Carleton Technical Report SCE-05-02 Version 3, October 2006

A State-based Approach to Integration Testing based on UML Models

Shaukat Ali[1], Lionel C. Briand[2], Muhammad Jaffar-ur Rehman[1], Hajra Asghar[1], Muhammad Zohaib Z. Iqbal[1], Aamer Nadeem[1]

[1]*Center for Software Dependability, Mohammad Ali Jinnah University*
Islamabad, Pakistan
{shaukat, hajra, zouhaib, anadeem}@jinnah.edu.pk

[2]*Software Quality Engineering Laboratory, Department of Systems and Computer Engineering, Carleton University, Canada*
briand@sce.carleton.ca

Abstract: Correct functioning of object-oriented software depends upon the successful integration of classes. While individual classes may function correctly, several new faults can arise when these classes are integrated together. In this paper, we present a technique to enhance testing of interactions among modal classes. The technique combines UML collaboration diagrams and statecharts to automatically generate an intermediate test model, called SCOTEM (State COllaboration TEst Model). The SCOTEM is then used to generate valid test paths. We also define various coverage criteria to generate test paths from the SCOTEM model. In order to assess our technique, we have developed a tool and applied it to a case study to investigate its fault detection capability. The results show that the proposed technique effectively detects all the seeded integration faults when complying with the most demanding adequacy criterion and still achieves reasonably good results for less expensive adequacy criteria.

Key Words: UML based testing, automated testing, object-oriented systems

1 Introduction

The Object-oriented (OO) paradigm offers several benefits, such as encapsulation, abstraction, and reusability to improve the quality of software. However, at the same time, OO features also introduce new challenges for testers: interactions between objects may give rise to subtle errors that could be hard to detect.

The Unified Modeling Language (UML) has emerged as the de-facto standard for analysis and design of OO systems. UML provides a variety of diagramming notations for capturing design information from different perspectives. In recent years, researchers have realized the potential of UML models as a source of information in software testing [17, 53, 7, 23, 26, 29, 30, 11, 35, 39, 56, 47, 14, 48]. Many UML design artifacts have been used in different ways to perform different kinds of testing. For instance, UML statecharts have been used to perform unit testing, and interaction diagrams (collaboration and sequence diagrams) have been used to test class interactions.

Modularity aims at encapsulating related functionalities in classes. However, complete system-level functionality (use case) is usually implemented through the interaction of objects. Typically, the complexity of an OO system lies in its object interactions, not within class methods which tend to be small and simple. As a result, complex behaviours are observed when related classes are integrated and several kinds of faults can arise during integration: interface faults, conflicting functions, missing functions [1]. Thus testing each class independently does not eliminate the need for integration testing. A large number of possible interactions between collaborating classes may need to be tested to ensure the correct functionality of the system.

In this paper, we present a technique that enhances the integration testing[1] of classes by accounting for all possible states of collaborating classes in an interaction. This is important as interactions may trigger correct behavior for certain states and not for others. In order to achieve such an objective the proposed technique builds an intermediate test model called SCOTEM (State COllaboration TEst Model) from a UML collaboration diagram and statechart diagrams of the objects involved in the object interactions. The SCOTEM models all possible paths for object state transitions that a message sequence may trigger. The test generator uses SCOTEM to generate test paths whose corresponding test cases aim at detecting the faults that may arise due to invalid object states during object interactions. This paper also reports a case study in order to assess the cost-effectiveness of various test adequacy criteria based on the SCOTEM test model.

The rest of this paper is organized as follows: Section 2 presents a brief survey of the related works in the areas of state-based testing and UML-based test path generation; Section 3 presents the proposed approach to integration testing, including a discussion of coverage criteria for test path generation; Section 4 describes the prototype tool that we have developed to automate the proposed technique; Section 5 presents a case study to investigate the cost-effectiveness of the proposed technique; and finally Section 6 concludes the paper.

2 Related Work

Traditional testing strategies for procedural programs, such as data flow analysis and control flow analysis cannot be directly applied to OO programs [35]. Extensions of these techniques for OO programs have been proposed by Buy et al. [26] and Martena et al. [41]. A structural test case generation strategy by Buy et al. [26] generates test cases through symbolic execution and automates deduction for the data flow analysis of a class. Kung et al. [36] proposed an idea to extract state models from the source code, whereas others suggest test generations from pre-existing state-models [58, 29, 48, 11, 47, 56, 40]. In the sections below, we will discuss more specific UML-based testing techniques.

2.1 UML based Test Generation

In the section below, we will discuss techniques that support different levels of testing using UML diagrams.

2.1.1 Unit Level Testing

Unit level testing refers to the tests performed to check the correct functionality of individual execution units. In case of procedural languages, functions or procedures are the smallest possible execution units, while for OO programs a class is considered the smallest unit. Unit testing forms the basis for the subsequent higher level testing activities as it establishes the minimal operability of units that will be integrated to produce the actual functionality.

Tse and Xu [57] have proposed an approach to derive test cases from Object Test Models (OTM). State space partitions of the attributes of a class are used with the OTM to generate a test tree. The actual test cases are derived from the test tree. Nori and Sreenivas [55] have proposed a specification-based testing technique that splits the specifications of a class into structural and behavioral components. Structural aspects define the attributes and method specifications of a class, whereas the behavioral component is a state machine describing the valid method invocation sequences. Another specification-based testing technique is presented by Vieira et al. [51] that utilizes UML statecharts to generate test cases automatically. A tool, DAS-BOOT, has also been built that partially automates the procedure. An idea of converting test generation into an AI planning problem has also been proposed [29, 48]. UML statecharts are used for testing information and producing AI planning specifications, which are then processed by planning tools to produce test cases. Kim et al. [11] propose a test case generation strategy from UML statecharts. These statecharts are converted to Extended Finite State Machines (EFSMs) and traditional control and data flow analysis is then performed on the EFSMs to generate test cases.

[1] In the paper, the term integration testing denotes the testing of interactions between classes

A unique approach for state-based class testing is proposed by Sokenou and Herrmann [56]. This approach uses UML statecharts and the Object Teams approach [59] to test classes. Another state-based approach is presented by Li et al. [38] that can be used to test specific properties of reactive systems. Kim et al. [34] used statecharts to generate test sequences for Java-Based concurrent systems. Kansomkeat and Rivepiboon [33] have converted UML statecharts into an intermediate model known as Testing Flow Graph (TFG). This graph reduces complexities of statecharts and produces a simple flow graph. Test cases are finally generated by traversing the TFG using state and transition coverage criteria. The proposed methodology was evaluated using mutation testing. A systematic way of defining test cases from statistical usage models is defined by Riebisch et al. [47]. A statechart is derived from the use cases and is used to generate a usage model. Another approach for statistical test case generation from UML statecharts is proposed by Chevalley and Thevenod-Fosse [27]. This approach applies functional distribution to generate input values and transition coverage to generate test paths. Offut et al. [42] present a formal technique to derive tests at different coverage levels from UML statecharts. These levels include All-Transition*,* Transition Pair Coverage, Full Predicate Coverage*,* and Complete Sequence Coverage. A tool has also been developed to support the strategy. An extension of this work to support system level test generation from state-based specification has been made [14]. Results of an experiment carried out to validate the application of Round Trip Test Strategy [1] on UML statecharts are presented in Briand et al. [7]. Authors also propose improvements on the strategy based on the analysis of these results. In [25], a method to derive test data based on UML statecharts is defined to generate test cases given in the form of transition sequences on an object.

2.1.2 Integration Level Testing

The integration testing of OO software is concerned with testing the interactions between units. This essentially attempts to validate that classes, implemented and tested individually, provide the intended functionality when made to interact with each other. The focus at this level is on testing interactions between classes through method calls or asynchronous signals, as well as interactions with databases or hardware. Adequate unit-level testing of individual classes supports but does not produce a reliable integrated cluster (group of interacting classes). This is because some of the bugs remain dormant during unit testing and pop-up only after integration for a specific state of the cluster. Hence the need of integration testing is inevitable. In the section below, we will discuss various approaches relating to integration testing of object-oriented software.

Abdurazik et al. have used collaboration diagrams for static checking of source code and measuring adequacy of a test suite [17]. The approach applies traditional control and data flow analysis on collaboration diagrams to generate integration tests. TEst Sequence generaTOR (TESTOR) [44] is an approach used to generate integration tests using UML statecharts and collaboration diagrams for component based systems. The behavior of each component is specified in the form of UML statecharts, while the test directives are specified in the form of UML collaboration diagrams. Basanieri and Bertolino [20] used a User Interaction Testing (UIT) model to perform integration testing. A UIT is generated from use cases and UML sequence diagrams. An enhancement to this approach is proposed in [21]. Pilskalns et al. [46] have proposed a methodology to generate test cases from use case diagram and sequence diagrams. These diagrams are converted into an intermediate form Object Method Directed Acyclic Graph (OMDAG), which is used to generate integration tests. The SeDiTec [28] approach is used for testing interactions between the classes involved in a sequence diagram. Badri et al. [50] have used use cases and collaboration diagrams to generate integration test cases. Another approach for UML based integration testing is proposed by Le Traon et al. [12] and Jeron et al. [12]. This approach uses UML class diagram and generates an intermediate model known as TD Graph (TDG). This graph has three types of dependencies: Class to Class, Class to Method, and Method to Method dependency. This information was further used to determine the ordering of classes. Le Hanh et al. [37] have compared different integration test ordering strategies based on TDG. Another approach for determining class test orders for integration testing is discussed by Briand et al. [6]. This approach uses UML class diagrams and finds a class test order which minimizes the number of stubs required for integration testing.

2.1.3 System Level Testing

A few approaches in the literature also discuss testing at the system level. Briand and Labiche [23] have proposed an approach to generate system level test cases. This approach uses use case diagrams, activity diagrams, and sequence diagrams to generate system level test cases. Various techniques are reported that target different types of systems such as embedded and web based systems [49, 8, 22, 54, 5].

2.1.4 Conformance Level Testing

Testing software in order to establish the fulfillment of the specified requirements is known as conformance testing. A conformance relation defines the correctness criterion of the implementation with respect to the formal specification. General interpretation of conformance testing makes it resemble system level testing where high-level requirements are verified against the observed behavior. In the section below, we will discuss various approaches relating to conformance testing of object-oriented software.

Hartmann [30] has proposed an approach for the conformance testing of distributed component based systems. UML statecharts are used for this purpose. UML statecharts are converted into extended statecharts and properties of Communicating Sequential Processes (CSP) are added. This extended model represents system wide behavior of the system. This model is given as an input to the Test Development Environment (TDE), which generates unit, integration, and system level conformance tests for the system. Another approach for conformance testing of component based systems using UML models has been proposed by Pickin et al. [45]. UML models have also been used to support regression testing [58, 24, 15, 32].

2.2 Other Related State based Testing Techniques

A few state-based testing techniques have been proposed in other contexts than UML and are nonetheless related to our approach.

Burton [53] presents an automated approach to test case generation from statechart specifications. The statecharts are represented in Z notation for the input to the test generation mechanism. Tests are generated using the Category Partition Strategy [43]. The generated test cases are also specified in Z. The paper also presents theorems to validate the existence of some desirable properties in the statecharts that can also help in removing faults before they are propagated to the implementation.

Bogdanov [52] describes an automated testing method based on X-machines (a powerful variation of a Finite State Machine) that attempts to check the compliance of an implementation against its design. Each feature of a statechart is considered independently. The approach also describes ways to reduce the size of generated test suites. The work also includes various case studies that validate the proposed testing technique and a tool (TestGen) to automate the proposed approach.

Gallagher et al. [10] have proposed a methodology to ensure that messages are exchanged between classes in an appropriate order and states of corresponding classes are changed properly. Classes are modeled as finite state machines, which are in turn converted into data flow graphs. Data flow analysis is performed on data flow graphs to automatically generate tests. The approach is applied to component based systems and focuses on testing one component at a time. The approach is also applicable in the context of concurrent applications.

2.3 Comparison with Existing Work

From the above survey, we note that different kinds of UML diagrams have been used for software testing from different perspectives. UML statecharts have been widely used for testing the state-based behavior of software (see 2.1.1). Similarly, UML interaction diagrams have been used for integration testing (see 2.1.2). However, existing approaches do not focus on exercizing the state-based behavior of interacting classes. Techniques in Section 2.1.1 focuses only on testing state-based behavior (internal behavior of a class), whereas techniques in Section 2.2 focus only on testing interactions among classes. More specifically, none of the above works discuss testing by integrating UML interaction and statechart diagrams to uncover state-dependent, class interaction faults

We propose a novel testing approach based on UML collaboration diagrams and statecharts that can be used to uncover state-dependent interaction faults, e.g., inconsistent states of classes involved in a collaboration diagram, wrong calling state of a class in a collaboration diagram, and wrong initial state of a class in a collaboration diagram. A similar methodology was recently proposed by Gallagher et al. [10]. Both approaches perform state dependent interaction testing. Their approach uses finite state machines and converts them to data flow graphs. Traditional data flow analysis criteria are then applied to generate possible test paths. Our proposed approach uses UML statecharts and collaboration diagrams to generate an intermediate model, SCOTEM, and applies different coverage criteria based on the SCOTEM graph representation. We use collaboration diagrams to determine the order of messages between classes and this is different from Gallagher's approach, which uses data flow analysis to determine such orderings. Their approach focuses on testing the possible interactions of a component under test with other components. Relevant states and guards associated with these interactions are collected in a database and are used to construct a Component Flow Graph. Data flow testing criteria are then applied to the graph with respect to define/use of state variables to generate test paths. In contrast, our approach tests class interactions involved in a particular collaboration, modeling object interactions for a specific use case, in all possible object states.

3 The Proposed Approach

The run-time behavior of an object-oriented system is modeled by well-defined sequences of messages passed among collaborating objects. In the context of UML, this is usually modeled as interaction diagrams (collaboration or sequence). In many cases, the states of the objects sending and receiving a message at the time of message passing strongly influence their behavior:

- An object receiving a message can provide different functionalities in different states.
- Certain functionalities may even be unavailable if the receiving object is not in the right state, e.g., a stack cannot be popped if it is in the empty state.
- The functionality provided by an object may also depend on the states of other objects including the sending object of a message.

