocks from the UML class diagram: association and its multiplicity, generalization/specialization and class attributes. It also defines related coverage criteria: association-end multiplicity (AEM) criterion, generalization (GN) criterion and class attributes (CA) criterion. Below is the table of the test criteria.

Association-end multiplicity (AEM) criterion	Given a test set *T* and system model *SM*, *T* must cause each representative multiplicity-pair in *SM* to be created
Generalization (GN) criterion	Given a test set *T* and a system model *SM*, *T* must cause every specialization defined in a generalization relationship to be created
Class attribute (CA) criterion	Given a test set *T*, a system model *SM*, and a class *C*, *T* must cause a set of representative attribute value combinations in each instance of class *C* to be created

Table 1: Test Criteria for Class Diagrams [1]

All above criteria are expressed in terms of representative values. In order to establish the set of representative values, a form of category-partition [7] adapted to UML diagrams is used. Using this method, the value domain is partitioned into equivalence classes, and one value from each class is selected for the set of representative values. The partitioned can be determined by either Knowledge-based partition: use knowledge of the problem domain; or by default partition: use minimum, non-boundary and maximum values. For example, given a multiplicity m..n, the minimum value partition is {m}, a non-boundary partition is {m+1, …, n-1}, and the maximum value partition is {n}. After obtaining the set of representative values, we create the Cartesian Product of each value set, then identify valid and invalid set. The default partition is used in this research work.

2.2 Interaction Diagram

Interaction diagram describes the intra-object communications. It includes a collaboration diagram and a sequence diagram. A collaboration diagram characterizes how objects interact to achieve a behavioral goal. A sequence diagram contain the same interaction information, but in a different format. In this paper, a collaboration diagram is used to depict structure and interactions among objects in the system. All message paths criterion (AMP) [1] is defined to exercise all the message sequences in the collaboration diagram.

All message paths (AMP) criterion Given a test set *T* and collaboration diagram *CD*, *T* must cause each possible message path (sequence of message numbers) in *CD* to be taken at least once.

3 Test Case Generation

Testing methods for UML design differ depending on the testing criteria used. In this paper, we assume the class diagram criteria need to be met as well as the All Message Paths (AMP) criterion in the collaboration diagram. This assumption comes naturally from the graph-based criteria.

From the class diagram criteria, we can define a set of target configuration. The test case is recorded using the format <<*sequence_of_signals*, *start_configuration*, *prefix*>> [1]. A configuration is a structure of objects that satisfies the constraint expressed in the class diagrams. A configuration includes 1) the class objects and the links that exist at a given time, and 2) the value of each attribute in each object in the configuration. The *start_configuration* is the configuration on which the test is started. The *prefix* is a sequence of signals that can be used to take the system from an initial configuration to the *start_configuration*. Once the system is in the chosen *start_configuration*, a *sequence_of_signals* is applied to run the test. The *sequence of signals* is derived from the message sequence in a collaboration diagram. Execution of a test case will result in a trace of configurations and the sequence of signals generated as a result of the test input sequence.

4. Case Study

To illustrate the testing methodology described, we are using a car rental example to illustrate the test case generation.

A car rental company has several different types of vehicles (cars, trucks, SUVs etc.). The rate for a vehicle varies with respect to its type as shown in Table 2. The actual rental charge includes the rate multiplied by the number of days it is rented, plus additional charges for the miles the vehicle is driven, if applicable. A customer who wants to rent a car may ask for a specific type of vehicle or may choose one of the available vehicles.

Vehicle Type	Option	Rate per day
Compact/Mid size	2-Door	$15.00
Compact/Mid size	4-Door	$20.00
Standard/Large/Family size	Without Child Seat	$30.00
Standard/Large/Family size	With Child Seat	$40.00
Premium/Luxury		$55.00
SUV/Minivan	Without TV	$55.00
SUV/Minivan	With TV	$65.00
Convertible		$65.00

Table 2: Rental Charge

For simplicity, we are assuming a customer can rent only one vehicle at a time and all payments must be made through credit cards only. This system is intended to keep track of all the rental records.

The class diagram for the car rental company is shown below in Figure 1.

