



NUI MAYNOOTH

Ollscoil na hÉireann Má Nuad

National University of Ireland, Maynooth

MAYNOOTH, CO. KILDARE, IRELAND.

DEPARTMENT OF COMPUTER SCIENCE,

TECHNICAL REPORT SERIES

**A Survey of UML-Based Coverage Criteria for Software
Testing**

Jacqueline A. McQuillan and James F. Power

NUIM-CS-TR-2005-08

A Survey of UML-Based Coverage Criteria for Software Testing

Jacqueline A. McQuillan and James F. Power

Department of Computer Science
National University of Ireland, Maynooth
Co. Kildare
Ireland

{jmcq, jpower}@cs.nuim.ie

Abstract: The Unified Modeling Language (UML) is a standard notation that can be used to model object oriented software systems. With the growing adoption of UML by the software development industry and academia, researchers have begun to investigate how it can be used in the testing phase of the software development process. Several approaches to software testing have been proposed in which test requirements and coverage criteria are derived from UML models. This report introduces and analyses the various UML-based coverage criteria that exist in the software testing literature.

Key words: UML, software testing, coverage criteria, class diagram, interaction diagram, statechart diagram

1 INTRODUCTION

Software testing is a vital part of the software development process. It can be used for the purposes of quality assurance, reliability estimation and verification and validation [20]. However, software testing is extremely costly and time consuming. Studies indicate that more than 50% of the cost of software development is devoted to testing [20]. Therefore, there is a need for effective testing strategies.

There are numerous techniques for software testing in the literature [5, 24]. Due to the increased use of the object oriented (OO) paradigm, several new testing strategies have been specifically proposed for OO software [6, 13]. Furthermore, with the increasing use of the Unified Modeling Language (UML) to model OO systems, researchers have begun investigating how the UML can be used in the testing phase of the software development process. Consequently, several UML-based approaches to software testing have been proposed [8, 21]. In these approaches, test requirements and coverage criteria are derived from UML models.

In this report, a survey of the research on UML-based coverage criteria is presented. To facilitate a comparison of the criteria an attempt is made to outline a formal definition for each of the UML-based coverage criteria found in the literature. It is important to ensure that the coverage criteria used during testing are effective. The effectiveness of a coverage criterion is a measure of how likely the criteria is to detect the presence of a fault. Experimental evaluation is necessary to determine the effectiveness of coverage criteria. In surveying the UML-based criteria, the experimental studies investigating the effectiveness of the criteria are reviewed. This report also attempts to examine the studies that have been carried out to compare the various criteria in terms of the coverage they provide.

The remainder of this report is organised as follows. Section 2 discusses software testing and coverage criteria and their relationship with the UML. In section 3, we present a survey of UML-based coverage criteria. Finally, section 4 summarises and concludes the report.

2 TESTING AND UML

In this section software testing and test coverage criteria are examined. Also, the UML is discussed from a testing point of view.

2.1 Software Testing

Software testing is an important and integral part of the software development process. It is used to reveal bugs in a system, to assure that the system complies with its specification and to verify that the system behaves in the intended way. Various definitions have been presented for software testing [5, 6]. For example, Myers [24] defines it as:

“... the process of executing a program [or system] with the intent of finding errors.”

For our purposes, we will consider software testing as the process of determining if the observed behaviour of a system corresponds with the expected behaviour of the system. This process involves executing the system on a set of inputs and determining if the actual behaviour of the system corresponds to the expected behaviour. These inputs to the system are known as test cases and a collection of test cases is referred to as a test set or a test suite.

One important aspect of software testing is deciding when enough testing has been done. How do you decide if a test set is adequate? This question was first addressed by Goodenough and Gerhart [17] when they considered the idea of a test adequacy criterion, that is, a criterion that defines what makes an adequate test. A test adequacy criterion is a rule or a set of rules that impose requirements on a test set. Adequacy criteria play an important role in the testing process. They can be used as a stopping rule. Testing stops when enough test cases have been produced to satisfy the criteria. They can also be used as a measurement of test quality. A measure of adequacy is associated with a test set, therefore

different test sets can be compared in terms of their adequacy measurement. Adequacy criteria also provide a basis for deciding what test cases to use during testing, making it more likely that faults will be found in the system. Coverage can be used to measure the extent to which an adequacy criterion is satisfied. The percentage of requirements (as specified by the adequacy criterion) that are satisfied by a test set can be used as an adequacy measurement. Therefore coverage criteria are a type of adequacy criteria that specify the percentage of requirements that must be covered.