We propose an integration testing technique that is based on the idea that the interactions between objects should ideally be exercised for all possible states of the objects involved. This is of particular importance in the context of OO software as many classes exhibit a state-dependent behavior. Such testing objective is implemented by generating a graph-based test model (SCOTEM) and by covering all paths in the model. The proposed technique can be applied during the integration test phase, right after the completion of class testing. It consists of the following four steps:

1. *SCOTEM Generation:* An intermediate test model, called SCOTEM (State COllaboration TEst Model) is constructed from a UML collaboration diagram, and the corresponding statecharts.
2. *Test Paths Generation:* Test paths are generated from the SCOTEM based on several possible alternative coverage criteria.
3. *Test Execution:* All selected test paths are executed by using manually-generated test data and an execution log is created, which records object states before and after execution of each message in a test path. The object states are determined using state invariant assertions.
4. *Result Evaluation:* The object states in the execution log are compared with the expected object states in the test paths generated from SCOTEM. This means that these test paths also contain oracle information in the form of expected states of the objects. If any state of any object after execution of a

test path is not in the required resultant state, then the corresponding test case is considered to have failed.

Figure 1 presents a flowchart for the proposed technique, which shows the above four phases, and their relationships with various artifacts. In the following sub-sections, we describe the proposed testing technique in greater detail with the help of a simple example.

3.1 Defining the SCOTEM Test Model

The SCOTEM is an intermediate test model used to automatically generate test specifications for class integration testing. In this sub-section, we describe how this test model is constructed from a given UML collaboration diagram, and its corresponding statecharts. We have, however, made some assumptions about these UML models:

- Collaboration diagrams may contain synchronous call messages or asynchronous signal messages, and statecharts their corresponding call events. Though we have not experimented with them in our case study, asynchronous signals are handled in a similar fashion to synchronous call messages and have no effect on the SCOTEM construction and test path generation algorithm.
- Statecharts are in a flattened form [1]. Note that the flattening process can be automated.
- State invariants of modal classes that are relevant to the selected collaboration diagram must be specified. It is, however, common practice to write state invariants during the definition of statecharts. Though it requires extra effort, the specification of state invariants is important to give a precise meaning to each possible state of a class. Moreover, these state invariants can be implemented as source code assertions that are to be checked during test execution. This is expected to help fault detection (oracle) and debugging.
- The algorithm to generate the SCOTEM test model depends, to some extent, on the version of UML used as a design notation. In the examples of this paper, collaboration diagrams and statecharts are assumed to conform to UML 1.5 (the latest standard when this research started).
- All guards, path, and loop conditions are assumed to be specified using the OCL 1.5.

In order to provide more precise rules for well-formedness and provide some degree of semantics, Figure 2 presents a meta-model for the SCOTEM using the UML class diagram notation. The SCOTEM is a specific graph structure: A vertex corresponds to an instance of a class (in a particular state) participating in the collaboration.

A *Modal Class* can receive a message in more than one state and exhibit distinct behavior for the same message in different states. To capture this characteristic, for modal classes, the SCOTEM contains multiple vertices, where each vertex corresponds to an instance of the class in a distinct abstract state (corresponding to states defined in statecharts). On the other hand, a non-modal class only requires a single vertex in the SCOTEM graph.

The edges in the SCOTEM test model are of two types: message and transition edges. A message edge represents a call action between two objects, and a transition edge represents a state-transition of an object on receiving a message. Each message edge may also contain a condition or iteration. Each message may cause a state transition to occur. A transition edge connects two vertices of the same class. Statecharts may have multiple transitions to distinct states for the same operation. Hence, there may be multiple transition edges (representing a conditional state transition) for the same message edge in SCOTEM. Each of these transitions is generally controlled by mutually exclusive conditions (to prevent non-determinism). The internal representation of a vertex holds the class name and state of the instance it corresponds to. Message edges are modeled in the SCOTEM by attributes of a message including message sequence number, associated operation, receiver object, and the sender object. The transition edges are modeled by the attributes of a transition including sequence number, associated operation, accepting state and sending state.

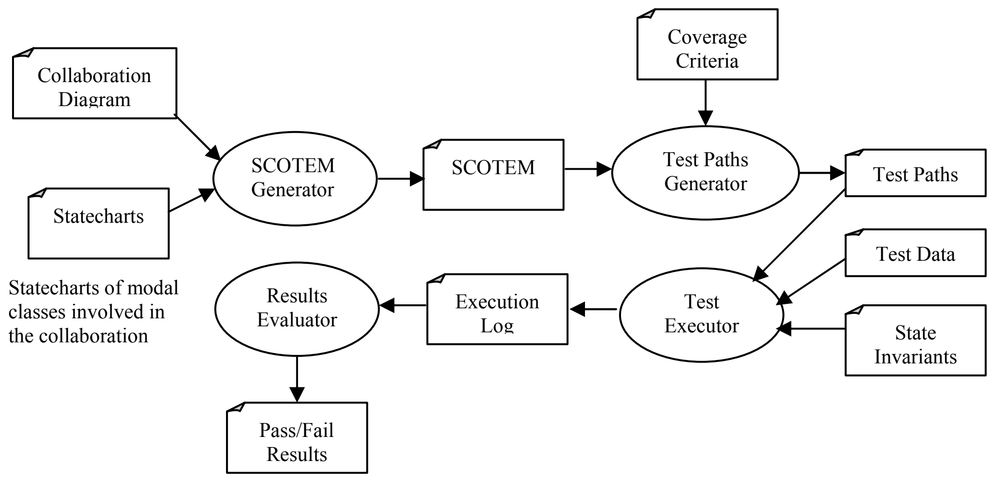

Figure 1. Flowchart for the proposed testing technique

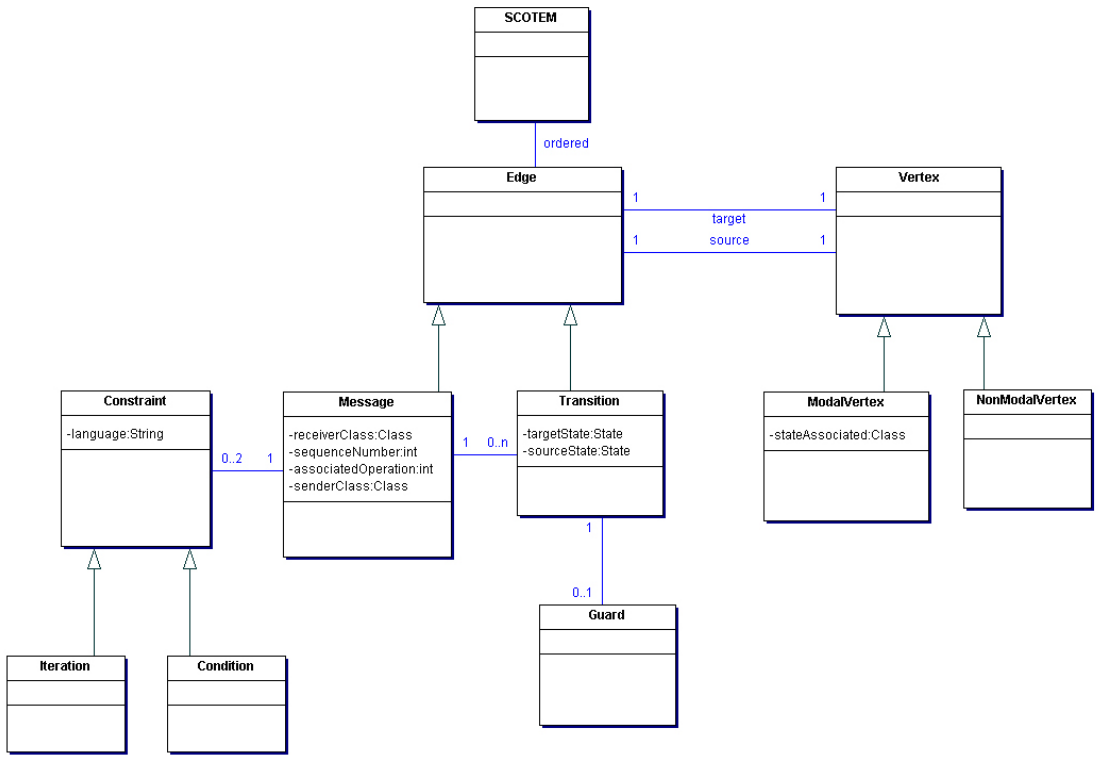

Figure 2. A Metamodel for the SCOTEM Test Model

3.2 Constructing the SCOTEM

In this section, we present an example of SCOTEM construction based on an algorithm presented in Section 4. In subsequent sections, we will discuss the process of test path generation (Section 3.3) and how to handle a potentially large number of test paths (Section 3.4).

The example used consists of an implementation of an Arithmetic Tutor (AT). In its comprehensive form, an arithmetic tutor can be a very complex and large application handling a variety of arithmetic operations, but for the sake of simplicity and clarity, we consider only a limited view of it.

A comprehensive version of an AT program generally runs in two modes: training and assessment modes. In the training mode, the AT provides a step-wise gradual explanation about a particular operation. On the other hand, in the assessment mode, the program evaluates a student's arithmetic skills by presenting randomly generated problems. A detailed performance log is maintained for each session and the history record of a student is updated accordingly at the end. Different levels, regarding complexity and type of problems of training and assessment are available to suit the age and mental aptitude of a student. During the assessment, the level of complexity of the presented problems is adjusted dynamically according to the performance in the current session, while in the training mode students can directly request explanations for problems of any type and complexity.

The implementation of AT that we consider in our example is a restricted form of the assessment mode that deals with the addition operation only. Currently, the application presents randomly generated problems, one after the other, to the students. Students are given unlimited time to solve each problem, but a stop-watch is provided so that they can keep track of the time they take to solve a problem. AT assists the students in solving the problems by laying out the problem in a format easy to understand and solve. AT manages the complexity of the generated problems based on the student's performance in the current session (time taken to solve a problem is not considered in this). If the error count increases beyond a defined value, AT lowers the complexity level for subsequent problems. Similarly, good performance on the part of students increases the complexity level. The class diagram of the AT system is provided in Appendix B.

Figure 3 shows a collaboration diagram for the `newProblem()` system-level operation that instructs AT to generate a new problem according to the current complexity level. An object of the `Coordinator` class receives the operation call message with an argument specifying the type of problem to be generated, which in this case is only addition. Several other classes also interact to generate the problem, which is then passed to the `DisplayManager` that handles its visual presentation. Several other operations, including `evaluateProblem()` and `adjustComplexity()`,are present in the AT. But for the purpose of explaining the technique, we only consider the `newProblem()` operation in the remainder of the section.

In the given collaboration diagram (Figure 3), classes `Coordinator`, `ProblemGenerator`, `Problem`, and `OperandGenerator` are non-modal, while `StopWatch`, `ComplexityRegulator`, `DisplayManager`, `Log`, and `PerformanceRegulator` are modal. Figures 4, 5, 6, 7, and 8 show flattened versions of the statecharts for these modal classes, respectively. The statecharts for the `ComplexityRegulator` class and the `DisplayManager` class had concurrency in the form of sub-states, and thus needed to be flattened before use. The "|" sign in these diagrams denotes a separator for the original concurrent states.

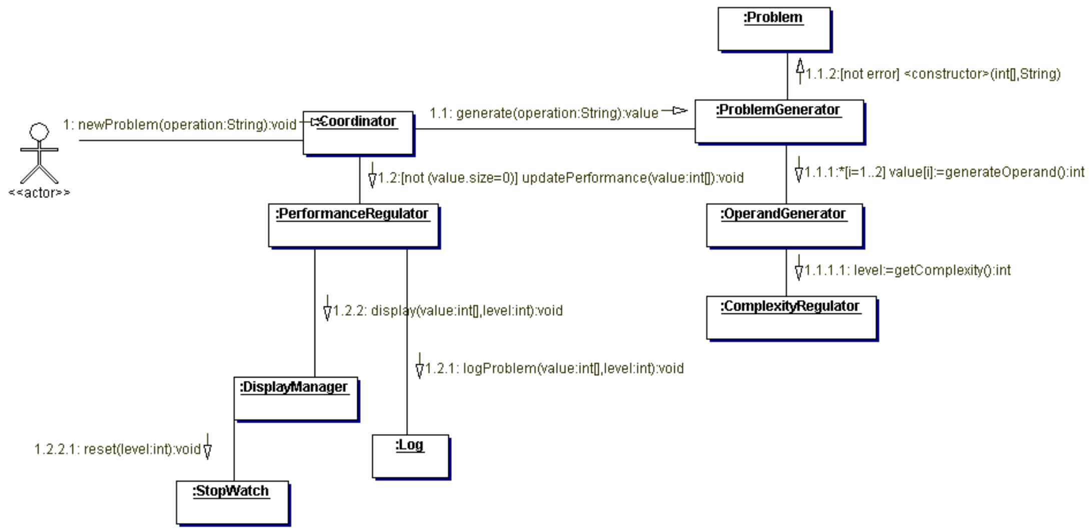
:Problem
1.1.2:[not error] <constructor>(int[],String)
1: newProblem(operation:String):void
:Coordinator
1.1: generate(operation:String):value
:ProblemGenerator
<<actor>>
1.2:[not (value.size=0)] updatePerformance(value:int[]):void
1.1.1:*[i=1..2] value[i]:=generateOperand():int
:PerformanceRegulator
:OperandGenerator
1.1.1.1: level:=getComplexity():int
:ComplexityRegulator
1.2.2: display(value:int[],level:int):void
1.2.1: logProblem(value:int[],level:int):void
:DisplayManager
1.2.2.1: reset(level:int):void
:Log
:StopWatch

Figure 3. Collaboration diagram for `newProblem()`

Collaboration diagrams usually model the execution of a use case, triggered by one or more system-level operations. The recipient of a system-level message is generally a boundary object that forwards the message to one or more use case controller objects [3]. Each message in the collaboration has a well-defined sequence number, source, and destination objects. On the other hand, the statechart of an object defines its states and the messages it can receive in those states. The SCOTEM annotates the chain of messages defined in the collaboration diagrams with the state information of each participant object.