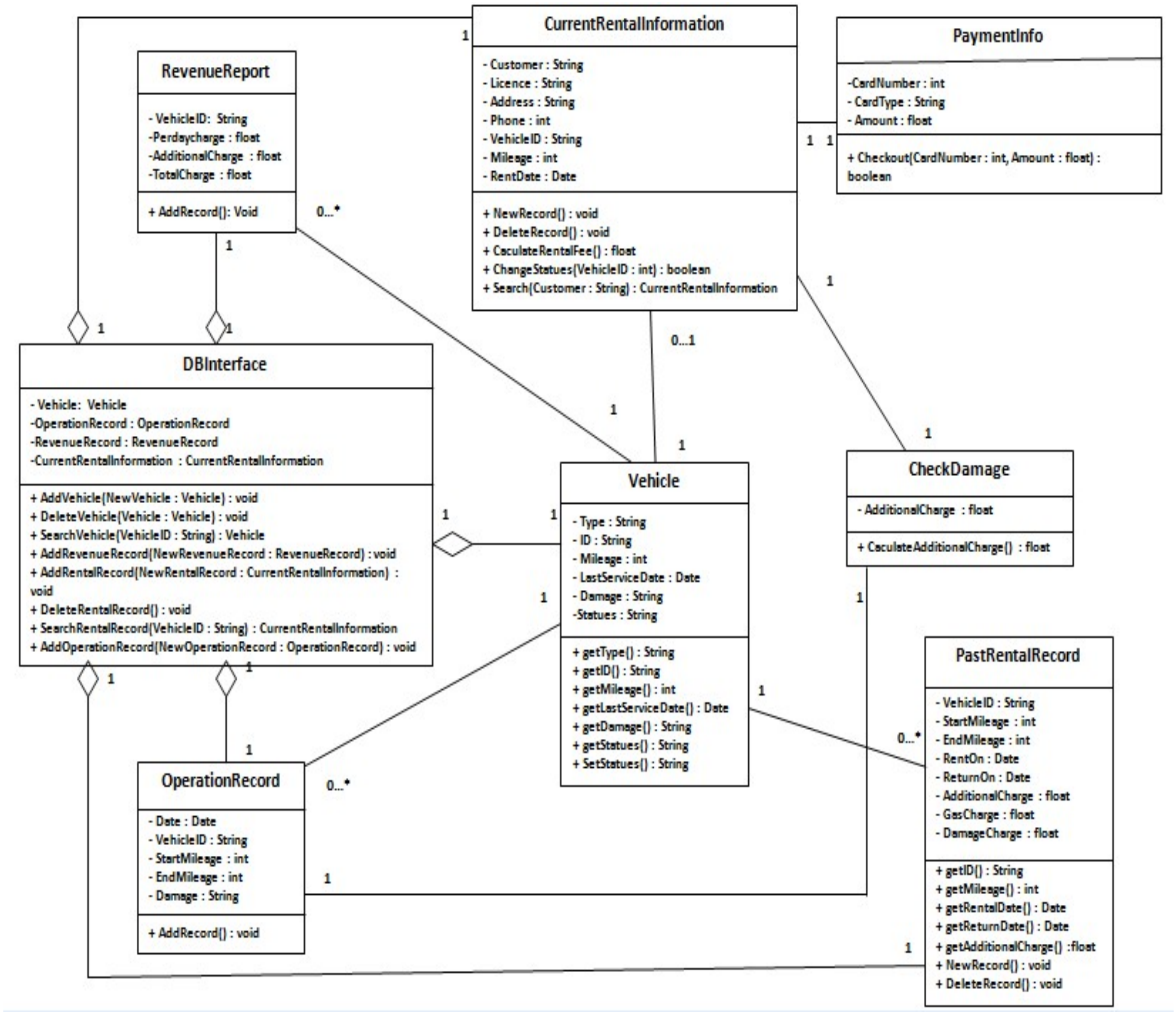

Figure 1: Car Rental Company’s Class Diagram

The link between the Vehicle class and CurrentRentalInformation class is a “1 to 0..1” association. A vehicle has two rental statuses in the system: rented out or not. That implies it has either no or only one current record. A closer look is shown in Figure 2 below.