2.2 UML as a Test Model

In OO software the complexity of the software is now not only associated with the functions and procedures but also with how the procedures and classes are connected and how they communicate. Many traditional testing techniques exist for procedural software. Clearly not all these techniques are applicable or effective for testing OO software. Thus, work has been carried out to develop new techniques for OO software. Furthermore, research has been conducted to develop testing techniques that are based on pre-code artifacts such as the UML.

The UML is a language for specifying, visualising, constructing and documenting the artifacts of software systems. It is the result of the integration of the concepts of Booch, Jacobson *et al.*, and Rumbaugh *et al.* In 1997, the UML was adopted by the Object Management Group (OMG) as a standard for modelling OO systems. The UML provides a variety of diagrams that can be used to present different views of an OO system at different stages of the development life cycle [19]. Testing techniques that are based on the UML involve the derivation of test requirements and coverage criteria from these UML diagrams. A detailed account of these techniques with emphasis on the coverage criteria is presented in the following section.

3 UML-BASED COVERAGE CRITERIA

This section introduces and analyses various coverage criteria based on UML diagrams. It consists of several subsections, each one is devoted to each of the different UML diagrams.

3.1 Class Diagram Criteria

A class diagram shows the static structure of a system. It identifies all the entities, along with their attributes, in the system and specifies the relationships between the entities [7]. These relationships or associations are represented by links among the entities. Multiplicity specifies the range of values for how many entities at one end of an association can be associated with a single entity at the other end. Andrews *et al.* propose the association-end multiplicity (AEM) criterion which requires that for each association in the class diagram a set of representative multiplicity pairs are created. The set of multiplicity pairs that

are to be covered are derived from the class diagram using a modified form of category-partition testing [26]. This involves the partitioning of the value domain of the multiplicity into equivalence classes and the selection of a single value from each class [3]. For each end of the association a set of possible multiplicity values are selected in this way. Each value from the first set is combined with each of the values from the second set, thus producing a set of multiplicity pairs. The association-end multiplicity criterion can be defined as follows:

Definition (Association-End Multiplicity (AEM) Criterion). A test set T satisfies the association-end multiplicity criterion if and only if for each representative multiplicity-pair p in a system model there exists t in T such that t causes p to be created.

Table 1: Coverage Criteria based on Class Diagrams

<i>Author</i>	<i>Criterion</i>
Andrews <i>et al.</i>	Association-End Multiplicity (AEM)
Andrews <i>et al.</i>	Generalisation (GN)
Andrews <i>et al.</i>	Class Attribute (CA)

One other coverage criteria proposed by Andrews *et al.* [3] is the generalisation (GN) criterion. Generalisation is typically implemented as inheritance. A generalisation/specialisation relationship is one in which objects of the specialised element (child) are substitutable for objects of the generalised element (parent) [7]. The generalisation criterion requires that testing cause each generalisation/specialisation relationship (or specialisation element) to be created at least once.

Definition (Generalisation (GN) Criterion). A test set T satisfies the generalisation criterion if and only if for every specialisation s defined in a generalisation relationship in a system model there exists t in T such that t causes s to be created.

The final coverage criterion based on class diagrams defined in [3] is the class attribute (CA) criterion. This criterion requires coverage of a set of attribute value combinations for each class in the class diagram. The category partition method is used to produce a set of possible values for each attribute in a class. Elements from each of these sets are combined to create a set of attribute values for each class.

Definition (Class Attribute (CA) Criterion). A test set T satisfies the class attribute criterion if and only if for each attribute value combination a , for each class c in a system model there exists t in T such that t causes an instance of c to be created with the values in a .

Pilkalns *et al.* [27] propose a systematic technique for testing and exercising UML design models. The approach uses information from both class diagrams and sequence diagrams to derive and execute test cases. The coverage criteria defined by Andrews *et al.* in [3] are used to evaluate these test cases. They demonstrate how the approach may be used with a simple example but present no results on how effective the approach is for finding faults.

3.2 Sequence Diagram Criteria

A sequence diagram is an interaction diagram that depicts the exchange of messages between a set of objects. In a sequence diagram the emphasis is on the time ordering of the messages [7].

Table 2: Coverage Criteria based on Sequence Diagrams

<i>Author</i>	<i>Criterion</i>
Basanieri and Bertolino	All-Paths-Coverage
Binder	Condition/Iteration Coverage
Briand and Labiche	All-Paths-Coverage
Fraikin and Leonhardt	All-Paths-Coverage
Rountev <i>et al.</i>	All-IRCFG-Paths
Rountev <i>et al.</i>	All-RCFG-Paths
Rountev <i>et al.</i>	All-RCFG-Branches
Rountev <i>et al.</i>	All-Unique-Branches

A start-end message path in a sequence diagram is a sequence of messages that begins with an externally generated event and ends with the production of a response that satisfies this event. A requirement of adequate testing based on sequence diagrams is that all the start-end message paths in the diagram are covered by test executions. This can be referred to as the all-paths coverage criterion and can be defined as follows:

Definition (All-Paths Coverage Criterion). A set of message paths P satisfies the all-paths coverage criterion if and only if P contains all start-to-end message paths in a sequence diagram.