To construct the SCOTEM model for `newProblem()`, we start from the collaboration diagram (Figure 3). For each class box in the collaboration diagram, depending on whether the class is modal, one ore more vertices are created in the SCOTEM. Multiple vertices in the SCOTEM corresponding to a modal class box in the collaboration represent various states in which the class can receive the incoming messages shown in the collaboration for that particular instance. For clarity, SCOTEM vertices corresponding to the same modal class box are grouped together within a dotted box in the SCOTEM diagram. Figure 9 shows a complete SCOTEM model for `newProblem()`. Vertices in the SCOTEM act as placeholders for objects/instances of the classes, and have labels of the form `X@S` (for modal classes), where `X` is the class name and `S` is the state identifier as represented in the statechart. For example, the label `StopWatch@Running` shows that an instance of the `StopWatch` class is in the `Running` state at this vertex. For non-modal classes, vertex labels are of the form `X@X`. For instance, label for the `ProblemGenerator` vertex would be written as `ProblemGenerator@ProblemGenerator`. The `null` vertex in Figure 9 is a dummy vertex that models an external message (e.g., from an actor, that is a GUI). For clarity, message edges are shown with solid lines and are labeled with the message sequence numbers as in the collaboration diagram. We assume that each sequence number corresponds to a full message signature, condition, and iteration, if any exists. Transitions are shown with dotted lines and are labeled with an enabling condition, if any, enclosed within square brackets.

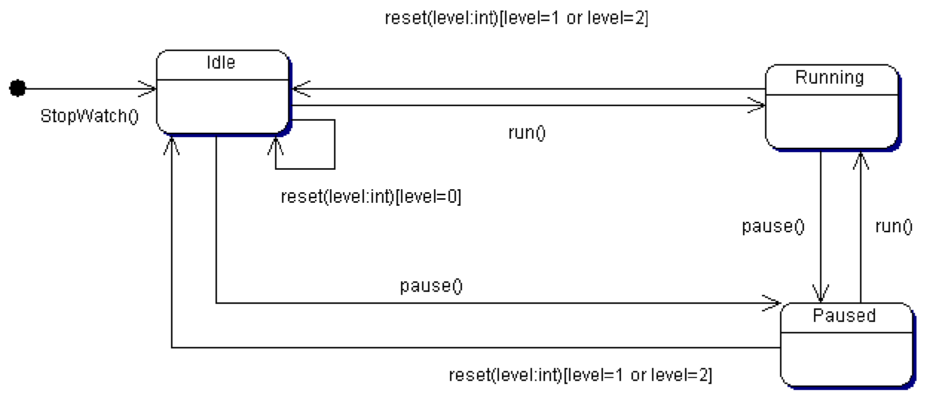
reset(level:int)[level=1 or level=2]
Idle
Running
StopWatch()
run()
reset(level:int)[level=0]
pause()
run()
pause()
Paused
reset(level:int)[level=1 or level=2]

Figure 4. Statechart for `StopWatch`

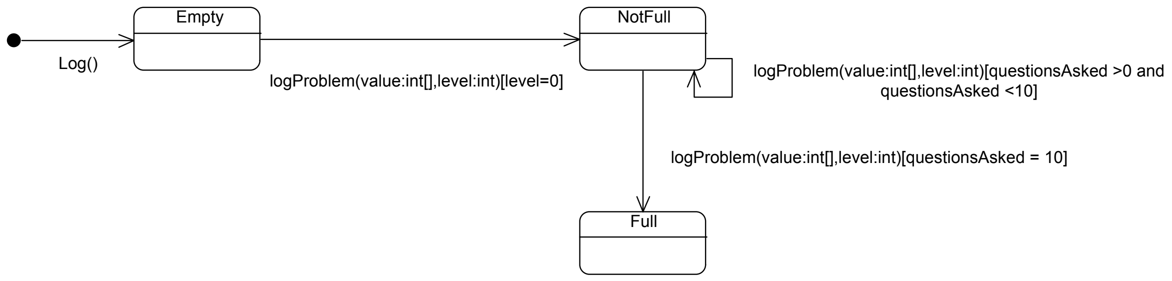
Empty
NotFull
Log()
logProblem(value:int[],level:int)[level=0]
logProblem(value:int[],level:int)[questionsAsked >0 and questionsAsked <10]
logProblem(value:int[],level:int)[questionsAsked = 10]
Full

Figure 5. Statechart for `Log`

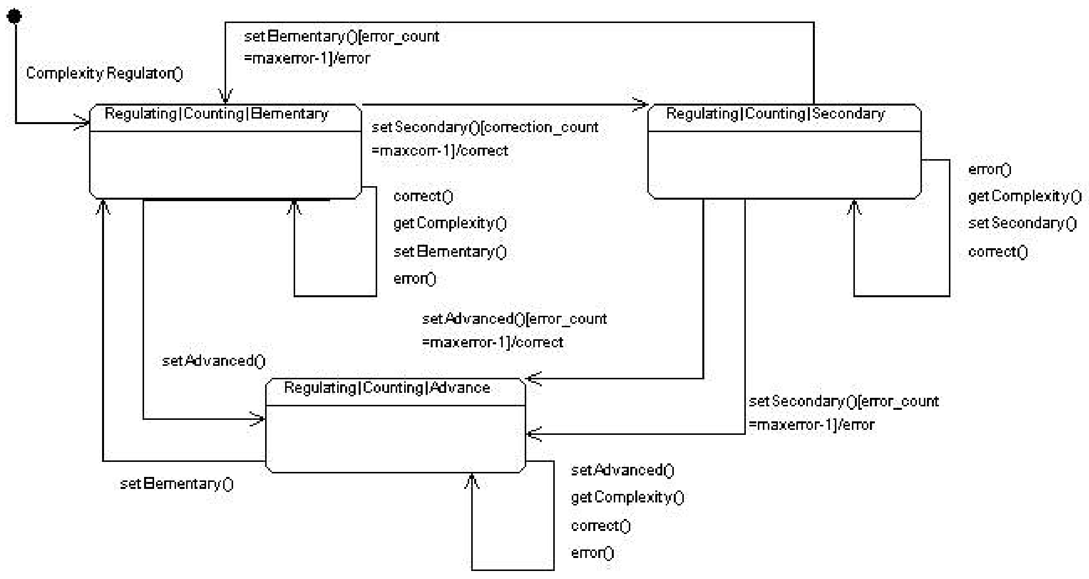
setElementary()[error_count =maxerror-1]/error
ComplexityRegulator()
Regulating|Counting|Elementary
setSecondary()[correction_count =maxcorr-1]/correct
Regulating|Counting|Secondary
error()
getComplexity()
setSecondary()
correct()
correct()
getComplexity()
setElementary()
error()
setAdvanced()[error_count =maxerror-1]/correct
setAdvanced()
Regulating|Counting|Advance
setSecondary()[error_count =maxerror-1]/error
setElementary()
setAdvanced()
getComplexity()
correct()
error()

Figure 6. Flattened statechart for `ComplexityRegulator`

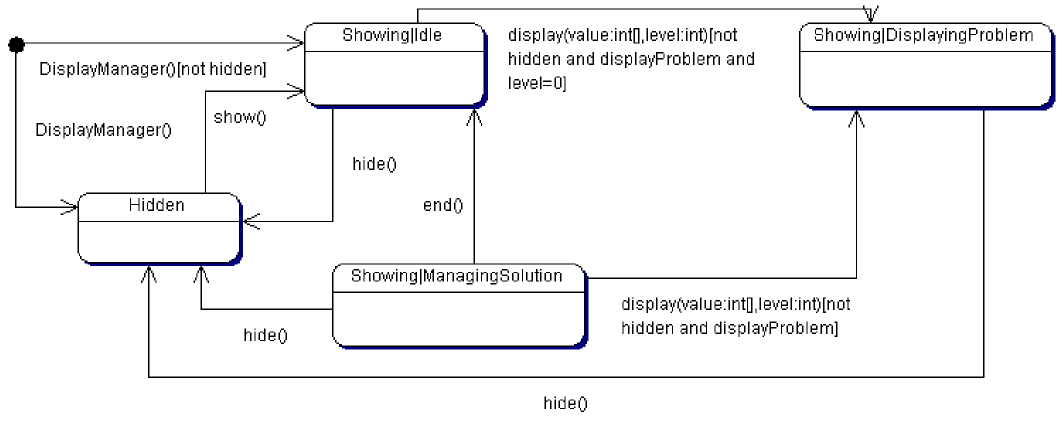
Showing|Idle
Showing|DisplayingProblem
Hidden
Showing|ManagingSolution
DisplayManager()[not hidden]
DisplayManager()
show()
hide()
end()
display(value:int[],level:int)[not hidden and displayProblem and level=0]
display(value:int[],level:int)[not hidden and displayProblem]
hide()
hide()

Figure 7. Flattened statechart for `DisplayManager`

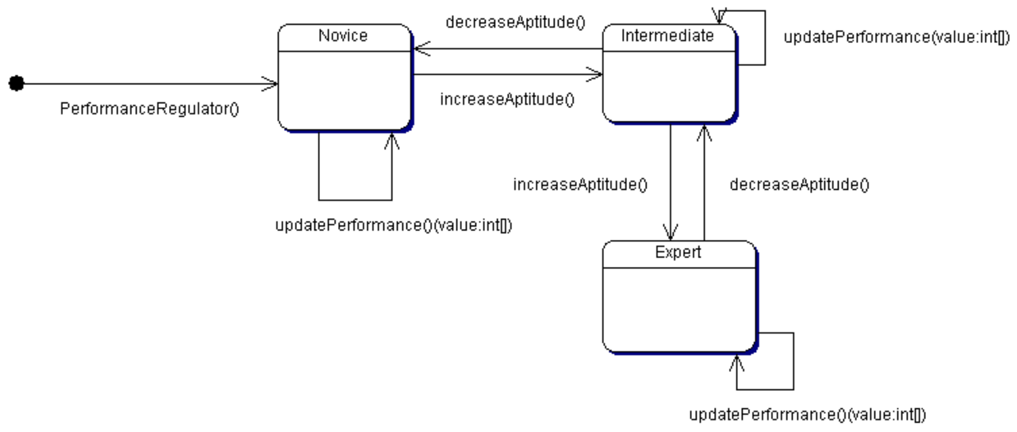

Figure 8. Statechart for `PerformanceRegulator`

3.3 Generating Test Paths from the SCOTEM

In this section, we will discuss the generation of test paths from traversing the SCOTEM graph. Each path tests some interactions between classes in appropriate states. The traversal of all SCOTEM paths tests all interaction between classes in all possible, valid states of classes involved in a particular collaboration.

A test path derived from the SCOTEM represents a path that starts with the initial (null) vertex and contains a complete message sequence of the collaboration. The total number of test paths in a SCOTEM can be determined by taking a product of the numbers of transition paths in each modal class, where each transition path is an internal transition of a modal class from a source state to a target state on receipt of a particular message. Recall from the earlier discussion on the SCOTEM construction (section 3.2) that we select only those transitions from the statechart of a modal class that are valid for a particular message of the collaboration. However, when they are guard conditions, not all paths generated by traversing the SCOTEM are necessarily feasible. Infeasible paths must therefore be detected manually by inspecting all paths containing guard conditions.

From Figure 9, it can be seen that the number of transition paths (selected transitions from a statechart of a modal class corresponding to a called method in a collaboration, e.g., in statechart of `Log` class, there are three transitions corresponding to `logProblem()`) for each of the modal classes is as below:

Class name	*Message received*	*No. of transitions*
`ComplexityRegulator`	`getComplexity()`	3
`PerformanceRegulator`	`updatePerformance()_`	3
`Log`	`logProblem()`	3
`DisplayManager`	`display()`	2
`StopWatch`	`reset()`	3

The total number of test paths can be calculated by taking the cross product of all transition paths of modal classes involved in a collaboration. In this particular case study, there are 3*3*3*2*3=162 test paths. The test generator is responsible for generating these test paths. Figure 10 presents an algorithm for test path generation from a SCOTEM instance. All algorithms in this paper will be formalized as a mix of pseudocode and OCL expressions based on the SCOTEM metamodel. The purpose is to present concise but precise algorithms. Each generated test path is in the form of a string representing the sequence of messages on objects in particular states starting from the system-level operation call message (the null vertex is ignored in this process) that triggers the use case execution modeled by the collaboration diagram. The routine *AllPathCoverage* takes a SCOTEM Test Model as its input and returns a set of all possible test paths for the SCOTEM, which is modeled as an OCL sequence of strings in the algorithm. Whenever a modal class is encountered (Line 13), the current number of test paths is multiplied by the number of transition paths within the encountered modal class (Line 13 to line 29). Line 30 to 34 handles the situation when a non-modal class is encountered.

The diagram in Figure 9 shows only the sequence numbers of messages along the edges, but the internal structure of an edge contains detailed information about the message. In test paths, messages are identified by their names and sequence numbers. Test paths consist of sequences of message expressions describing a complete message sequence for a system-level operation call. The general form of a message expression is as below:

```
Sequence_NO:  *[iteration][Condition]message_name$class_name@
              state_identifier→[Guard]resultant_state
```

where the condition and iteration clauses are omitted for unconditional messages, and guards are omitted for unguarded transitions. The sequence number on the SCOTEM is identical to the corresponding collaboration diagram sequence number.

A message expression indicates a call of message on an object of the specified class (class_name) in the specified state (state_identifier). The part of the expression after the symbol "@" indicates the resultant state of the object on the completion of this message (provided the guard condition is true). This part is not included in the message expression for a non-modal class. The message expression for a non-modal class is therefore of the form:

```
:*[iteration][Condition]message_name$class_name@class_name
```

A test path consists of a sequence of message expressions concatenated to each other, as shown below:

```
:message_expression {: message_espression}*
```

where ":" is used as a separator for message expressions and * denotes repetition.

3.4 Coverage Criteria for Test Paths

In Section 3.3, we presented an algorithm to generate all test paths from the SCOTEM and illustrated it using an example. However, for a more complex system, the number of modal classes involved in a collaboration and the number of states of such classes can be large. This typically results in an exponential growth of the number of test paths that can be generated. It may be impractical, or even impossible, to test all the paths due to the cost involved. To allow for an acceptable level of testing while keeping the cost at the minimum, we define various coverage criteria based on SCOTEM.