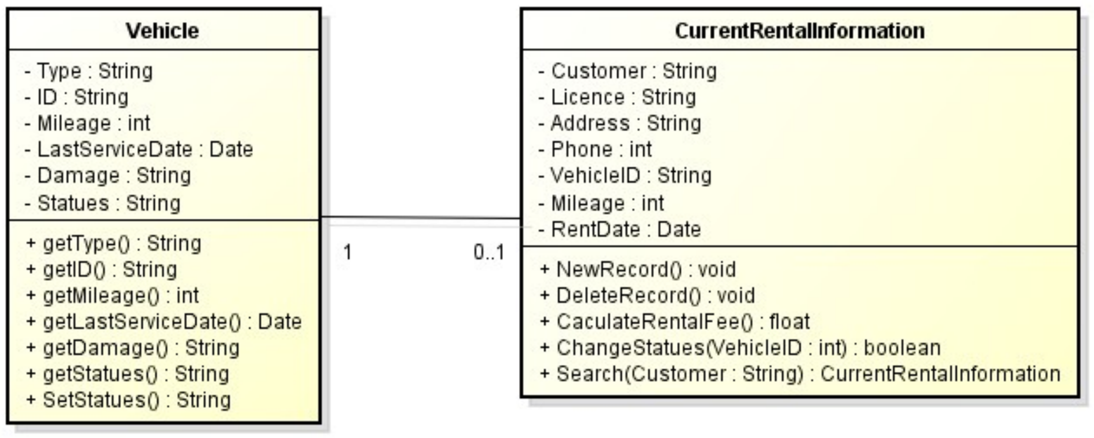

Figure 2 : The link between classes Vehicle and CurrentRentalInformation

Table 3 below are three test cases generated based on the Association-end Multiplicity criterion. It tested whether the same vehicle can be rented with no current rental record (*invalid input*), with one current rental record (*valid input*) and with two current rental records (*invalid input*). Here we also used the information from an association's multiplicity to identify valid and invalid inputs. The *Coverage* column shows the coverage elements from the class diagram criteria that were actually exercised during the test.

No	Parameters	Vehicle-config	Information-config	Vehicle Type	Coverage	Expect Result	Actual Result
1	**V-VehicleID** MHJ689HDG683JG7 **I-Info**	**True**	**False**	**Compact**	**AEM:Vehicle(1)-PastRentalRecord(0) CA:Rental information incorrect**	**No Record Found**	**No record**
2	**V-VehicleID** MHJ689HDG683JG7 **V-Info**	**True**	**True**	**Compact**	**AEM:Vehicle(1)-PastRentalRecord(1) CA:Rental information correct**	**Have one correct rental information**	**Record add correctly**
3	**V-VehicleID** MHJ689HDG683JG7 **V-Info**	**True**	**True**	**Compact**	**AEM:Vehicle(1)-PastRentalRecord(2) CA:Rental information correct**	**Can't add two current rental record for one vehicle at the same time**	**Error Information, can't add the rental record**

Table 3: The Test Case Generated Based on the AEM criterion

The test execution result indicated the implementation confirmed the expected result.

The collaboration diagram for "Return a Vehicle" is shown in Figure 2.