Several authors propose the use of the all-paths coverage criterion in their approaches to testing. Basanieri and Bertolino [4] present an approach to integration testing that uses the category partition method [26] and sequence diagrams to generate test cases. Their approach requires coverage of all message paths in the sequence diagrams. Briand and Labiche [8] also make use of the all-paths coverage criterion in their approach to system testing. They describe the TOTEM methodology that uses use cases, their corresponding sequence and collaboration diagrams, class diagrams and the Object Constraint Language (OCL) used within these diagrams to produce test cases. Each use case consists of one or more scenarios; each scenario corresponds to a start-to-end message path in the corresponding sequence diagram. The test cases

produced are required to cover all such message paths (scenarios). Another approach to testing based on sequence diagrams is described in [14] by Fraikin and Leonhardt. This approach requires coverage of all possible start-to-end message paths in a set of related sequence diagrams. They describe a tool SeDiTeC that implements this testing approach.

Rountev *et al.* [29] define the interprocedural restricted control-flow graph (IRCFG) which depicts the set of message sequences in a sequence diagram. An IRCFG consists of a set of nodes and edges that connect these nodes. Each node contains a restricted control flow graph (RCFG) which corresponds to a particular method call in the sequence diagram; the RCFG shows the sequence of messages that are invoked in response to the method call. This IRCFG is used to define a set of coverage criteria for sequence diagrams. They define their version of the all-paths criterion and refer to it as the all-IRCFG-paths criterion. An IRCFG path is a complete start to end traversal of the IRCFG and corresponds to a complete start-to-end message path in a sequence diagram. The all IRCFG-paths criterion requires coverage of all such paths.

In practice, full all-paths coverage cannot always be achieved because of the possibility of an infinite number of start-to-end message paths. In all of the above approaches Rountev *et al.* [29] are the only authors to consider this problem. Their criteria assume that no cycles exist in the IRCFG thus preventing this possibility. However, it is possible for the number of start-to-end message paths to be finite and very large, making it still infeasible to achieve this criterion.

Binder [6] presents a testing technique that considers a subset of all start-to-end message paths in a sequence diagram. The technique involves converting the sequence diagram to a control flow graph (CFG) and deriving test cases from this graph. The coverage criteria he uses provides branch/iteration coverage and is an extension of the traditional branch coverage criterion which can be defined as follows:

Definition (Branch Coverage Criterion). A set of paths P satisfies the branch coverage criterion if and only if for all edges e in the control flow graph, there is at least one path p in P such that p contains the edge e .

The test cases produced by the technique must satisfy the above criterion and must provide the following iteration coverage:

Definition (Iteration Coverage Criterion). Given a test set T and Sequence Diagram SD , for each loop L in SD , T must cause the loop to be either bypassed or taken for the minimum number of iterations, to be taken at least once and to be taken for the maximum number of iterations.

Other coverage criteria based on the IRCFG are presented by Rountev *et al.* [29]. Recall, an IRCFG consists of a set of RCFGs and a set of edges connecting the RCFGs. An RCFG corresponds to a particular method call in the sequence

diagram and shows the sequence of messages that are invoked in response to the method call. Rountev *et al.* [29] defines the all-RCFG-paths criterion that considers all RCFG paths in an IRCFG. An RCFG path is a sequence of RCFG nodes within some RCFG R, beginning at the start node of R and finishing at the end node of R.

Definition (All-RCFG-Paths Criterion). A set of IRCFG paths P satisfies the all-RCFG-paths coverage criterion if and only if P contains all RCFG paths.

The following criteria is also proposed and requires coverage of all RCFG edges rather than all RCFG paths.

Definition (All-RCFG-Branches Criterion). A set of IRCFG paths P satisfies the all-RCFG-branches coverage criterion if and only if for all edges e in each RCFG, there is at least one path p in P such that p contains the edge e .

According to Rountev *et al.* [29] this criterion is equivalent to the one proposed by Binder [6]. Finally, a weaker version of this criterion called the all-unique-branches criterion is proposed. Since the same RCFG may appear more than once in the IRCFG, each RCFG edge can appear more than once in the IRCFG. These repeated RCFG edges are considered *equivalent*. The all-unique-branches criterion requires that only one of these equivalent edges be covered by an IRCFG path, regardless of how many times it appears in the IRCFG.