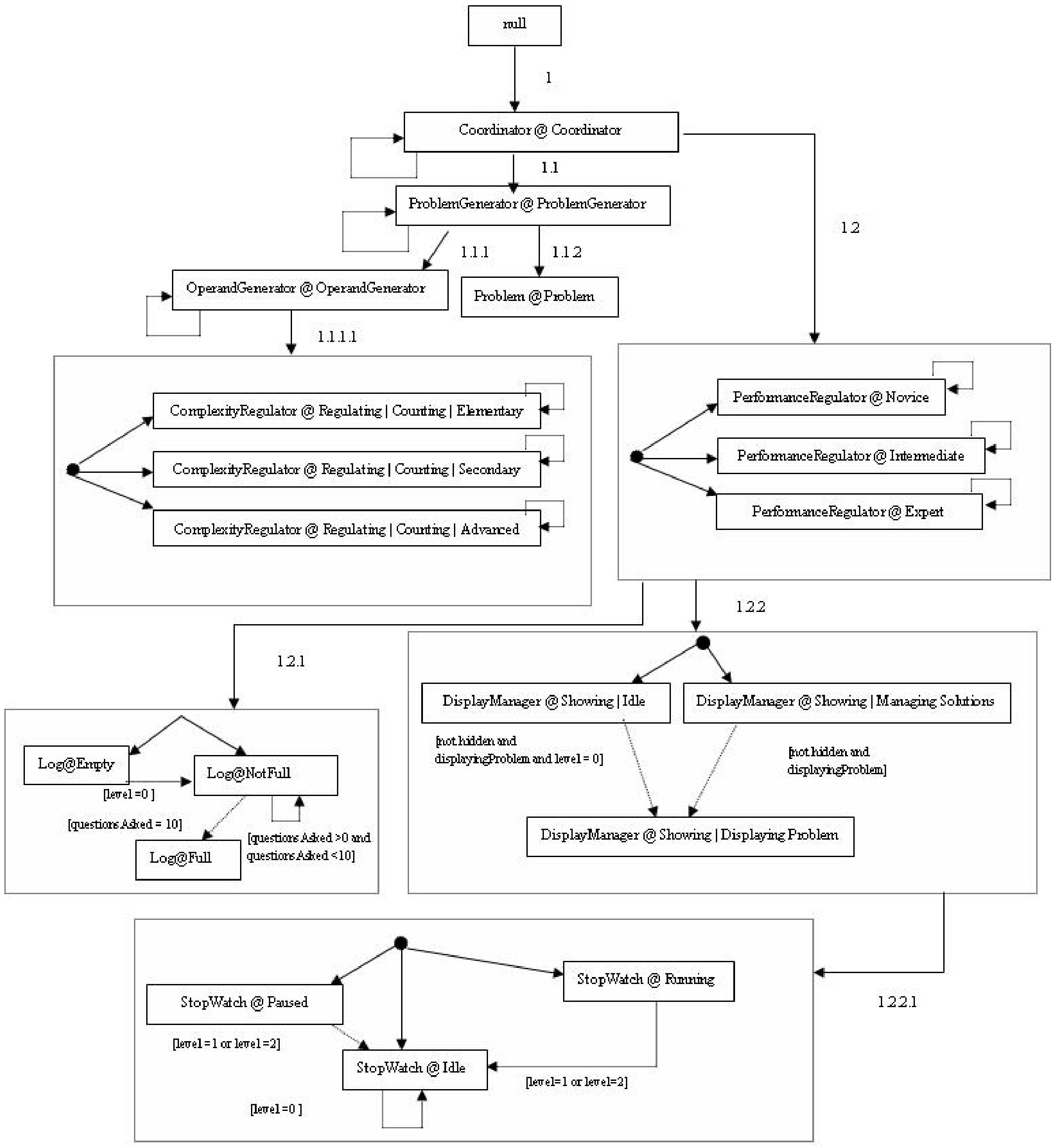
null
1
Coordinator @ Coordinator
1.1
ProblemGenerator @ ProblemGenerator
1.1.1
1.1.2
OperandGenerator @ OperandGenerator
Problem @ Problem
1.1.1.1
1.2
ComplexityRegulator @ Regulating | Counting | Elementary
ComplexityRegulator @ Regulating | Counting | Secondary
ComplexityRegulator @ Regulating | Counting | Advanced
PerformanceRegulator @ Novice
PerformanceRegulator @ Intermediate
PerformanceRegulator @ Expert
1.2.2
1.2.1
Log@Empty
Log@NotFull
[level =0]
[questionsAsked = 10]
[questionsAsked >0 and questionsAsked <10]
Log@Full
DisplayManager @ Showing | Idle
DisplayManager @ Showing | Managing Solutions
[not hidden and displayingProblem and level = 0]
[not hidden and displayingProblem]
DisplayManager @ Showing | Displaying Problem
StopWatch @ Running
StopWatch @ Paused
1.2.2.1
[level = 1 or level = 2]
StopWatch @ Idle
[level=1 or level=2]
[level = 0]

Figure 9. SCOTEM for `newProblem()`

3.4.1 Path Coverage Criteria

This criterion, already presented in Section 3.3, may result in a prohibitive number of test paths, making it impractical. In this subsection, we present simple path coverage criteria that generate a subset of all paths. An appropriate path coverage criterion can be chosen based on the level of testing required, and the test budget available. All paths for the SCOTEM instance in Figure 9 are provided in Appendix D.

```
Algorithm    AllPathCoverage(Ť): $TPS_{Seq}$
Input        Ť: SCOTEM Test Model
Output       $TPS_{Seq}$: A sequence (OCL 1.5) of test paths
Declare      $TPS_{Seq}$: A sequence (OCL 1.5) of test paths
             $MES_{Seq}$: A sequence (OCL 1.5) of message edges in Ť
             medg: A message edge in Ť
             tedg: A transition edge in Ť
             tpm: Test sub path corresponding to a message edge of type String (OCL 1.5)
             tpt: Test sub path corresponding to a transition edge of the current
             message edge of type String (OCL 1.5)
             $TME_{Seq}$ : A sequence (OCL 1.5) of transition edges corresponding to a message
             edge

1. begin
2.    $TPS_{Seq}$ <- {}
3.    $MES_{Seq}$ <- Ť.edge.message
4.    for all medg ∈ $MES_{Seq}$ do
5.          tpm <-medg.sequenceNumber+":"+medg.constraint+medg.associatedOperation
                +"$"+medg. receiverClass+"@"
6.          if |$TPS_{Seq}$| = 0
7.                $TPS_{Seq}$->insertAt(1,tpm)
8.          else
9.                for (i= 1 to |$TPS_{Seq}$|) do
10.                     $TPS_{Seq}$.insertAt(i, $TPS_{Seq}$->at(i)->concat(tpm))
11.               end for
12.         end if
13.         if medg.vertex.oclIsTypeOf(ModalVertex)
14.               $TME_{Seq}$ <- medg.transition
15.               n <- |$TPS_{Seq}$|
16.               for (j=1 to |$TME_{Seq}$|-1) do
17.                     for (i=1 to n) do
18.                           $TPS_{Seq}$->append($TPS_{Seq}$->at(i))
19.                     end for
20.               end for
21.               n <- 0
22.               for (i=1 to |$TME_{Seq}$|) do
23.                     for j=1 to |$TPS_{Seq}$|/ |$TME_{Seq}$| do
24.                           tedg<- $TME_{Seq}$ ->at(i)
25.                           tpt <- tedg.sourceState+"->"+ tedg.guard+
                              tedg.targetState
26.                           $TPS_{Seq}$->insertAt((n+j), $TPS_{Seq}$ ->at(n+j)->concat(tpt))
27.                     end for
28.                     n <- n+j
29.               end for
30.         else
31.               for (i=1 to |$TPS_{Seq}$|) do
32.                     $TPS_{Seq}$->insertAt(i, $TPS_{Seq}$ ->at(i)->concat(medg.receiverClass))
33.               end for
34.         end if
35.   end for
36.  return $TPS_{Seq}$
37. end
```

Figure 10. Algorithm for test paths generation for the All-Path Coverage criterion

Single-Path Coverage
This is the minimal coverage criterion for test path generation that randomly generates a single test path from the SCOTEM. This criterion ensures that each message in an end-to-end sequence of messages in a collaboration is tested once. However, this is the weakest coverage criterion and can be used only to check if the interactions between classes are taking place correctly, regardless of the object states. A test path generation algorithm for *Single-Path Coverage* is given in Appendix A-1 along with an example generated path based on the SCOTEM in Figure 9.

All-Transition Coverage
This criterion ensures that each state transition in a modal class is followed at least once. It subsumes *Single-Path Coverage*. The number of transitions in the modal class with the maximum number of transitions determines the number of paths generated by this criterion. For example, in the SCOTEM model

of Figure 9, the maximum number of transitions in `ComplexityRegulator`, `PerformanceRegulator`, `Log`, and `StopWatch` classes is three (coincidentally, all these classes have same number of transitions), therefore the number of test paths generated by this criterion is three. It can reveal some of the faults occurring due to invalid transitions within a modal class. A test path generation algorithm and example paths based on Figure 9 are given for *All-Transition Coverage* in Appendix A-2.

n-Path Coverage
This coverage criterion selects a specified number (n) of paths from the SCOTEM. The value of n ranges from the number of paths required to achieve *All-Transition Coverage* to the maximum number of possible paths. The *n-Path Coverage* subsumes *All-Transition Coverage*. It first generates test paths such that each state transition in a modal class is followed at least once. The remaining paths are then selected randomly until n paths are obtained. A test path generation algorithm for *n-Path Coverage* is given in Appendix A-3. This algorithm calls the test path generation algorithm for *All-Transition Coverage*. While currently the paths, after fulfilling *All-Transition Coverage*, are generated randomly, our future work will look at identifying optimal path subsets such that these subsets increase the chances of detecting faults.

3.4.2 Additional Coverage Criteria

In addition to the path coverage, we also define coverage criteria for loops and condition predicates. The collaboration diagrams may have path conditions and iterations with messages. Similarly statecharts may have guards associated with the transitions.

Loop Coverage
A collaboration diagram may contain iterative messages, which may also appear in a test path generated from the SCOTEM. The loop coverage criterion requires that the iterative messages in a test path be tested at the loop boundaries. Thus, if an iterative message can be sent a maximum of n times, then using the loop coverage criterion, the iterative message should be sent [2]:

- *n+1 times*
- *n times*
- *n-1 times*
- *0 time*
- *1 time*
- *2 times*

Predicate Coverage
Predicate expressions appear in both the collaboration diagrams and the statecharts as path conditions and guards, respectively. As such, these predicates appear in the SCOTEM model as well. In order to provide coverage for the predicate expressions, a message sequence may need to be tested several times with different test data. For full predicate coverage, test data needs to be generated such that all combinations of truth values of clauses in the predicate are formed. For instance, if a predicate expression contains n clauses, its complete coverage would require each path to be tested 2^n times with different data values. This results in an enormous increase in the total number of paths to be tested.

To reduce this exponential growth in the number of test paths, while still maintaining an acceptable level of predicate coverage, several coverage criteria have been proposed in the literature. Simple approaches such as All True, All False, and All Primes are proposed by Binder [1]. Each-Condition/All Condition was proposed by Myers [4] and can also be used for reducing the number of test paths that involve predicates. Active Clause Criteria, Inactive Clause Criteria, and their variants were proposed by Ammann et al. [18].

In our proposed approach, we use *Active Clause Coverage (ACC)* [18] to test predicates in the test paths. This coverage criterion ensures that each clause in the predicate determines, in at least one test case, the truth value of the predicate such that it is true and false. The use of *ACC* ensures we exercise each clause in a predicate, for both true and false values, without testing all possible combinations of clause truth values and without masking effects for faulty clauses. To illustrate this point, we present an example below.

Consider a class `C` that has two states `A` and `B` as shown in Figure 11. The transition from state `A` to state `B` can occur if the guard predicate *(a>b and b>c)* is true. The guard predicate consists of two clauses, i.e., *(a>b)* and *(b>c)*. To apply *ACC*, we first set *(a>b)* to true so that *(b>c)* drives the truth value of the predicate. Then, appropriate values for *b, c* are chosen to make the major clause *(b>c)* true and false, respectively. We then proceed in a similar way to exercise *(a>b)*. Table 1 and Table 2 show truth values of clauses when exercising *(b>c)* and (*a>b)*, respectively. The combinations of truth values that actually get tested with *ACC* are obtained by the union of rows in the two tables, which results into three rows

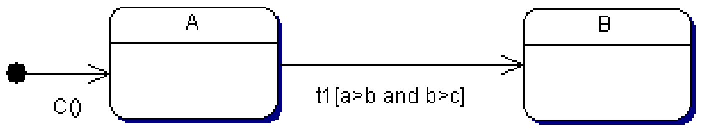

Figure 11. Statechart for class C

Table 1. Truth values when *(b>c)* is a major clause

a>b	*b>c*	*a>b and b>c*
True	True	True
True	False	False

Table 2. Truth values when *(a>b)* is a major clause

a>b	*b>c*	*a>b and b>c*
True	True	True
False	True	False

3.5 Executing the Test Cases

The algorithm in Figure 10 generates a set of test paths. Each test path is parsed to identify the objects and their initial states. The sequence numbers in a test path determine the sequence of sending the messages. The execution of each test path requires test data, which is generated manually, for example by using a black-box technique such as Category-Partition [43].

Once an operation call message has been triggered, the rest of the message chain is executed automatically. The states of all objects involved in a message sequence must be set before the execution of a test case begins. Setting the state of an object may require access to its private data members. We use the Java Reflection facility to access class private data members.

4 Automation

We have developed a prototype tool to validate our approach. The prototype tool automatically constructs a SCOTEM model from a given UML collaboration diagram, and corresponding statecharts (for modal classes). It uses the SCOTEM to generate test paths according to the specified coverage criterion and then, it executes the test paths (using manually generated test data), evaluates the execution results, and creates a log. The tool is composed of four major modules, namely `SCOTEM Constructor`, `Test Path Generator`, `Test Executor`, and `Results Evaluator`. This section describes the function of each of the four modules.

4.1 SCOTEM Constructor

The `Scotem constructor` is responsible for constructing the test model SCOTEM from the UML collaboration diagram and UML statecharts. These UML models are provided as input to the system in XMI format [61]. The XMI format file representing the models can be generated by a number of UML case tools such as *`Borland's Together`* [60]. An XMI Parser has been written to populate the UML metamodel of collaboration diagrams and statecharts using the XMI files. The module generates the SCOTEM from these models using the algorithm presented in Figure 12.

The procedure `BuildTestModel()` in the algorithm takes as input a collaboration diagram $\c{C}$ and a set of flattened statecharts $\breve{S}$ for all the modal classes participating in the collaboration. The output is a SCOTEM test model for these UML artifacts. The procedure iteratively retrieves each message from the collaboration diagram following the message sequence numbering, and builds the SCOTEM.

