Comprehensively Evaluating Conformance Error Rates of Applying Aspect State Machines

Shaukat Ali
Certus Software V&V Center, Simula Research Laboratory
P.O. Box 134, 1325, Lysaker, Norway
Department of Informatics, University of Oslo, Oslo, Norway
shaukat@simula.no

Tao Yue
Certus Software V&V Center, Simula Research Laboratory
P.O. Box 134, 1325, Lysaker, Norway
tao@simula.no

Zafar I. Malik
Academy of Educational Planning and Management,
Ministry of Education, Islamabad, Pakistan
zafarimalik@acm.org

Abstract

Aspect Oriented Modeling (AOM) aims to provide enhanced separation of concerns during the design phase and proclaims many benefits (e.g., easier model evolution, reduced modeling effort, and reduced modeling errors) over traditional modeling paradigms such as object-oriented modeling. However, empirical evaluations of these benefits is severely lacking in the AOM community. In this paper, we empirically evaluate one of the AOM profiles: AspectSM, via a controlled experiment to assess if it can help in reducing modeling errors (referred as conformance errors in this paper), which is one of the benefits offered by AOM. AspectSM is a UML profile, which is developed to support automated state-based robustness testing. With AspectSM, crosscutting behaviors are modeled as aspect state machines using the stereotypes defined in AspectSM. We evaluate the conformance error rates of applying AspectSM from various perspectives by conducting four activities: 1) identifying modeling defects, 2) comprehending state machines, 3) modeling state machines, and 4) weaving aspect state machines into base state machines. For most of these activities, experimental results show that the error rates while performing these four activities using AspectSM are significantly lower than standard UML state machine modeling approaches.

Categories and Subject Descriptors D.2.2 [**Design Tools and Techniques**] State diagrams, Object-oriented design methods
G.3 [**Probability and Statistics**] Experimental Design

General Terms Measurement, Design, Reliability, and Experimentation

Keywords Aspect-oriented Modeling, Controlled Experiment, Modeling Errors, UML State machines

1. Introduction

Aspects are increasingly being used in various software development phases such as requirements, analysis, design, development and testing. Aspect-oriented paradigm aims to provide enhanced Separation of Concerns (SoC) and hence yield several potential benefits such as enhanced modularization, easier evolution, increased reusability, understandability, and applicability. Aspect-Oriented Modeling (AOM) is a research stream in this field and aims to support SoC during design modeling and it also claims similar benefits. However, there is very little empirical evidence of such benefits. Empirical studies are required to support these benefits about AOM and better understand its limitations.

While modeling behavior of industrial systems, such as using UML state machines, it is important to not only model nominal behavior but also robustness behavior which describes how the system should react to abnormal environmental conditions. It is needed to support, for example, model-based robustness testing of embedded or communication systems, which is the aim of the AspectSM profile [1]. It was developed to support model-based robustness testing of video conference system of Cisco, Norway [1]. However the approach is general enough to be used for other embedded or communication systems. Using AspectSM, crosscutting behavior can be modeled as aspect state machines, in order to facilitate the use of AOM for the purpose of SoC and therefore increase the readability, understandability, and applicability.

Motivated by this, we report a controlled experiment to comprehensively check whether AspectSM can help achieve lower error rates when performing the following four modeling activities: 1) identifying seeded defects, 2) comprehending state machines, 3) modeling crosscutting behaviors, and 4) weaving crosscutting behaviors (modeled as aspect state machines) to their corresponding base state machines. AspectSM is compared with standard UML state machines when crosscutting behavior is modeled using different modeling approaches. The controlled experiment was conducted with 25 fully trained, graduate students taking a graduate course in "Advanced Software Architecture" at the University Institute of Information Technology (UIIT) at the Pir Mehr Ali Shah Arid Agriculture University, Rawalpindi, Pakistan.

Results of the experiment showed that AspectSM has significantly lower error rates in terms of identifying modeling defects than the standard UML state machine modeling approach. Regarding comprehensibility, we observed that standard UML state machines with hierarchy achieved significantly lower error rates than aspect state machines. For the activity of modeling crosscutting behaviors, we found that aspect state machines have significantly lower error rates than standard UML state machines. Finally, we observed that for weaving aspect state machines, conformance error rate is on average less than 30%.

The rest of the paper is organized as follows: Section 2 describes the necessary background to understand the rest of the paper. Section 3 provides details on our experiment planning, and Section 4 reports on results and discussions. Section 5 presents possible threats to validity and Section 6 compares existing,

AOSD'12, March 25–30, 2012, Potsdam, Germany.

related experiments in Aspect-oriented Programming (AOP) to our experiment. Finally, we conclude our paper in Section 7.