Definition (All-Unique-Branches Criterion). A set of IRCFG paths P satisfies the all-unique-branches coverage criterion if and only if for each edges u (up to equivalence) there is at least one path p in P such that p contains the edge u .

With the exception of Rountev *et al.* [29], the authors do not provide formal definitions for the above criteria. Instead, the coverage is specified as a more general testing requirement.

A search of the literature revealed no empirical evidence demonstrating the fault revealing capabilities of the criteria described above. The SeDiTeC tool has been used in two industry projects [14]. But there have been no reports on the outcome of the tool in these projects.

Rountev *et al.* [29] compare the proposed criteria according to testing effort. The testing effort is measured by the minimum number of test cases needed to satisfy each criterion. They discover that in order to satisfy the all-paths coverage criterion, a large amount of test cases are required. This means that it is infeasible to achieve such coverage. It is also found that the remaining criteria require less test cases and thus less testing effort.

No studies could be found about how these sequence diagram criteria compare with other UML-based coverage criteria.

3.3 Communication Diagram Criteria

A communication diagram was formerly called a collaboration diagram in UML 1.x [18]. There have been no criteria defined explicitly for communication diagrams but several exist for collaboration diagrams. Like a sequence diagram, a collaboration diagram is an interaction diagram that illustrates the exchange of messages between a set of objects. However, in a collaboration diagram the emphasis is on the structural organisation of the objects that participate in an interaction [7].

Table 3: Coverage Criteria based on Communication Diagrams

<i>Author</i>	<i>Criterion</i>
Abdurazik and Offutt	Message Sequence Path Coverage
Andrews <i>et al.</i>	Condition Coverage (Cond)
Andrews <i>et al.</i>	Full Predicate Coverage (FP)
Andrews <i>et al.</i>	Each Message on Link (EML)
Andrews <i>et al.</i>	All Message Paths (AMP)
Andrews <i>et al.</i>	Collection Coverage (Coll)
Wu <i>et al.</i>	All Transitions Coverage
Wu <i>et al.</i>	All Message Sequence Paths Coverage
Wu <i>et al.</i>	All Content-Dependence Relationships Coverage

Various coverage criteria based on collaboration diagrams have been proposed. One such criterion is the all message sequence paths criterion and requires all message sequence paths in a collaboration diagram to be covered by test executions. It is not always feasible to achieve full all message sequence paths coverage as a collaboration diagram may contain an infinite or very large number of message sequence paths. This criterion can be defined as follows:

Definition (All Message Sequence Paths Criterion). A test set T satisfies the all message sequence paths criterion if and only if for each message sequence path p in a collaboration diagram there exists t in T such that t causes p to be executed.

Wu *et al.* [31] present a set of criteria that can be used for integration testing of component-based software. They propose two types of relationships that can exist in a component-based system and describe ways in which to derive these relationships from interaction diagrams and statechart diagrams. One such relationship is the content-dependence relationship and an explanation of this relationship can be found in [31]. They define the all message sequence paths criterion along with the all transitions and all content dependence relationship coverage criteria.

Definition (All Transitions Coverage Criterion). A test set T satisfies the all transitions coverage criterion if and only if for each transition tr in a collaboration diagram there exists t in T such that t tests tr .

Definition (All Content Dependence Relationships Coverage Criterion). A test set T satisfies the all content dependence relationships coverage criterion if and only if for each content dependence relationship r derived from a collaboration diagram there exists t in T such that t tests r .

Abdurazik and Offutt [1] propose a criterion which requires that for each collaboration diagram in a specification there is at least one test case that causes *the* message sequence path of the collaboration diagram to be executed in the implementation. The authors state that this criterion is defined under the assumption that there is one collaboration diagram per operation and the implementation of an operation conforms to the collaboration. It is possible for a collaboration diagram to contain more than one message sequence path. This possibility is not considered by the authors. Their criterion is defined under the assumption that each collaboration diagram contains only *one* message sequence path and this assumption is not explicitly stated.

The second set of criteria defined by Andrews *et al.* [3] are those that relate to collaboration diagrams. They define the condition coverage (Cond) criterion. In a collaboration diagram it is possible to specify that messages may only be executed under certain circumstances. This is achieved by associating a condition with the message. The condition coverage criterion requires that each condition in the collaboration diagram evaluate to both TRUE and FALSE. Therefore there must exist a test case that causes the condition to evaluate to TRUE and one to cause it to evaluate to FALSE.