As specified in Figure 12, the procedure iteratively builds the SCOTEM by adding a message to it within each iteration. Line 9 of the algorithm stores all possible states of the source object in which a message can be received. If the instance set of the target class already contains some instances and is modal, it implies that this class has already received some messages. The current state of this instance set must be a subset of the set of all the states capable of receiving the message. A violation of this condition implies a design inconsistency because the previous operation(s) left the object in an inconsistent state for the subsequent messages (line 13). Thus, the process of building SCOTEM from UML artifacts can also be used to detect inconsistencies in the design of the system.

Once these consistency checks have been performed, all the vertices corresponding to the instances in the instance set of the source class are connected to each vertex for the objects in the target instance set and message edges are added on each edge. The `SCOTEM Constructor` algorithm has been used to construct the SCOTEM models for various examples, including the Arithmetic Tutor example presented in Section 3. It was found that it successfully detects design inconsistencies, and constructs test models from the UML artifacts.

4.2 Test Path Generator

`Test Path Generator` generates test paths from the test model developed by the `SCOTEM Constructor`. The inputs to this module are the SCOTEM model and one of the four path coverage criteria mentioned in Section 3.4.1: *Single-Path Coverage, All-Transition Coverage, n-Path Coverage,* or *All-Path Coverage.* A test path generation algorithm for *All-Path Coverage* criterion is given in section 3.3, while the algorithms for the other three are given in Appendix A.

In addition to the basic coverage criterion for test paths, the *Test Path Generator* also allows the user to specify coverage criteria for loops and predicates. The output of the *Test Path Generator* is a set of test paths that meets the specified coverage criteria.

4.3 Test Executor

The execution of test paths requires test data to be generated. At this stage, test data is generated manually for the test paths using the state invariants. We require test data for the values of parameters in initial message calls and values of instance variables to set states of classes involved in collaboration diagrams. The user manually generates test values for each test path, by randomly picking values from state invariants and saves them in a text file. The `Test Executor` constructs concrete test cases by filling in the test data in the method calls in test paths. Each test case is then executed on the implementation and the execution results are logged in a file. The execution log for a test path consists of *before* and *after* states of the objects, for each message in a test path. The object states are determined from the instrumentation provided in the source code for getting object states using the state invariants for each object class.

```
Algorithm     BuildTestModel(Ç, Š): Ť
Inputs        Ç: Collaboration Diagram corresponding to a message
              Š: Set of statecharts required for classes involved in a collaboration
              diagram Ç
Output        Ť: SCOTEM test model generated by the algorithm
Assumptions   - All state-charts are flattened and contain call events only
              - The collaboration diagram has synchronous call messages and composition
                visibility only
Declare       RESULTANTSTATES$_{Set}$: A set of resultant states corresponding to a message
              STATE$_{Set}$: A set of states of a particular class
              CURRENTSTATESOURCE$_{Set}$: A set of states of source class of current message of
              collaboration diagram
              CURRENTSTATETARGET$_{Set}$: A set of states of target class of current message of
              collaboration diagram
              ME$_{Seq}$: Sequence of message edges of collaboration diagram Ç
              msg: A message edge in Ť
              v, v$_{source}$,v$_{rs}$: All these variables represent vertices in a SCOTEM test model
              addVertex(v, Ť): A function that adds a vertex v to the SCOTEM test model Ť
              addMessageEdge(v$_{source}$ , v$_{target}$): A function that adds a message edge from
              source vertex to target vertex including loop and path condition if any
              exists addTransitionEdge(v$_{source}$ , v$_{target}$): A function that adds a transition
              edge from source vertex to target vertex including guard if any exists
              getVertex(state,class): A function that returns a vertex of test model Ť
              representing class in a state
              createVertex(state,class): A function that creates a vertex of test model Ť
              representing class in a state

begin
   v <- null
   addVertex(v,Ť)
   CURRENTSTATESOURCE$_{Set}$ <- null
   ME$_{Seq}$ <- Ç.message
   RESULTANTSTATES$_{Set}$ <-  {}
   for all msg ∊ ME$_{Seq}$ do
         if ( not msg.source = null)
               RESULTANTSTATES$_{Set}$ -> Select(state:State¦Select(s:Š¦s.class
               = msg.target).transition = msg and state=transition.sourceState)
         end If
         STATE$_{Set}$ -> Select(s:Š¦s.class = msg.target).state
         if (not RESULTANTSTATES$_{Set}$ ⊆ STATE$_{Set}$)
               Report Inconsistency
         end If
         CURRENTSTATETAGET$_{Set}$ -> Select(state:State¦Select(s:Š¦s.class =
         msg.target).transition = msg)
         for all os ∊ CURRENTSTATESOURCE$_{Set}$ do
               v$_{source}$ <- getVertex(os,msg.source)
               for all ot ∊ CURRENTSTATETARGET$_{Set}$ do
                     v <- createVertex(ot,msg.target)
                     addVertex(v,Ť)
                     addMessageEdge(v$_{source}$ , v)
                     CURRENTSTATES$_{source}$ <- RESULTANTSTATES$_{Set}$ <- Select(state:State¦
                     Select(s:Š¦s.class = msg.target).transition = msg and
                     state=transition.targetState)
                     for all rs ∊ RESULTANTSTATES$_{Set}$ do
                           if (not rs ∊ Ť)
                                 v$_{rs}$ <- createVertex(rs,msg.target)
                                 addVertex(v$_{rs}$,Ť)
                           end If
                           addTransitionEdge(v,v$_{rs}$)
                     end for
               end for
        end for
    return Ť
end
```

Figure 12. Algorithm for SCOTEM construction: `BuildScotem()`

4.4 Results Evaluator

The `Results Evaluator` is responsible for comparing the results of a test run with the expected results. The expected results are derived from the generated test paths. An expected result for a test path consists of the object states *before* and *after* each message in the test path. The `Results Evaluator` compares the objects states in the execution log with those of an expected result. A test path is considered *Passed* if all the object states match, otherwise it is considered *Failed*. The *Pass/Fail* results are logged in a file. For failed test cases, the `Results Evaluator` also logs the test path number, the failed message whose expected state doesn't match with resultant state, and object states.

5 Case Study

In order to validate the effectiveness of our proposed approach, we implemented the Arithmetic Tutor (AT) example of Section 3 using the Java language, and generated mutant programs by seeding faults using mutation operators. This strategy is well established for assessing and comparing test techniques and has shown to yield useful results [19]. Table 3 below summarizes the characteristics of Java classes in the code for AT. The source code is given in Appendix C.

5.1 Experimental Setup

In order to seed faults in the code, we used 12 mutation operators. Some of these operators such as *Replace Return Statement* and *Remove Function Call* are defined by Delamaro et al. [9]. *Wrong Initial State*, *Condition Missing*, and *Loop Error Set* are defined by Souza et al. [16]. *Parameter Changed Operator (PCO)*, *Alter Condition Operator (ACO)*, and *Guard Condition Violated (GCV)* are implemented in MuJava [13]. In order to introduce more variety in the type of seeded faults, we propose a number of additional operator categories described below: *Missing Called Function, Wrong Calling State, Conflicting States Operator,* and *Target State as Source State*. The selection of seeded faults was based on the charateristics of the code for the AT case study, and the ability of the operators to introduce interaction faults, i.e., the faults that can only be revealed by messages sent from one class to the other and not by testing any of the classes in isolation. For instance, *Wrong Initial State (WIS)* introduces an interaction fault by changing the state of the recipient object of a message.

Table 3: Classes considered in case study

S#	Class	Methods	Instance variables	Modality	Lines of Code	Super Class
1	Coordinator.java	3	2	Non-Modal	23	Object
2	ProgramGenerator.java	2	3	Non-Modal	24	Object
3	OperandGenerator.java	2	1	Non-Modal	24	Object
4	Problem.java	6	2	Non-Modal	33	Object
5	ComplexityRegulator.java	9	2	Modal	47	Object
6	DisplayManager.java	5	4	Modal	46	Object
7	StopWatch.java	5	3	Modal	75	java.lang.Thread
8	PerformanceRegulator.java	3	2	Modal	32	Object
9	Log.java	3	2	Modal	35	Object

The following is a complete list of the mutation operators that we used in the case study:

1. *Parameter Changed Operator (PCO):* This operator [13] changes the parameter passed in an operation call. The valid value of the parameter is replaced with an invalid value.

2. *Wrong Initial State (WIS):* This operator changes the initial state of an object before it receives a message. The initial state is replaced by an invalid state, in which the object should not receive a particular message [16].

3. *Replace Return Statement (RetStaDel):* This operator is proposed in [9]. It replaces each return statement in a method with each of the other return statements in the method, one at a time.

4. *Remove Function Call (FunCalDel):* This operator is also proposed in [9]. It removes each function call in a method, one at a time.

5. *Condition Missing (CM):* This operator removes the condition of a conditional message in the code [16].

6. *Loop Error Set (LES):* This is a set of loop mutation faults. This set of operators forces the loop control variable to be executed an incorrect number of times. This operator executes the loop exactly n+1 times (where n is the maximum number of times a loop can be executed), n-1 times, 0 times, and 1 time [16].

7. *Alter Condition Operator (ACO):* This operator [13] changes the condition in the code corresponding to a path condition in a collaboration.

8. *Guard Condition Violated (GCV):* This operator [13] negates the guard condition of a transition.

9. *Missing Called Function (MCF):* This operator removes the functions that are called by an object, one at a time.

10. *Target State as Source State (TSSS):* This operator sets the target state of an object as source state before sending a message.

11. *Wrong Calling State (WCS):* This operator sets the state of the calling object to an invalid state.

12. *Conflicting State Operator (CSO):* This operator sets the states of two objects of different classes in states that are conflicting with each other.

The number of faults seeded in each category was based on the application design and code of the AT system. For instance, the statecharts for our case study have six guard conditions and, therefore, six faults of the *GCV* category could be seeded. Similarly, there are two path conditions in the collaboration diagram, so we could seed only two faults in each of *CM* and *ACO* category. In our case study, we used a collaboration corresponding to the operation call `newProblem()`, which has a `String` parameter. Therefore, we seeded six *PCO* faults by providing six invalid values for the parameter. A summary of the seeded faults for the case study is given in Table 4.

Table 4: Number and types of faults seeded

No	*Mutant Operator*	*Number of Faults Seeded*
1	PCO	6
2	WIS	1
3	RetStaDel	4
4	FunCalDel	5
5	CM	2
6	LES	3
7	ACO	1
8	GCV	8
9	MCF	6
10	TSSS	5
11	WCS	4
12	CSO	4

We created 49 versions of the program (mutant programs), with each version containing one of the 49 faults. Out of these 49 faults, 13 turned out to lead to corrupt/wrong source or target object states (state faults). Test paths were generated using *Single-Path Coverage, All-Transition Coverage, n-Path Coverage,* and *All-Path Coverage* criteria. We chose a value of 82 for n in the *n-Path Coverage* criterion as this is an intermediary number (and therefore a compromise in terms of cost) between the number of paths required for *All-Transition* and *All-Path Coverage*. However, any other value of n less than the total number of paths could have been used. In terms of testing cost, for that criterion to make economical sense, n should be significantly lower than the total number of paths.

The test data were manually generated for each test path, and then the test paths were executed on each of the 49 versions, using our prototype tool. In order to ensure integrity of the empirical results, the test data were generated by a different team of testers who were not aware of the faults seeded in the code. Since different test suites could be generated for *Single-Path Coverage, All-Transition Coverage,* and *n-Path Coverage* criteria, with varying effect on the number of mutants killed, to provide a realistic evaluation of each coverage criterion we chose 10 randomly selected test suites for each coverage criterion, and then analyzed the minimum, average, and maximum numbers of mutants killed by each test suite. This is not necessary for *All-Path Coverage* since it only yields one possible test suite.

5.2 Test Results and Discussion

The *All-Path Coverage* criterion was able to kill all 49 mutants. For the other three coverage criteria, the number of mutants killed for each of the ten test suites is shown in Table 5 along with the number of paths tested, that we assume to be proportional to the cost of testing. Note that we generated 10 random test suites for each coverage criterion in order to capture the randomness usually associated with their fault detection. Table 6 summarizes the results by providing the minimum, average, and maximum number of mutants killed by each coverage criterion across the 10 randomly generated test suites. Figure 13 depicts a bar chart representation of the minimum, average and maximum mutation scores for each coverage criterion.

Table 5: Number of mutants killed in each test suite

Coverage Criterion	**Mutant Scores for each of the 10 randomly chosen test suites**	**Number of test paths**
Single-Path Coverage	36, 37, 39, 38, 36, 37, 38, 36, 37, 38	1
All-Transition Coverage	41, 45, 45, 45, 45, 46, 46, 46, 46, 44	3
n-Path Coverage (n=82)	49, 43, 49, 43, 49, 47, 49, 44, 49, 49	82

Table 6: Mutation score against coverage criteria

	Single-Path Coverage	**All-Transition Coverage Coverage**	**n-Path Coverage**	**All-Path Coverage**
Min	73% (36)	83% (41)	88% (43)	100% (49)
Avg	75% (37)	91% (45)	95% (47)	100% (49)
Max	79% (39)	93% (46)	100% (49)	100% (49)

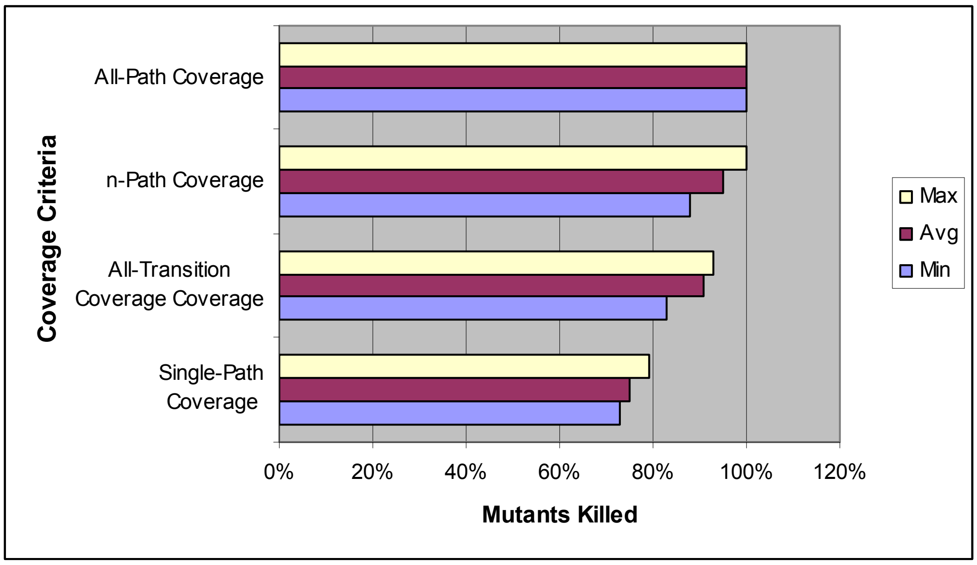

Figure 13. Mutation score against coverage criteria

Single-Path Coverage
Depending on the test path chosen, the *Single-Path Coverage* criterion detected from 36 to 39 faults out of a total of 49 faults. It detected all faults of type *PCO, WIS, RetStaDel, FunCalDel, CM, LES, ACO, MCF,* and *TSSS.* However, it could not completely detect faults of type *GCV, WCS,* and *CSO*. In all of the test suites executed for *Single-Path Coverage*, only one fault from each of the *WCS* and *CSO* categories was detected.