2. Background

In this section, we provide a brief reminder of UML state machines and an overview of aspect state machines in AspectSM, the AOM mechanism being evaluated in our controlled experiment.

2.1 UML State Machines

UML state machines enable modeling the dynamic behavior of a class, subsystem, or system. State machines in general are extensively used to model a variety of systems such as communication [2] and control systems [3]. Due to the ability of state machines to capture rich and detailed information, they have been used for automatic code generation [4] and the automated generation of test cases [5-7]. UML state machines provide many advanced features such as concurrency and hierarchy, which aim at supporting large-scale modeling. Concurrency enables the modeling of concurrent behavior whereas state hierarchies capture commonalities among states. A submachine state in a state machine functions like a simple state, but is referring to another state machine. A submachine can be reused in more than one state machine and may refer to other submachines [8]. They can therefore help reduce the structural complexity of state machines. State machines developed using the hierarchical features of UML will be referred to as hierarchical state machines in this paper and the ones developed without using submachine states, with only basic features of UML state machines, will be referred as flat state machines.

2.2 Aspect State Machines

AspectSM is a UML profile described in [1], which supports the modeling of a system robustness behavior, which is very common type of crosscutting behavior in many types of systems. An example of a robustness behavior for a communication system is related to how the system should react, in various states, in the presence of high packet loss. The system should be able to recover lost packets and continue to behave normally in a degraded mode. In the worst case, the system should go back to the most recent state and not simply crash or show inappropriate behavior. In a control system, one needs to model, for example, how the system should react, in various states, when a sensor breaks down. Though AspectSM was originally defined to support scalable, model-based, robustness testing, including test case and oracle generation, a fundamental question is whether it is easier to model crosscutting concerns such as robustness with AOM in general, and AspectSM in particular, than simply relying on UML state machines to do it all. In AspectSM, the core functionality of a system is modeled as one or more standard UML state machines (called base state machines). Crosscutting behavior of the system (e.g., robustness behavior) is modeled as aspect state machines using the AspectSM profile. State machines developed using this profile will be referred as aspect state machines. A weaver [1] then automatically weaves aspect state machines into base state machine to obtain a complete model, that can for example be used for testing purposes. The AspectSM profile specifies stereotypes for all features of AOM, in which the concepts of Aspect, Joinpoint, Pointcut, Advice, and Introduction [9] are the most important ones. Below, we briefly describe these concepts along with how they are represented in the profile. Figure 1 shows the metamodel representing and relating these concepts. The complete discussion of the AspectSM profile can be found in [1]. We can see from that description that proper modeling requires the modeler to master AOM concepts and mentally determine the end result of weaving; an exercise that cannot be taken for granted and be a priori considered easier than directly modeling crosscutting concerns in a state machine. Investigating the benefits of AspectSM, and more generally AOM, is the main purpose of our experiment.

2.2.1 Main Concepts in AspectSM

Aspect This concept describes a crosscutting concern, e.g., the robustness behavior of a system in the presence of failures in its environment (e.g., network failures in communication systems). Using the AspectSM profile, we model each aspect as a UML 2.0 state machine augmented with stereotypes and attributes.

Joinpoint A Joinpoint is a model element selected by a *Pointcut* (defined next) where an *Advice* or *Introduction* (additional behavior) can be applied [9]. In the context of UML, all modeling elements in UML can be possibly joinpoints. In UML state machines, joinpoints can be, for example, *State*, *Activity*, *Constraint*, *Transition*, *Behavior*, *Trigger*, and *Event*.

Pointcut A *Pointcut* selects one or more joinpoints, where *Advice* or *Introduction* can be applied. A *Pointcut* can have at most one *Before* advice, one *Around* advice and one *After* advice. In the AspectSM profile, all pointcuts are expressed with the Object Constraint Language (OCL) [8] on the UML 2.0 metamodel [8]. We decided to use the OCL to query joinpoints because the OCL is the standard way to write constraints and queries on UML models and can therefore be used to query jointpoints in UML state machines. Also, several OCL evaluators are currently available that can be used to evaluate OCL expressions such as the IBM OCL evaluator [10], OCLE 2.0 [11], and EyeOCL [12]. Furthermore, writing pointcuts as OCL expressions does not require the modeler to learn a notation that is not part of the UML