Definition (Condition Coverage (Cond) Criterion). A test set T satisfies the condition coverage criterion if and only if for each condition c in a collaboration diagram there exists $t1$ in T such that $t1$ causes c to evaluate to TRUE and there exists $t2$ in T such that $t2$ causes c to evaluate to FALSE.

It is possible for a condition to be made up of more than one clause. These clauses are combined using Boolean operators. The full predicate coverage (FP) criterion requires that for each condition in the collaboration diagram, each clause must evaluate to TRUE and to FALSE while all other clauses in the condition take on values such that the condition and the clause being tested evaluate to the same result.

Definition (Full Predicate Coverage (FP) Criterion). A test set T satisfies the full predicate coverage criterion if and only if for each clause c in each condition in a collaboration diagram there exists $t1$ in T such that $t1$ causes c to evaluate to TRUE and there exists $t2$ in T such that $t2$ causes c to evaluate to FALSE while all other clauses in the condition have values such that the value of the condition will always be the same as the clause under test.

Another criteria defined in [3] is the each message on link criterion which requires that for each link connecting objects in a collaboration diagram that each message on that link will occur at least once.

Definition (Each Message on Link (EML) Criterion). A test set T satisfies the each message on link criterion if and only if for each message m on each link in a collaboration diagram there exists t in T such that t causes m to be executed.

They also define the all message sequence path criterion and refer to it as the all message paths (AMP) criterion. Lastly, they define the collection coverage (Coll) criterion. The description of this criterion in [3] is very unclear and difficult to interpret. Therefore, the following explanation and definition of the the collection criterion is taken from [3].

Collaboration diagrams may specify communications with collections of objects. The collections can be partitioned into different domains, similar to the partitioning of multiplicities of objects and relations in class diagrams. The collection coverage (Coll) criterion requires that a collection in each domain be instantiated at least once.

Definition (Collection Coverage (Coll) Criterion). A test set T satisfies the collection coverage criterion if and only if T tests each interaction with collection objects of various representative sizes at least once.

Andrews *et al.* do not present an empirical evaluation of their proposed criteria in [3]. But some evaluations of the criteria have been performed elsewhere. Ghosh *et al.* [16] describe an approach to design testing that incorporates the criteria defined by Andrews *et al.* [3]. They perform a single case study to illustrate the use of the criteria and compare the number of test cases required to satisfy each of the criteria. They find that test cases have the ability to satisfy more than one criteria. They do not provide empirical evidence of the effectiveness of their approach. Using a case study, Kawane [22] perform an evaluation of the fault detection effectiveness of the criteria proposed by Andrews *et al.*. Two models are used in the case study. The first model is made up of eight use cases and their corresponding collaboration diagrams, and a class diagram containing eight classes. The second model is made up of of nine use cases and their corresponding collaboration diagrams, and a class diagram containing six classes. These models are seeded with faults and test cases are generated to satisfy the criteria. These test cases succeed in finding 8 out of 10 faults in the first model and 9 out of 12 faults in the second model. These results cannot be generalised as the two models used in the study are smaller and less complex than typical real world models.

3.4 State Machine Diagram Criteria

Statechart diagrams are now called state machine diagrams in UML 2.0 [18]. They are based on finite state machines and model the life-cycle of an object; they specify the states an object can go through during its lifetime along with the transitions between the states [7].

One requirement of adequate testing based on statechart diagrams is that all transitions in the diagram are covered by test executions. This is called the

Table 4: Coverage Criteria based on State Machine Diagrams

<i>Author</i>	<i>Criterion</i>
Chevally and Thevenod-Fosse	All Transitions Coverage
Kim <i>et al.</i>	Control and Data Flow Coverage
Offutt and Abdurazik	All Transitions Coverage
Offutt and Abdurazik	Full Predicate Coverage
Offutt and Abdurazik	Transition-Pair Coverage
Offutt and Abdurazik	Complete Sequence Coverage
Wu <i>et al.</i>	All Transitions Coverage
Wu <i>et al.</i>	All Context-Dependence Relationship Coverage

all-transitions coverage criterion and is different to the all-transitions criterion defined for collaboration diagrams in section 3.3. It can be defined as follows:

Definition(All-Transitions Coverage Criterion). A test set T satisfies the all-transitions criterion if and only if for each transition tr in a statechart there exists t in T such that t causes tr to be traversed.

The all-transitions coverage criterion is proposed by several authors. Chevally and Thevenod-Fosse [10] propose a testing technique for the generation of test cases from statechart diagrams. The technique uses an adapted form of statistical functional testing and the all-transitions coverage criterion as the testing criterion. Wu *et al.* [31] make use of the criterion in their approach to integration testing of component-based software.