All-Transition Coverage
Each test suite in *All-Transition Coverage* criterion consisted of three test paths. As expected, this criterion produced better results (+16% in fault detection on average) than *Single-Path Coverage*. It detected all faults for all mutation operators, except *WCS* and *CSO.* The total number of detected faults ranged from 41 to 45, depending on the test suite chosen.

n-Path Coverage
We set the value of *n* to *82*, which is the middle of the range between 3 (*All-Transition Coverage*) and 162 (*All-Path Coverage*) -- the *n-Path Coverage* is supposed to be a compromise when *All-Path* Coverage is too expensive. The results produced were better than those of the *All-Transition* Coverage criterion by 4% on average. The total number of faults detected ranged from 46 to the maximum of 49. Some of the *WCS* and *CSO* faults were still left undetected in four of the ten test suites.

All-Path Coverage
The only test suite generated by the *All-Path Coverage* criterion consisted of all 162 paths, and was able to detect all 49 of the seeded faults. This criterion is however much more expensive than the previous ones.

Analysis of the Results
Further analysis of the coverage criteria and the test results showed that each test path consisted of all messages in the original collaboration diagram (this is because there were no mutually exclusive messages in the collaboration diagram). Executing each message of the collaboration therefore ensures that all faults of type *PCO, WIS, RetStaDel, FunCalDel, CM, LES, ACO, MCF,* and *TSSS* are detected. As a result, faults of these nine categories were detected by all coverage criteria. However, the mutation operators *GCV, WCO,* and *CSO* resulted in state faults due to wrong state transitions. To detect a state fault, it is necessary that certain object states be covered by at least one test path.

In our case study, such state faults could not be completely detected by a single test path, even though the path included all messages of the collaboration. This could be expected as the particular combinations of object states required to detect such faults were not completely covered by any of the single test paths. *Single-Path Coverage* was able to detect 75% of the faults on average. All test suites generated by *Single-Path Coverage* were able to detect some state faults in certain cases and all other faults. At the most, *Single-Path Coverage* was able to detect three state faults. *All-Transition Coverage* detected more faults (91% on average) and was able to detect 10 state faults at the most. As expected, *n-Path Coverage* achieved an intermediary result between *All-Transition Coverage* and *All-Path Coverage*. From the 10 test suites achieving n-Path Coverage, six test suites were able to detect all of the faults. Only four test suites missed a few of the state faults. This coverage criterion detected 95% of the faults on average. This means that, on average, *n-Path Coverage* was able to detect four additional state faults when compared to *All-Transition Coverage*. *All-Path Coverage* was able to systematically detect all of the faults.

From the above discussion, we see that to ensure that all state faults are detected; the test suite must include test paths that cover all valid combinations of object states. The case study results therefore provide a strong justification for our proposed test strategy. However, because of the cost associated with the *All-Path* criterion, future work will have to investigate strategies to select a subset of combinations while still detecting all or most of the state faults.

6 Conclusion and Future Work

In this paper, we have presented a new strategy for class integration testing that is based on a test model (SCOTEM) that combines information from collaboration and statechart diagrams into the form of a graph. The motivation is to exercise class interactions in the context of multiple state combinations in order to detect state faults. Therefore, it takes into account the states of all objects involved in a collaboration to exercise class interactions in the context of integration testing. For instance, if the functionality provided by an object depends on the states of other objects (including the caller object), then the proposed technique can effectively detect faults occurring due to invalid object states.

We ran a carefully designed case study using a prototype tool and generated 49 faulty versions of the system under test using carefully selected mutation operators. The empirical results show that the proposed approach effectively detects various kinds of integration faults. In particular, the *All-Path Coverage* criterion successfully detected all of the seeded faults and is particularly effective at detecting faults related to the state-behavior of interacting classes. A detailed analysis of live mutants confirmed our suspicion that finding certain categories of integration faults required exercising all possible object state combinations involved in a collaboration, thus justifying our test strategy.

We have also presented various coverage criteria to generate test paths, and algorithms to automate generation of test paths. The most demanding criterion, *All-Path Coverage*, is however very expensive and it is not clear that it can scale up in all situations. It is therefore important that less expensive criteria be carefully investigated in future work as they will likely be more suitable in many situations where test budgets are limited. Now that the potential of our SCOTEM approach has been demonstrated, new criteria to select high yield subsets of the SCOTEM test paths should be devised.

An additional limitation is that the case study presented in this paper, though far from simple, may not be representative of an industrial system. Industrial case studies are required to carefully analyze the cost-benefit of the proposed integration testing strategy in a realistic context.

Acknowledgements

Professor M. Jaffar-ur-Rehman played a significant role in this work. For students, Dr. Jaffar was a mentor, a source of knowledge, inspiration, and motivation. His tragic death on October 2005 in devastating earthquakes in Islamabad, Pakistan has created a big hole in the lives of many of his colleagues in academia and industry. He touched hearts of people by his kindness, humbleness, intelligence, knowledge, and above all his nature to help everyone. His absence has saddened many hearts and is being immensely felt. May God give him highest merits in heaven (Amin).

7 References

[1] R.V. Binder, Testing Object-Oriented Systems - Models, Patterns, and Tools, Addison-Wesley, 2000.
[2] B. Beizer, Software Testing Techniques, 2nd Edition, Van Nastrand Ranhald, 1990.
[3] B. Bruegge, A.H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, Prentice Hall, Second Edition, 2003.
[4] G.J. Myers, The Art of Software Testing, John Wiley and Sons, 1979.
[5] A. Baldini, A. Benso, P. Prinetto, System-level Functional Testing from UML Specifications in End-of-production Industrial Environments, International Journal of Software Tools Technology and Transfer, Vol. 7 (4), 2004, pp. 326-340.
[6] L. Briand, Y. Labiche, Y. Wang, An Investigation of Graph-Based Class Integration Test Order Strategies, IEEE Transactions on Software Engineering, Vol. 29 (6), 2003, pp. 594 - 607.
[7] L.C. Briand, M. Di Penta, Y. Labiche, Assessing and Improving State-Based Class Testing: A Series of Experiments, IEEE Transactions on Software Engineering, Vol. 30 (11), 2004, pp. 770-793.
[8] A. Cavalli, S.M. Maag, S. Papagiannaki, G. Verigakis, From UML Models to Automatic Generated Tests for the dotLRN e-learning Platform, Electronic Notes in Theoretical Computer Science, Vol. 116 (1), 2004, pp. 133-144.
[9] M.E. Delamaro, J.C. Maldonado, A.P. Mathur, Interface Mutation: An Approach for Integration Testing, IEEE Transactions on Software Engineering, Vol. 27 (3), March 2001, pp. 228-247.
[10] L. Gallagher, A.J. Offutt, A. Cincotta, Integration Testing of Object-oriented Components using Finite State Machines, Journal of Software Testing, Verification, and Reliability, January 2006.
[11] Y.G. Kim, H.S. Hong, D.H Bae, S.D. Cha, Test cases generation from UML State Diagrams, IEEE Software, Vol. 146(4), 1999, pp.187-192.
[12] Y. Le Traon, T. Jeron, J.M. Jezequel, P. Morel, Efficient Object-oriented Integration and Regression Testing, IEEE Transactions on Reliability, Vol. 49 (1), March 2000, pp. 12-25.
[13] Y. Ma, J. Offutt, Y. Kwon, MuJava: An Automated Class Mutation System, Journal of Software Testing, Verification and Reliability, Vol. 15 (2), June 2005, pp. 97-133.
[14] J. Offutt, S. Liu, A. Abdurazik, P. Ammann, Generating Test Data from State-based Specifications, Journal of Software Testing, Verification and Reliability, Vol. 13 (1) , 2003, pp. 25-53.
[15] A.S.M. Sajeev, B. Wibowo, UML Modeling for Regression Testing of Component Based Systems, Electronic Notes Theoretical Computer Science, Vol. 82(6), 2003, pp.1-9.
[16] S.R.S. de Souza, S.C.P.F. Fabbri, W.L. de Souza, J.C. Maldonado, Mutation Testing Applied to Estelle Specifications, Software Quality Journal, Vol. 8 (4), 1999, pp. 285-301.
[17] A. Abdurazik, J. Offutt, Using UML Collaboration Diagrams for Static Checking and Test Generation, Proceedings of the Third International Conference on the Unified Modeling Language (UML '00), York, UK, October 2000, pp. 383-395.
[18] P. Ammann, J. Offutt, and H. Huang, Coverage Criteria for Logical Expressions, Proceedings of the 14th International Symposium on Software Reliability Engineering, (ISSRE'03), 2003, pp. 99-107.
[19] J.H. Andrews, L.C. Briand and Y. Labiche, Is Mutation an Appropriate Tool for Testing Experiments?, Proceedings of the IEEE 27th International Conference on Software Engineering (ICSE) 2005, St. Louis, Missouri, USA, May 2005, pp. 15-21.
[20] F. Basanieri, A. Bertolino, A Practical Approach to UML-Based Derivation of Integration Tests, Proceedings of the Software Quality Week QWE2000, 2000.
[21] F. Basanieri, A. Bertolino, E. Marchetti, COWTest: Cost Weighted Test Strategy, Proceedings of the ESCOM-SCOPE, London, England, 2001, pp. 387-396.
[22] A. Bertolino, E. Marchetti, Introducing a Reasonably Complete and Coherent Approach for Model-based Testing, Proceedings of the International Workshop on Testing and Analysis of Component-based Systems (TACoS 2004), Electronic Notes in Theoretical Computer Science, 2004, pp. 85-97.
[23] L. Briand, Y. Labiche, A UML-Based Approach to System Testing, Proceedings of the Fourth International Conference on the Unified Modeling Language (UML'01), 2001, pp. 194-208**.**
[24] L. Briand, Y. Labiche, G. Soccar, Automating Impact Analysis and Regression Test Selection Based on UML Designs, Proceedings of the International Conference on Software Maintenance (ICSM'02), Canada, 2002, pp. 252-261.
[25] L.C. Briand, J. Cui, Y. Labiche, Towards Automated Support for Deriving Test Data from UML Statecharts, Proceedings of the ACM/IEEE International Unified Modeling Language conference (UML 2003), July 2003, pp. 249-264.
[26] U. Buy, A. Orso, M. Pezze, Automated Testing of Classes, Proceedings of the International Symposium on Software Testing and Analysis, ACM Press, 2000, pp. 39–48.
[27] P. Chevalley, P. Thevenod-Fosse, Automated Generation of Statistical Test Cases from UML State Diagrams, Proceedings of the 25th Annual International Computer Software and Applications Conference (COMPSAC'01), 2001, pp. 205-214.
[28] F. Fraikin, T. Leonhardt, SeDiTeC — Testing Based on Sequence Diagrams, Proceedings of the 17th IEEE International Conference on Automated Software Engineering (ASE'02), 2002, pp. 261-266.