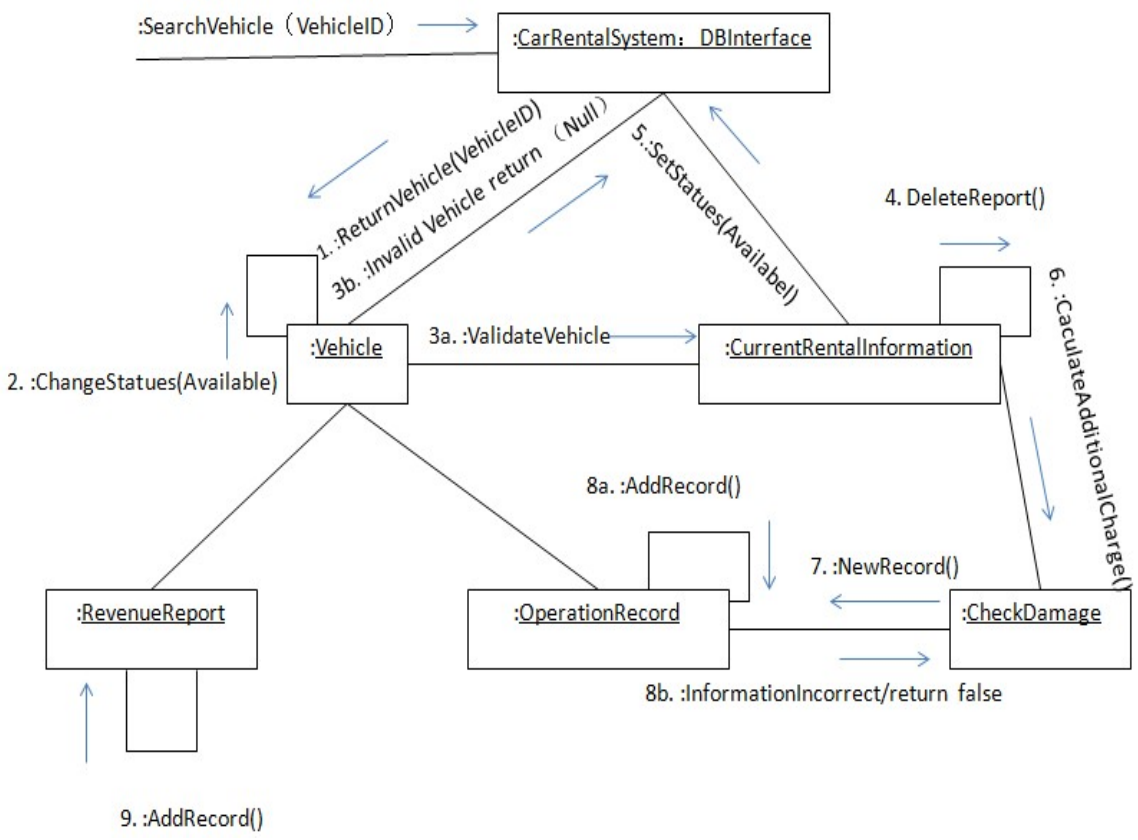
:SearchVehicle（VehicleID）
:CarRentalSystem：DBInterface
1. :ReturnVehicle(VehicleID)
3b. :Invalid Vehicle return（Null）
5.:SetStatues(Availabel)
4. DeleteReport()
2. :ChangeStatues(Available)
:Vehicle
3a. :ValidateVehicle
:CurrentRentalInformation
6. :CaculateAdditionalCharge()
8a. :AddRecord()
7. :NewRecord()
:RevenueReport
:OperationRecord
:CheckDamage
8b. :InformationIncorrect/return false
9. :AddRecord()

Figure 2: The Collaboration Diagram for "Return a Vehicle"

Based on the class diagram criteria and the target configuration using "return a vehicle", we generated the test cases below in Table 3.

No	Parameters	Vehicle-config	Information-config	Additional charges	Vehicle Type	Coverage	Expect Result	Actual Result
1	**I-VehicleID I-Info**	**False**	**False**	**NA**	**NA**	**AEM:Vehicle(0)-PastRentalRecord(0) CA:Rental information incorrect, no additional charge**	**Invalid VehicleID no record found**	**No record found with the invalid VehicleID**
2	**V-VehicleID** NG527GK D648LOD7 **I-Info**	**True**	**False**	**NA**	**Convertible**	**AEM:Vehicle(1)-PastRentalRecord(0) CA:Rental information incorrect, no additional charge**	**Cannot add the past rental information with the incorrect rental information data.**	**Cannot add the past rental information to the record**
3	**V-VehicleID** NG527GK D648LOD7 **V-Info**	**True**	**True**	**NA**	**Convertible**	**AEM:Vehicle(1)-PastRentalRecord(1) CA:Rental information correct, no additional charge**	**Add the past rental record successfully.**	**Can add the past rental information for the input VehicleID.**
4	**V-VehicleID** NG527GK	**True**	**True**	**Yes**	**Convertible**	**AEM:Vehicle(1)-PastRentalR**	**Add the past rental record successfully.**	**Can add the past rental information for the input**