Offutt and Abdurazik [25] describe a technique for system testing and define several statechart based coverage criteria. They define the all-transitions coverage criterion along with the full predicate coverage criterion, the transition pair coverage criterion and the complete sequence criterion. Their criteria are defined for change event enabled transitions. A transition between two states in a statechart diagram may be annotated with a predicate (condition). This specifies that movement from one transition to the other can only occur if the predicate is true. The full predicate coverage criterion requires that each clause in each predicate on each transition be tested independently.

Definition (Full Predicate Coverage). A test set T satisfies the full predicate coverage criterion if and only if for each clause c in each predicate in a statechart diagram there exists $t1$ in T such that $t1$ causes c to evaluate to TRUE and there exists $t2$ in T such that $t2$ causes c to evaluate to FALSE while all other clauses in the condition have values such that the value of the condition will always be the same as the clause under test.

The transition pair coverage criterion requires pairs of (adjacent) transitions to be traversed.

Definition (Transition Pair Coverage). A test set T satisfies the transition pair coverage criterion if and only if for each pair of adjacent transitions $S_i:S_j$ and $S_j:S_k$ in a statechart, there exists t in T such that t causes the pair of transitions to be traversed in sequence.

A complete sequence is a sequence of state transitions that form a complete practical use of the system [25]. It is not feasible to cause every complete sequence to be taken as the number of complete sequences is usually very large and in some cases it is possible to have an infinite number of complete sequences. Therefore, the complete sequence criterion requires the expertise of the test engineer to choose a subset of all the possible complete sequences. It is defined as follows:

Definition (Complete Sequence). A test set T satisfies the complete sequence criterion if and only if for each complete sequence s defined by the test engineer there exists t in T such that t causes s to be taken.

Offutt and Abdurazik [25] describe a tool that has been developed to generate test cases to satisfy the criteria and carry out an empirical investigation of the criteria on a moderate system. The system is seeded with 25 faults and test cases are generated using the developed tool to satisfy the transition and full predicate criteria. Other test cases are generated by hand to satisfy the statement coverage criterion. They find that the test cases satisfying the full predicate criterion reveal all the seeded faults. They discover that the test cases satisfying the transition coverage criterion perform marginally better than those satisfying the statement coverage criterion.

Binder proposes several testing strategies that make use of the Flattened Regular Expression (FREE) state model. The strategies involve applying conventional state based testing criteria such as those found in [11] to the expanded diagram.

Kim *et al.* propose a method for transforming state diagrams into a representation to which conventional control and data flow criteria can be applied. The state diagram is first transformed to an extended finite state machine (EFSM). An EFSM is a tuple that consists of a set of global states, an initial global state and a set of global transitions. Test cases are generated using the following proposed criteria: (a) path coverage, which requires all paths through the EFSM to be covered (b) state coverage, which requires all states in the EFSM to be covered and (c) transition coverage, which requires all transitions in the EFSM to be covered. The extended finite state machine is then transformed into a flow graph. Test cases are generated by applying data flow criteria such as those found in [15] to the flow graph.

Wu *et al.* [31] propose the all context-dependence relationships coverage criterion. It requires coverage of all the context dependent relationships derived from a statechart diagram. For the explanation of a context dependent relationship the reader is referred to [31].

Definition(All Context Dependence Relationships Coverage Criterion). A test set t satisfies the all context dependence relationships criterion if and only if for each context dependence relationship r derived from a statechart diagram, there exists t in T such that t tests r .

Abdurazik *et al.* [2] carry out a single experiment to compare the fault revealing capabilities of test sets that satisfy sequence diagram criteria with test sets that satisfy statechart diagram criteria. The all message paths criterion [1] for sequence diagrams and the full predicate criterion [25] for statechart diagrams are the criteria used in this experiment. They find that the sequence diagram test sets are better at revealing integration level faults than the statechart test sets, and that the statechart test sets are better at revealing unit level faults than the sequence diagram test sets. They also find that more test cases are required to satisfy the statechart criterion than are required for the sequence diagram criterion.

A study is conducted by Briand *et al.* [9] to compare the cost effectiveness and fault detection effectiveness of various statechart based testing techniques. The transition tree (TT) criterion proposed by Binder [6] and the all transitions (AT), the all transitions pairs (ATP), and the full predicate (FP) criteria proposed by Offutt [25] are used in this study. The results of the study demonstrate that when the AT criterion is used alone it does not provide an adequate level of fault detection. It appears that the ATP criterion is very successful at detecting nearly all the faults introduced but the drawback of using this criterion is that it comes at a large increase in cost. It is found that the TT criterion is not always cost effective; the FP is more expensive than the ATP but is just as effective at revealing faults. Here, Briand *et al.* present a systematic procedure for comparing these criteria, thus making it possible for similar studies to be replicated in the future.