[29] P. Frohlich, J. Link, Automated Test Case Generation from Dynamic Models, Proceedings of the 14th European Conference on Object-Oriented Programming, 2000, pp: 472 – 492.
[30] J. Hartmann, C. Imoberdorf, M. Meisinger, UML-Based Integration Testing, Proceedings of the International Symposium on Software Testing and Analysis (ISTA'00), 2000, pp. 60 - 70.
[31] A. Hartman, K. Nagin, AGEDIS Tools for Model-Based Testing, Proceedings of the International Symposium on Software Testing and Analysis (ISTA'04), pp. 129-132.
[32] S. Gnesi, D. Latella, M. Massink, Formal Test-Case Generation for UML Statecharts, Proceedings of the Ninth IEEE International Conference on Engineering Complex Computer Systems (ICECCS'04), 2004, pp. 75- 84.
[33] S. Kansomkeat, W. Rivepiboon, Automated-Generating Test Case Using Statechart Diagrams Test Case Using UML Statechart Diagrams, Proceedings of the Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on Enablement through Technology, 2003, pp. 296-300.
[34] S. Kim, L. Wildman, R. Duke, A UML Approach to the Generation of Test Sequences for Java-based Concurrent Systems, Proceedings of the 2005 Australian Software Engineering Conference (ASWEC'05), 2005, pp. 100-109.
[35] D.C. Kung, N. Suchak, P. Hsia, Y. Toyoshima, C. Chen, On Object State Testing, Proceedings of the 17th Annual International Computer Software and Applications Conference (COMPSAC'94), 1994.
[36] D.C. Kung, P. Hsia, Y. Toyoshima, C. Chen, J. Gao, Object-Oriented Software Testing- Some Research and Development, Proceedings of the 3rd IEEE International Symposium on High-Assurance Systems Engineering (HASE'98), November 1998, pp. 158 - 165.
[37] V. Le Hanh , K. Akif, Y. Le Traon , J.M. Jezequel, Selecting an Efficient OO Integration Testing Strategy: An Experimental Comparison of Actual Strategies, Proceedings of the 15th European Conference on Object-Oriented Programming (ECOOP 2001), Lecture Notes In Computer Science; Vol. 2072, pp. 381 – 401.
[38] S. Li, J. Wang, Z. Qi, Property Oriented Test Generation from UML Statecharts, Proceedings of the 19th International Conference on Automated Software Engineering, 2004, pp. 122- 131.
[39] W. Linzhang, Y. Jiesong, Y. Xiaofeng, H. Jun, L. Xuandong, Z. Guoliang, Generating Test Cases from UML Activity Diagram based on Gray-Box Method, Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC'04), 2004, pp. 284 - 291.
[40] L. Liuying, Q. Zhichang, Test Selection from UML Statecharts, Proceedings of the 31st International Conference on Technology of Object-Oriented Language and Systems, 1999, pp. 273-280.
[41] V. Martena, A. Orso, M. Pezzè, Interclass Testing of Object Oriented Software, Proceedings of IEEE International Conference on Engineering of Complex Computer Systems (ICECCS), 2002, pp. 135- 144.
[42] J. Offutt, A. Abdurazik, Generating tests from UML specifications, Proceedings of the Second IEEE International Conference on the Unified Modeling Language (UML99), October 1999, pp. 416-429.
[43] T.S. Ostrand, E.J. Weyuker, Using Data Flow Analysis for Regression Testing, Proceedings of the Sixth Annual Pacific Northwest Software Quality Conference, September 1988, pp. 233-247.
[44] P. Pelliccione, H. Muccini, A. Bucchiarone, F. Facchini, TeStor: Deriving Test Sequences from Model-based Specifications, Proceedings of the Eighth International SIGSOFT Symposium on Component-based Software Engineering (CBSE 2005), Lecture Notes in Computer Science, LNCS 3489, 2005, pp. 267-282.
[45] S. Pickin, C. Jard, Y. Le Traon , T. Jéron, J.M. Jézéquel, A. Le Guennec, System Test Synthesis from UML Models of Distributed Software, Proceedings of the 22nd IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems (FORTE 2002), Lecture Notes in Computer Science, November 2002, pp. 97-113.
[46] O. Pilskalns, A. Andrews, R. France, S. Ghosh, Rigorous Testing by Merging Structural and Behavioral UML Representations, Proceedings of the Sixth International Conference on the Unified Modeling Language (UML 2003), 2003, pp. 234-248.
[47] M. Riebisch, I. Philippow, M. Götze, UML-Based Statistical Test Case Generation, Proceedings of the International Conference on Objects, Components, Architectures, Services, and Applications for a Networked World, LNCS 2591, September 2003, pp. 394-411.
[48] M. Scheerz, A.V. Mayrhauser, R. France, E. Dahlman, A.E. Howe, Generating Test Cases from an OO Model with an AI Planning System, Proceedings of the 10th International Symposium on Software Reliability Engineering (ISSRE'99), November 1999, pp. 250-260.
[49] J. Wittevrongel, F. Maurer, Using UML to Partially Automate Generation of Scenario-Based Test Drivers, Proceedings of the 7th International Conference on Object Oriented Information Systems (OOIS'01), 2001, pp. 303-306.
[50] M. Badri, L. Badri, M. Naha, A Use Case Driven Testing Process: Towards a Formal Approach Based on UML Collaboration Diagrams, Proceedings of the 3rd International Workshop on Formal Approaches To Testing of Software FATES 2003, Lecture Notes in Computer Science, Springer-Verlag Berlin Heidelberg, pp. 223-235.
[51] M.E. Vieira, M.S. Dias, D.J. Richardson, Object-Oriented Specification-Based Testing Using UML State-chart Diagrams, Proceedings of the Workshop on Automated Program Analysis, Testing, and Verification (at ICSE'2000), June 2000.
[52] K.E. Bogdanov, Automated Testing of Harel's statecharts, PhD Thesis, University of Sheffield, 2000.
[53] S. Burton, Towards Automated Unit Testing of Statechart Implementations, Technical Report (YCS 319), Department of Computer Science, University of York. September 1999.

[54] M. Nilawar, A UML-Based Approach for Testing Web Applications, Master's thesis ,University of Nevada, Reno, 2003.
[55] A. Nori, A. Sreenivas, A Technique for Model-Based Testing of Classes, Technical Report, Tata research, Development and Design Centre, Pune, India, 2001.
[56] D. Sokenou, S. Herrmann, Using Object Teams for State-Based Class Testing, Technical Report, Bericht-Nr. 2004/10, Fak. IV, Technical University Berlin, 2004.
[57] T.H. Tse, Z. Xu, Class-Level Object-Oriented State Testing: A Formal Approach, HKU CSIS Technical Report TR-95-05, 1995.
[58] Y. Wu, J. Offutt, Maintaining Evolving Component-Based Software with UML, Technical Report, ISE-TR-02-07, http://www.isse.gmu.edu/techrep/2002 /02_07.pdf.
[59] http://www.objectteams.org
[60] http://www.borland.com/together/
[61] http://www.omg.org/technology/documents/formal/xmi.htm

Appendix-A: Test Path Generation Algorithms

A-1 Single-Path Coverage Algorithm

Figure 14 below shows algorithm for *Single-Path Coverage* Criteria

```
Algorithm     SinglePathCoverage(Ť): tp
Input         Ť: SCOTEM Test Model
Output        tp : A string (OCL 1.5) representing a test path
Declare       $TPS_{Seq}$: A Sequence (OCL 1.5) of test paths
              $MES_{Seq}$: A Sequence (OCL 1.5) of message edges in Ť
              medg: A message edge in Ť
              tedg: A transition edge in Ť
              tpm: Test sub path corresponding to a message edge of type String (OCL 1.5)
              tpt: Test sub path corresponding to transition edge of the current edge of
              type String (OCL 1.5)
              $TME_{Seq}$ : A sequence (OCL 1.5) of transition edges corresponding to a message
              edge
              tp : A string (OCL 1.5) representing a single test path
              random($TME_{Seq}$):returns a random transition from $TME_{Seq}$


1.  begin
2.      $TPS_{Seq}$ <- {}
3.      tp <- ""
4.      $MES_{Seq}$ <- Ť.edge.message
5.      for all medg ∈ $MES_{Seq}$ do
6.              tpm <-medg.sequenceNumber+":"+medg.constraint+medg.associatedOperation
                      +"$"+medg. receiverClass+"@"
7.              tp -> concat(tpm)
8.              if medg.vertex.oclIsTypeOf(ModalVertex)
9.                      $TME_{Seq}$ <- medg.transition
10.                     tedg <- random($TME_{Seq}$)
11.                     tpt <- tedg.sourceState+"->"+ tedg.guard+
                                     tedg.targetState
12.                     tp -> concat(tpt)
13.             else
14.                     tp -> concat(medg.receiverClass))
15.             end if
16.     end for
17.   return tp
18. end
```

Figure 14. Algorithm for test paths generation for Single-Path Coverage criterion

Explanation of the algorithm

The routine takes SCOTEM as an input and returns a set of test paths as an output. Lines 5 to 14 generate a test path. Random transitions of modal classes are picked up by calling to a function `random()`.

By applying this criterion on SCOTEM of Figure 9, the following test path is generated.

```
1.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle
```

A-2 All-Transition Coverage Algorithm

Figure 15 shows contains algorithm for *All-Transition Coverage*

```
Algorithm     AllTransiionCoverage(Ť): $TPS_{Seq}$
Input         Ť: SCOTEM Test Model
Output        $TPS_{Seq}$: A sequence (OCL 1.5) of test paths
Declare       $TPS_{Seq}$: A sequence (OCL 1.5) of test paths
              $MES_{Seq}$: A sequence (OCL 1.5) of message edges in Ť
              medg: A message edge in Ť
              tedg: A transition edge in Ť
              tpm: Test sub path corresponding to a message edge of type String (OCL 1.5)
              tpt: Test sub path corresponding to transition edge of the current edge of
              type String (OCL 1.5)
              $TME_{Seq}$ :A sequence (OCL 1.5) of transition edges corresponding to a message
              edge
              getMaxTransitionNumber(Ť): A function that returns an Integer (OCL 1.5)
              representing maximum transitions a modal class has from all modal classes
              in SCOTEM Ť
              random($TME_{Seq}$): A function that returns a random transition from $TME_{Seq}$

1. begin
2.     $TPS_{Seq}$ <- {}
3.     $MES_{Seq}$ <- Ť.edge.message
4.     for (i=1 to getMaxTransitionNumber(Ť))
5.           tpm <- medg.sequenceNumber+":"+medg.constraint+medg.associatedOperation
                  +"$"+medg. receiverClass+"@"
6.           for all medg ∈ $MES_{Seq}$ do
7.                 $TPS_{Seq}$ ->insertAt(i,tpm)
8.                 if medg.vertex.oclIsTypeOf(ModalVertex)
9.                       $TME_{Seq}$ <- medg.transition
10.                      tedg<- random($TME_{Seq}$)
11.                      tpt <- tedg.sourceState+"->"+ tedg.guard+
                               tedg.targetState
12.                      $TPS_{Seq}$->insertAt(i, $TPS_{Seq}$ ->at(i)->concat(tpt))
13.                else
14.                      $TPS_{Seq}$->insertAt(i, $TPS_{Seq}$ ->at(i)->concat(medg.receiverClass))
15.
16.                end if
17.          end for
18.    end for
19. return $TPS_{Seq}$
20. end
```

Figure 15. Algorithm for test paths generation for All-Transition Coverage criterion

Explanation of the algorithm

The routine takes SCOTEM as an input and returns a set of test paths as an output. The loop at line 4 executes for number of transition paths in a modal class that has maximum number of transition paths. The `Ť.edge.message` returns all message edges in order of message sequence number. A test path is generated from lines 7 to 16. The function `getUniqueTransition()` returns a transition of a modal class that is not tested before. When all transition edges of a modal class have been tested then for remaining test paths random transitions are selected. This check is performed at line 10 and function `random()` selects a random transition.

By applying the above algorithm to the SCOTEM, the following test paths are generated:

```
1.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
```

```
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle


2.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@NotFull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

3.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced
→Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@NotFull→[ questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle
```

A-3 n-Path Coverage Algorithm

Figure 16 shows algorithm for *n-Path Coverage* Criteria

Explanation of the algorithm

The routine takes SCOTEM as an input and returns a set of test paths as an output. This is a flexible coverage criterion, which generates test paths based on a parameter provided by the user. Lines 3 and 4 handles the situation when value of parameter is 1 and *n-Path Coverage* becomes equal to *Single-Path Coverage*. Sometimes *n-Path Coverage* criterion becomes equal to *All-Transitions Coverage* criterion. This situation is handled in Lines 5 and 6. *All-Path Coverage* criterion is dealt in (lines 7 and 81). Lines 15 to 19 handle a situation when value of parameter is greater than maximum number of transition paths in a modal class. In this case, n paths are generated from *AllTransitionEdges()*, while remaining paths are obtained randomly from SCOTEM. When parameter value is less than number of paths in *All-Transition Coverage* citeria then paths are randomly chosen from *All-Transition Coverage* criterion.

```
Algorithm     NPathCoverage(Ť,n): $TPS_{Seq}$
Input         Ť: SCOTEM Test Model
              n: Integer (OCL 1.5)
Output        $TPS_{Seq}$: A sequence (OCL 1.5) of test paths
Declare       $TPS_{Seq}$: A sequence (OCL 1.5) of test paths
              i: Integer (OCL 1.5)
              $TEMP_{Seq}$ : A sequence (OCL 1.5) that temporarily store test paths
              getMaxTransitionNumber(Ť): A function that returns an Integer (OCL 1.5)
              representing maximum transitions a modal class has from all modal classes
              in SCOTEM Ť
              getRandomPath($TEMP_{Seq}$): A function that takes an OCL 1.5 sequence as an
              input and returns a unique random element from the sequence

1.  begin
2.     $TPS_{Seq}$ <- {}
3.     if (n=1)
4.            $TPS_{Seq}$ -> insertAt(0,SinglePathCoverage(Ť))
5.      else if (n=getMaxTransitionNumber(Ť))
6.            $TPS_{Seq}$ -> AllTransitionEdges(Ť)
7.      else if ( n= AllPathCoverage(Ť)-> count() )
8.            $TPS_{Seq}$ -> AllPathCoverage(Ť)
9.      else if (n > getMaxTransitionNumber() )
10.           $TPS_{Seq}$ ->  AllTransitionEdges(Ť)
11.           for (i= getMaxTransitionNumber(Ť)+1 to n) do
12.                  $TPS_{Seq}$ -> append(getRandomPath(AllPathCoverage(Ť)))
13.           end for
14.    else
15.           $TEMP_{Seq}$ -> AllTransitionEdges(Ť)
16.           for (i=1 to n)
17.                  $TPS_{Seq}$ -> add($TEMP_{Seq}$ -> getRandomTransition($TEMP_{Seq}$))
18.           end for
19.    end If
20.    return $TPS_{Seq}$
21. end
```

Figure 16. Algorithm for test paths generation for n-Path Coverage criterion

For instance, when n = 4, the following four test paths will be generated:

```
1.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

2.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@NotFull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

3.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
```

```
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced
→Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

4.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle
.
```