	D648LOD7 **V-Info**					**ecord(2) CA:Rental information correct, has additional charge**		**VehicleID.**
5	**V-VehicleID** NG527GK D648LOD6 **V-Info**	**True**	**True**	**NA**	**Convertible**	**AEM:Vehicle(2)-PastRentalRecord(1) CA:Rental information correct, no additional charge**	**Add the past rental record successfully.**	**Can add the past rental information for the input VehicleID.**
6	**V-VehicleID** MHJ689H DG683JG7 **I-Info**	**True**	**False**	**NA**	**Compact**	**AEM:Vehicle(3)-PastRentalRecord(0) CA:Rental information incorrect, no additional charge**	**No Past Rental information matched with the input VehicleID.**	**No Past Rental Record has been founded.**
7	**V-VehicleID** MHJ689H DG683JG7 **V-Info**	**True**	**True**	**NA**	**Compact**	**AEM:Vehicle(3)-PastRentalRecord(1) CA:Rental information correct, no additional charge**	**Can add one Past Rental Information with the input vehicle id**	**Add one Past Rental Record successfully.**
8	**V-VehicleID** MHJ689H DG683JG7 **V-Info**	**True**	**True**	**Yes**	**Compact**	**AEM:Vehicle(3)-PastRentalRecord(2) CA:Rental information correct, has additional charge**	**Can add the Second Past Rental Information with the input vehicle id**	**Have two past rental records with the input VehicleID.**
9	**V-VehicleID** MHJ689H DG683JG7 **V-Info**	**True**	**True**	**NA**	**Compact**	**AEM:Vehicle(3)-PastRentalRecord(3) CA:Rental information correct, has additional charge**	**Can add the third Past Rental Information with the input vehicle id**	**Have three past rental records with the input VehicleID.**
10	**V-VehicleID** MHJ689H DG683JG8 **V-Info**	**True**	**True**	**NA**	**Compact**	**AEM:Vehicle(4)-PastRentalRecord(1) CA:Rental information correct, no additional charge**	**Add the past rental record successfully.**	**Can add the past rental information for the input VehicleID.**

Table 3: Test Cases Generated for "Return a Vehicle"

Table 4 below shows the extent the above test cases satisfy the AMP criterion for the collaboration diagram in Figure 2. The set of test cases in Table 3 are able to satisfy AMP criterion completely.

Path coverage	Test Case
1,2,3b	1
1,2,3a,4,5,6,7,8b,9	1,2,6
1,2,3a,4,5,6,7,8a,9	3,4,5,7,8,9,10

Table 4: Path Coverage

This case study also showed that one test case is able to cover quite a few coverage criteria.

5 Conclusion

There are a lot of development to support the test generation from UML design models, including sequence diagram, state diagram and activity diagram in addition to class and collaborations. The various techniques are all based on the graph criteria. However, an intermediate graph from the UML diagrams is often required to derive the test case. The testing technique described in this paper is based on [1], and the test case generation is directly from model elements, not intermediate graph. However, the original paper mainly discussed using this technique to evaluate the design itself. We used the generated test cases to evaluate the implementation's conformance with the system models. We also adopted the category partition approach to get the function units, then for each function unit, generate test cases from class diagram criteria. The method sequence from the interaction diagram is used to generate sequence of the signals in the test case. The generated test set is also able to meet AMP criterion.

For the same car rental problem, we compared the generated test cases with the test set using the testing techniques in [3] and [4], the results showed the test case generated from model elements directly are able to satisfy the all required graph coverage criteria discussed in those approaches.

References

[1] Andrew, R. France, S. Ghose, G. Craig. "Test Adequacy Criteria for UML Design Models". *Journal of Software Testing, Verification and Reliability* 13 : 95-127, 2003

[2] Sudipto Ghosh, Robert France, Conrad Braganza, Nilesh Kawane. "Test Adequacy Assessment for UML Design Model Testing", *14th International Symposium on Software Reliability Engineering (ISSRE 2003)*, pp.332-343; ISSN: 1071-9458, 2003

[3] D. Kundu, D. Samanta. "A Novel Approach to Generate Test Case from UML Activity Diagrams", *Journal of Object Technology*, Vol. 8 – No. 3, May-June 2009.

[4] S. K. Swain, D. P. Mohapatra. "Test case generation from UML sequence and activity models." *International Journal of Computer Applications (0975-8887)* Volume 6 – No. 8, September 2010

[5] A.Abdurazik and J.Offutt, "Using UML Collaboration Diagrams for Static Checking and Test Generation", *3rd International Conference on the UML*, pp.383-395, Oct, 2000.

[6] J. Offutt, Shaoying Liu, Aynur Abdurazik and Paul Ammann: "Generating test data from state-based specifications", *Software Testing, Verification and Reliability Softw. Test. Verif. Reliab. 2003*; 13:25-53 (DOI:10.1002/STVR.264).

[7] A. J. Offutt and A. Irvine. "Testing Object-Oriented Software Using the Category-Partition Method". In *Proceedings of the 17th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS USA) 1995*, pages 293-304, Santa Barbara, California, August 1995.

[8] The Object Management Group. OMG Unified Modeling Language Specification. Version 2.0, OMG, 2004.