3.5 ACTIVITY DIAGRAM CRITERIA

Essentially, an activity diagram is a flow chart that shows the activities that take place among objects. In an activity diagram the emphasis is on the flow of control from activity to activity. Activity diagrams are special cases of statechart diagrams [7]. Therefore, criteria proposed for statechart diagrams can be applied to activity diagrams.

Dinh-Trong [12] consider coverage criteria for activity diagrams. These criteria are presented as a general discussion and not explicitly defined. They suggest the all edge criterion which requires every activity edge of an activity diagram to be covered during testing. These criteria have been adapted from existing statechart diagram criteria.

Linzhang *et al.* [23] propose a gray-box coverage criterion based on activity diagrams. They define a basic path as a complete path through an activity diagram where each loop is executed either zero or one times. The set of all basic paths is the set of basic paths that cover all action states and transitions. They define the all basic paths coverage criterion.

Definition (All Basic Paths Coverage Criterion). Let BP be the basic path set of an activity diagram, a test set T satisfies the all basic paths coverage criterion if and only if for each p in BP there exists t in T such that t causes p to be traversed.

3.6 USE CASE DIAGRAM CRITERIA

Use case diagrams are used to visualise the behaviour of a system. They consist of a set of use cases and actors and their relationships [7]. It is noted in [28] that Winter [30] has proposed several coverage criteria based on use case diagrams which are adaptations of the Myers' criteria [24]. These are use case step coverage, use case branch coverage, use case scenario coverage, use case boundary body coverage, and use case path coverage. These are not discussed further here as no other information could be found detailing these criteria.

Table 5: Coverage Criteria based on Use Case Diagrams

<i>Author</i>	<i>Criterion</i>
Winter	Use Case Step Coverage
Winter	Use Case Branch Coverage
Winter	Use Case Scenario Coverage
Winter	Use Case Boundary Body Coverage
Winter	Use Case Path Coverage

3.7 OTHER UML DIAGRAMS

There have been no criteria defined in the literature for the UML deployment and component diagrams. Several new diagrams have been introduced in UML 2.0. These are the composite structure diagram, the interaction overview diagram and the timing diagram. At present no criteria exist for these diagrams.

4 SUMMARY AND CONCLUSION

Recently, research has been carried out to investigate how UML can be used in the testing phase of the software development process. As a result, a number of UML-based coverage criteria have been proposed in the literature. These criteria are based on coverage of elements of UML diagrams. In this report, these UML-based criteria are surveyed. For each of the criteria a formal definition is presented which is necessary to facilitate a comparison of the criteria. The research on the comparison and evaluation of these criteria is also reviewed. It has been found that relatively little work has focussed on empirically investigating how effective the criteria are at detecting faults. Furthermore, no research has been carried out to show how the various criteria relate to each other in terms of the coverage they provide. Therefore, it is believed that these are important topics for future investigation.

REFERENCES

- [1] A. Abdurazik and J. Offutt. Using UML collaboration diagrams for static checking and test generation. In *Proceedings of 3rd International Conference on The Unified Modeling Language*, York, UK, 2-6 October 2000.
- [2] A. Abdurazik, J. Offutt, and A. Baldini. A controlled experimental evaluation of test cases generated from UML diagrams. Technical report, Information and Software Engineering Department, George Mason University, May 2004.
- [3] A. Andrews, R. France, S. Ghosh, and G. Craig. Test adequacy criteria for UML design models. *Software Testing, Verification and Reliability*, 13:95–127, 2003.
- [4] F. Basanieri and A. Bertolino. A practical approach to UML-based derivation of integration tests. In *Proceedings of 4th International Quality Week Europe*, Brussels, Belgium, 20-24 November 2000.
- [5] B. Beizer. *Software Testing Techniques*. John Wiley & Sons, Inc., New York, NY, USA, 1990.
- [6] R. V. Binder. *Testing Object-Oriented Systems: Models, Patterns and Tools*. Addison-Wesley, 2000.
- [7] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.
- [8] L. Briand and Y. Labiche. A UML-based approach to system testing. *Journal of Software and Systems Modeling*, 1:10–42, 2002.
- [9] L.C. Briand, Y. Labiche, and Y. Wang. Using simulation to empirically investigate test coverage criteria based on statechart. In *Proceedings of the 26th International Conference on Software Engineering*, pages 86–95, Edinburgh, Scotland, 23-28 May 2004.
- [10] P. Chevalley and P. Thevenod-Fosse. Automated generation of statistical test cases from UML state diagrams. In *Proceedings of 25th Annual International Computer Software and Applications Conference*, pages 205–214, Chicago, Illinois, USA, 8-12 October 2001.
- [11] T. Chow. Tesing software design modeled by finite state machines. *IEEE Transactions on Software Engineering*, 4(3):178–186, 1978.
- [12] T. Dinh-Trong. A systematic approach to testing UML design models. In *Doctorial Symposium, 7th International Conference on The Unified Modeling Language*, Lisbon, Portugal, 10-15 October 2004.
- [13] R.H. Doong and P.G. Frankl. The ASTOOT approach to testing object-oriented programs. *ACM Transactions on Software Engineering and Methodology*, 3(2):101–130, 1994.