Appendix B: Class Diagram of AT System

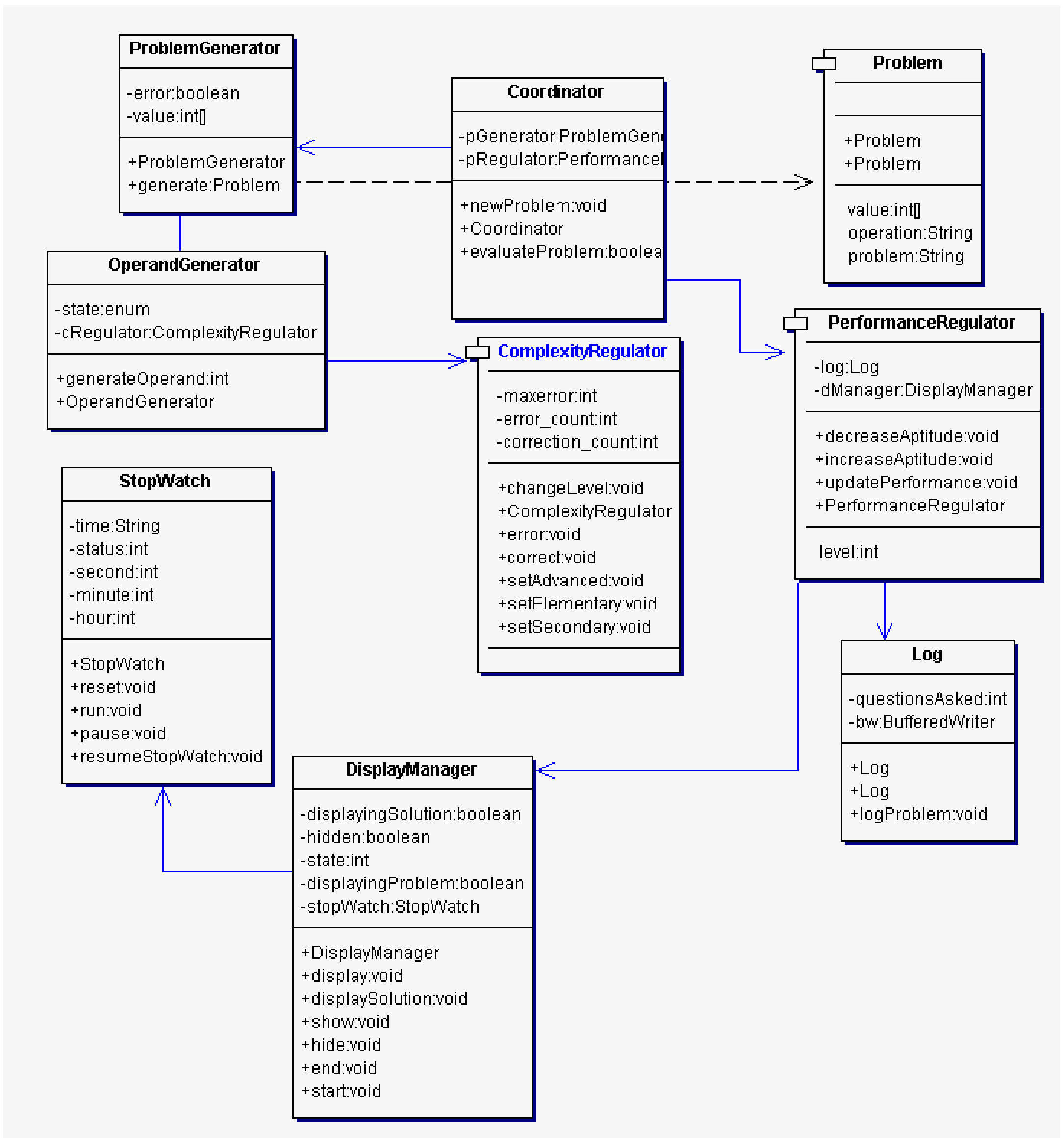

Figure 17. Class diagram for AT system

Appendix C: Code Listing for the Case Study

```
public class Coordinator
{
   ProblemGenerator pGenerator;
   PerformanceRegulator pRegulator;
   public Coordinator()
   {
           Regulator=new PerformanceRegulator();
           pGenerator=new ProblemGenerator();
   }
// This method generates a new problem Currently only + operations are Supported
   public void newProblem(String operation)
   {
           int[] value=pGenerator.generate(operation);
           if (value.length!=0)
                   pRegulator.updatePerformance(value);
   }
// This method evaluates the result of a Problem
   public boolean evaluateProblem(int userResult, int expectedResult)
{
           if (userResult==expectedResult)
                   return true;
                   else
                   return false;
   }
}
```

Code Listing 1: Coordinator.java

```
public class ComplexityRegulator
{
   private int level;
   private int error_count;
   public ComplexityRegulator()
   {
           level=Constants.ELEMENTARY;
           error_count=10;
   }

// This method return current complexity level

   public int getComplexity()
   {
           return level;
   }

// This method sets current complexity level to Elementary

   public void setElementary()
   {
   level=Constants.ELEMENTARY;
   }

// This method sets current complexity level to Secondary
public void setSecondary()
{
           level=Constants.SECONDARY;
}

//This method sets current complexity level to Advanced
public void setAdvanced()
{
    level=Constants.ADVANCED;
}

// This method sets error count  to current error count
public void setLevel(int level)
{
           this.level=level;
}

// This method increments error count
   public void error()
   {
   error_count++;
   }

// This method decrements error count

public void correct()
{
            error_count--;
}

// This method change complexity level based on current level

   public void changeLevel()
   {
            if (error_count<5)
                    setAdvanced();
            else if ((error_count>=5) && (error_count<10))
                    setSecondary();
            else
                     setElementary();
   }
}
```

Code Listing 2: ComplexityRegulator.java

```
public class DisplayManager
{
   private boolean hidden;
   private StopWatch stopWatch;
   private boolean displayingProblem;
   private boolean displayingSolution;
   public DisplayManager()
   {
            not hidden;
            stopWatch=  new StopWatch();
            displayingProblem = false;
            displayingSolution = false;
   }

// This method displays a problem and resets Stop Watch
   public void display(int[] value,int level)
   {

            if((displayingProblem==false)&&(displayingSolution==false)&&(level==0))
            {
                    displayingProblem = true;
                    displayingSolution = false;
                    hidden=false;
            }
            if ((displayingProblem==false)&&(displayingSolution==true))
            {
                    displayingProblem = true;
                    displayingSolution = false;
                    hidden=false;
            }
            stopWatch.reset(level);
}

// This method displays solution of problem

   public void displaySolution(int result)
   {
            displayingSolution = true;
            displayingProblem = false;
            not hidden;
            System.out.println("Result is = "+ result);
   }



// This method hides the display

public void hide()
{
   hidden=true;
}

// This method shows the display
public void show()
{
        not hidden;
    }
}
```

Code Listing 3: DisplayManager.java

```
public class OperandGenerator
{
   ComplexityRegulator cRegulator;
public OperandGenerator()
{
           cRegulator=new  ComplexityRegulator();
   }

// This method generates operands based upon current complexity
level
   public int generateOperand()
   {
           int level=cRegulator.getComplexity();
           int number;
           if (level==Constants.ELEMENTARY)
                   number= level+(int)Math.ceil(Math.random()*9999);
           else if (level==Constants.SECONDARY)
                   number= level+(int)
                   Math.ceil(10000+Math.random()*20000);
           else if (level==Constants.ADVANCED)
                   number= level+(int)
                   Math.ceil(20000+Math.random()*30000);
           else
                   number=-1;
           return number;
   }
}
```

Code Listing 4: OperandGenerator.java

```
public class Problem
{
   private String operation;
   private int value[];
   public Problem()
   {
   }

// Constructor that creates a problem based on values and
operation provided as arguments

public Problem(int[] value, String operation)
{
           this.value=value;
           this.operation=operation;
}

// Setter method for operation

public void setOperation(String operation)
{
           this.operation=operation;
}

// Setter method for operand
public void setValue(int[] value)
{
   this.value=value;
}

// getter method for operation
public String getOperation()
{
           return operation;
}

// getter method for operands
public int[] getValue()
{
           return value;
}
```

```java
  // returns problem as a string
  public String getProblem()
  {
     return Integer.toString(value[0])+operation+Integer.toString(value[1]);
  }
}
```

Code Listing 5: Problem.java

```java
  public class ProblemGenerator
  {
     private int value[];
     private boolean error;
     public ProblemGenerator()
     {
              value=new int[2];
              error=false;
              oGenerator=new OperandGenerator();
     }

  // This method generates a problem based on operation provided as
  an argument
  public int[] generate(String operation)
  {
              for (int i=0;i<2;i++)
              {
                       value[i]=oGenerator.generateOperand();
              }
              if ((!operation.equals("+"))||(value[0]==-1) ||(value[1]==-1))
                                error=true;
              if (error==false)
                       return value;
              else
                       return new int[0];
  }
}
```

Code Listing 6: ProblemGenerator.java

```java
  public class StopWatch extends Thread
  {
     private int hour,minute,second;
     private String time;
     private int status;
     public StopWatch()
     {
              hour=0;
              minute=0;
              second=0;
              status=0;
     }

  // This method Starts the StopWatch
     public void run()
     {
              while(true)
              {
                       try{
                                Thread.sleep(1000);
                            }catch (Exception e) {System.out.println("Error");}
                       second++;
                       if (second==60)
                       {
                       minute++;
                       second=0;
                       }
                       if (minute==60)
                       {
                       hour++;
                       minute=0;
                       }
                       if (hour==13)
                       {
                       hour=0;
                       }
                       String time= ""+hour+":"+minute+":"+second;
     }
     }

  // This method resets StopWatch
  public void reset(int level)
  {
     int level=p.getLevel();
     if ((status==Constants.SUSPEND) &&((level==1)||(level==2)))
     {
             hour=0;
             minute=0;
             second=0;
             status=0;
      }
     else if ((hour==0)&&(minute==0)&&(second==0) &&(level==0))
      {
             hour=0;
             minute=0;
             second=0;
             status=0;
        }
     else if (((hour>0)||(minute>0)||(second>0))&&((level==1)||(level==2)))
      {
             hour=0;
             minute=0;
             second=0;
             status=0;
     }
     else
             minute=1;

  // This method pauses StopWatch
 public void pause()
 {
     status= Constants.SUSPEND;
     suspend();
  }

  // This method resumes StopWatch once it is paused
  public void resumeStopWatch()
  {
     status=0;
     resume();
  }
 }
```

CodeListing 7: StopWatch.java

```
  public class PerformanceRegulator
  {
     int level;
     DisplayManager dManager;
     Log log;
     public PerformanceRegulator()
     {
             dManager = new DisplayManager();
             log=new Log();
     }

     public void updatePerformance(int value[])
     {
             log.logProblem(value, level);

             dManager.display(value, level);
     }

     public int getLevel()
     {
             return level;
     }

     public void decreaseAptitude()
     {
             level--;
     }

    public void increaseAptitude()
     {
             Level++;
     }
    public void setLevel(int level)
     {
                     this.level = level;
     }
  }
```

CodeListing 8: PerformanceRegulator.java

```
import java.io.*;
public class Log
{
    int questionsAsked;
    BufferedWriter bw;
    public Log()
    {
            questionsAsked=0;
    }
    public Log(int questionsAsked)
    {
            this.questionsAsked=questionsAsked;
    }
    public void logProblem(int value[],int level)
    {
            if ((questionsAsked==0)&&(level==0) )
            {
                    questionsAsked=questionsAsked+1;
                    try{
                            bw=new BufferedWriter(new OutputStreamWriter(new
                            FileOutputStream("log.txt")));
                            bw.write(p.getProblem()+"\n");
                        }catch (Exception e){System.out.println("IO Exception");}
            }
            else if (questionsAsked>0)
            {
                    questionsAsked=questionsAsked+1;
                    try{
                            bw.write(p.getProblem()+"\n");
                        } catch (Exception e){System.out.println("IO Exception");}
            }
            else
                    questionsAsked=0;
    }
}
```

CodeListing 9: Log.java

Appendix D: Test Paths Generated by All-Path Coverage Criterion

```
1.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

2.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

3.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

4.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

5.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

6.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
```

```
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle
7.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@NotFull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

8.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@Notfull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

9.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@Notfull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

10.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@Notfull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

11.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@Notfull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

12.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
```

```
1.2.1:logProblem$Log@Notfull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

13.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

14.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

15.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

16.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

17.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

18.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
```

```
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

19.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

20.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

21.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

22.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

23.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

24.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
```

```
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

25.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@Notfull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

26.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@Notfull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

27.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@Notfull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

28.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@Notfull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

29.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@Notfull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

30.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
```

```
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@Notfull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

31.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

32.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

33.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

34.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

35.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

36.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
```

```
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

37.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

38.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

39.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

40.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

41.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

42.
1:newProblem$Coordinator@Coordinator
```

```
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

43.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@Notfull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

44.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@Notfull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

45.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@Notfull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

46.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@Notfull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

47.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@Notfull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

48.
```

```
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@Notfull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

49.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

50.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

51.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

52.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

53.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle
```

```
54.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Elementary
→Regulating|Counting|Elementary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

55.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

56.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

57.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

58.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

59.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle
```

```
60.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

61.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@Notfull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

62.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@Notfull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

63.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@Notfull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

64.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@Notfull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

65.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@Notfull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
```

```
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

66.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@NotFull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

67.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

68.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

69.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

70.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

71.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

72.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

73.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

74.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

75.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

76.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

77.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@Empty→[level=0]NotFull
```

```
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

78.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

79.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@NotFull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

80.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@NotFull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

80.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.11.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@NotFull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

82.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@NotFull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

83.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
```

```
1.2.1:logProblem$Log@NotFull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

84.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@NotFull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

85.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

86.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

87.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

88.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

89.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
```

```
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

90.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

91.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

92.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

93.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

94.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

95.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
```

```
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

96.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

97.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@NotFull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

98.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@NotFull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

99.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@NotFull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

100.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@NotFull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

101.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
```

```
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@NotFull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

102.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@NotFull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

103.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

104.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

105.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

106.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

107.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
```

```
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

108.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Secondary
→Regulating|Counting|Secondary
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

109.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

110.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

111.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

112.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

113.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
```

```
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

114.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

115.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
11.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@NotFull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

116.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@NotFull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

117.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@NotFull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

118.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@NotFull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

119.
1:newProblem$Coordinator@Coordinator
```

```
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@NotFull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

120.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@NotFull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

121.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

122.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

123.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

124.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

125.
```

```
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

126.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Novice→Novice
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

127.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

128.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

129.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

130.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle
```

```
131.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

132.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

133.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@NotFull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

134.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@NotFull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

135.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@NotFull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

136.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@NotFull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle
```

```
137.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@NotFull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

138.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@NotFull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

139.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

140.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

141.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

142.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
```

```
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

143.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

144.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Intermediate→Intermediate
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

145.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

146.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

147.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[hidden=false and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

148.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

149.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

150.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@Empty→[level=0]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

151.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
11.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@NotFull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

152.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@NotFull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

153.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@NotFull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

154.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@NotFull→[questionsAsked >0 and questionsAsked < 10]NotFull
```

```
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

155.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@NotFull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

156.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log NotFull→[questionsAsked >0 and questionsAsked < 10]NotFull
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

157.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

158.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

159.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|Idle→[not hidden and
displayingProblem and level=0]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle

160.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
```

```
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Running→[level=1 or level=2]Idle

161.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Paused→[level=1 or level=2]Idle

162.
1:newProblem$Coordinator@Coordinator
1.1:generate$ProblemGenerator@ProblemGenerator
1.1.1:*[i=1..2]generateOperand$OperandGenerator@OperandGenerator
1.1.1.1:getComplexity$ComplexityRegulator@Regulating|Counting|Advanced→
Regulating|Counting|Advanced
1.1.2:[not error]Problem$Problem@Problem
1.2:[not (value.size=0)]updatePerformance$PerformanceRegulator@Expert→Expert
1.2.1:logProblem$Log@NotFull→[questionsAsked = 10]Full
1.2.2:display$DisplayManager@Showing|ManagingSolutions→[not hidden and
displayingProblem]Showing|DisplayingProblem
1.2.2.1:reset$Stopwatch@Idle→[level=0]Idle
```