- [14] F. Fraikin and T. Leonhardt. SeDiTeC-testing based on sequence diagrams. In *Proceedings 17th IEEE International Conference on Automated Software Engineering*, pages 261–266, Edinburgh, Scotland, 23-27 September 2002.
- [15] P.G. Frankl and E.J. Weyuker. An applicable family of data flow testing criteria. *IEEE Transactions on Software Engineering*, 14(10):1483–1498, 1988.
- [16] S. Ghosh, R. France, C. Braganza, N. Kawane, A. Andrews, and O. Piskalns. Test adequacy assessment for UML design model testing. In *Proceedings of the 14th International Symposium on Software Reliability Engineering*, 17-20 November 2003.
- [17] J.B. Goodenough and S.L. Gerhart. Toward a theory of test data selection. In *Proceedings of the International Conference on Reliable Software*, pages 493–510, Los Angeles, California, 1975.
- [18] The Object Management Group. UML 2.0 draft specification, 2003.
- [19] The Object Management Group. The Unified Modelling Language Version 1.5 OMG. formal/2003-03-01, March 2003.
- [20] M.J. Harrold. Testing: a roadmap. In *ICSE - The Future of Software Engineering Track*, pages 61–72, Limerick, Ireland, 4-11 June 2000.
- [21] J. Hartmann, C. Imoberdorf, and M. Meisinger. UML-based integration testing. *SIGSOFT Software Engineering Notes*, 25(5):60–70, September 2000.
- [22] N. Kawane. Fault detection effectiveness of UML design model test adequacy criteria. In *Student Paper for International Symposium on Software Reliability Engineering*, Denver, USA, 17-20 November 2003.
- [23] W. Linzhang, Y. Jiesong, Y. Xiaofeng, H. Jun, L. Xuandong, and Z. Guoliang. Generating test cases from UML activity diagrams based on gray-box method. In *11th Asia-Pacific Software Engineering Conference*, pages 284–291, Busan, Korea, 30 November - 3 December 2004.
- [24] G. J. Myers. *Art of Software Testing*. John Wiley & Sons, Inc., New York, NY, USA, 2004.
- [25] J. Offutt and A. Abdurazik. Generating tests from UML specifications. In *Proceedings of 2nd International Conference on The Unified Modeling Language*, pages 416–429, Fort Colins, Colarado, USA, October 1999.
- [26] T.J. Ostrand and M.J. Balcer. The category-partition method for specifying and generating functional tests. *Communications of the ACM*, 31(6):676–686, 1988.

- [27] O. Pilskalns, A. Andrews, S. Ghosh, and R. France. Rigorous testing by merging structural and behavioral UML representations. In *Proceedings of 6th International Conference on The Unified Modeling Language*, San Francisco, CA, USA, 20-24 October 2003.
- [28] A. Reuys, S. Reis, E. Kamsties, and Klaus Pohl. Derivation of domain test scenarios from activity diagrams. In *Workshop on Product Line Engineering - The Early Steps: Planning, Modeling, and Managing*, Erfurt, Germany, September 2003.
- [29] A. Rountev, S. Kagan, and J. Sawin. Coverage criteria for testing of object interactions in sequence diagrams. In *Fundamental Approaches to Software Engineering*, Edinburgh, Scotland, 2-10 April 2005.
- [30] M. Winters. *Quality Assurance for Object-Oriented Software - Requirements Engineering and Testing w.r.t. Requirements Specification (in German)*. PhD thesis, University of Hagen, Germany, Sept 1999.
- [31] Y. Wu, M. Chen, and J. Offut. UML-based integration testing for component-based software. In *Proceedings of 2nd International Conference on COTS-Based Software Systems*, Ottawa, Canada, 10-12 February 2003.