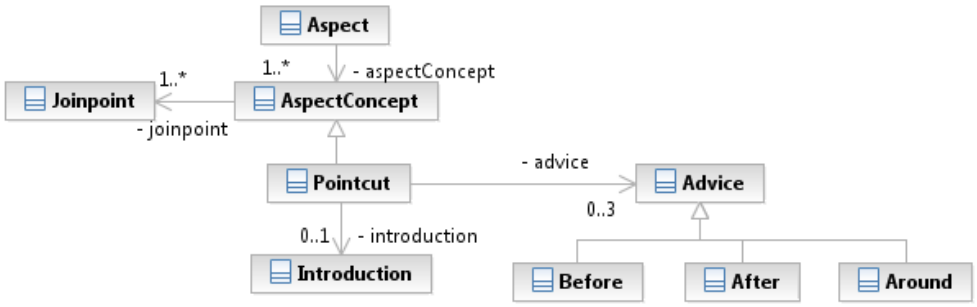

Figure 1. Conceptual domain model of the AspectSM profile

standard. In the literature, several alternatives are proposed to write pointcuts [13-17] but all of them either rely on languages (mostly based on wildcard characters to select joinpoints, for instance, '*' to select all joinpoints) or diagrammatic notations which are not standard, thus forcing modelers to learn and apply new notations or languages. Using the OCL, we can write precise pointcuts to select jointpoints with similar properties. We do so by selecting modeling elements (jointpoints) based on the properties of UML metaclasses. This further gives us the flexibility to specify precise pointcuts as any condition defined based on some or all of the properties of a UML metaclass, e.g., a pointcut on the *Transition* metaclass, selecting a subset of transitions in a base state machine, which have triggers of type *CallEvent* and do not have any guard.

Advice An *Advice* is one of the crosscutting behaviors of the Aspect. The *Advice* is attached to Joinpoint(s) selected by the *Pointcut*. In correspondence to AspectJ [18] concepts, an *Advice* can be of type *Before*, *After*, or *Around*. A *Before* advice is applied before Joinpoint(s), an *After* advice is applied after Joinpoint(s), whereas an *Around* advice replaces Joinpoint(s). For example, introducing guards on a set of transitions of a state machine is an example of a *Before* advice on transitions (*Joinpoint*).

Introduction An *Introduction* is similar to the inter-type declaration concept in AspectJ [18]. Using *Introduction* in our context, new modeling elements (e.g., state or transition) can be introduced into a UML state machine.

2.2.2 Example of applying AspectSM

In this section, we present an example of the application of AspectSM. An aspect state machine modeling crosscutting behavior *EmergencyStop* is shown in Figure 2. This UML state machine is stereotyped as <<Aspect>>, which means that it is an aspect state machine. The <<Aspect>> stereotype has two attributes: *name* and *baseStateMachine*, whose values are shown in the note labeled as '1' in Figure 2. The name attribute contains the name of the aspect (*EmergencyStop* in this example), whereas the *baseStateMachine* attribute holds the name of the base state machine, on which this aspect will be woven, which is *ElevatorControl* provided in [19] in this example.

The aspect state machine consists of two states: *SelectedStates* and *ElevatorStopped*. *SelectedStates* is stereotyped as <<Pointcut>>, which means that this state selects states from the base state machine. There are two attributes of <<Pointcut>>, whose values are shown in the note labeled as '2' in Figure 2. The name attribute indicates the name of the pointcut and type denotes the type of the pointcut, which is *All* in this case meaning that we are selecting all states of the base state machine. In AspectSM, different types of pointcuts can defined, a complete list of other types of pointcuts is presented in [1]. All the model elements stereotyped as <<Introduction>> (one state, two transitions) will be newly introduced elements in the base state machine during weaving. This aspect introduces the *ElevatorStopped* state in the base state machine, and selects all states of the base state machines and introduces transitions from them to *ElevatorStopped* with trigger *EmergencyStopButtonPressed*. In addition this aspect introduces transitions from *ElevatorStopped* to all the states selected by *SelectedStates* with trigger *EmergencyStopButtonReleased*.

3. Experiment Planning

This section discusses planning of the experiment based on the experiment reporting template defined by Wohlin et al. [20].

3.1 Goal, Research Questions and Hypotheses

The objective of our experiment is to assess the AspectSM profile with respect to the modeling errors made by the subjects while performing different modeling activities. Modeling errors are assessed from four perspectives corresponding to four activities of applying AspectSM: 1) errors made while inspecting state machines to identify seeded defects, 2) errors made while comprehending state machines via answering a comprehension questionnaire, 3) errors made while designing state machines, and 4) errors made while weaving an aspect state machine into its base state machine.

Based on the above-mentioned objectives of our experiment, we defined the following research questions:

RQ1: Does the use of AspectSM reduce errors while inspecting aspect state machines (against their specifications) to identify defects, when compared to hierarchical and flat state machines?

We compare aspect state machines with two different types of state machines capturing crosscutting behavior: hierarchical and flat state machines for this activity. None of the expected differences between them can a priori be certain to be in a specific direction. This therefore leads to the definition of two-tailed hypotheses.

H^1_0: Error rate in inspecting aspect state machines to detect defects is the same as hierarchical state machines.

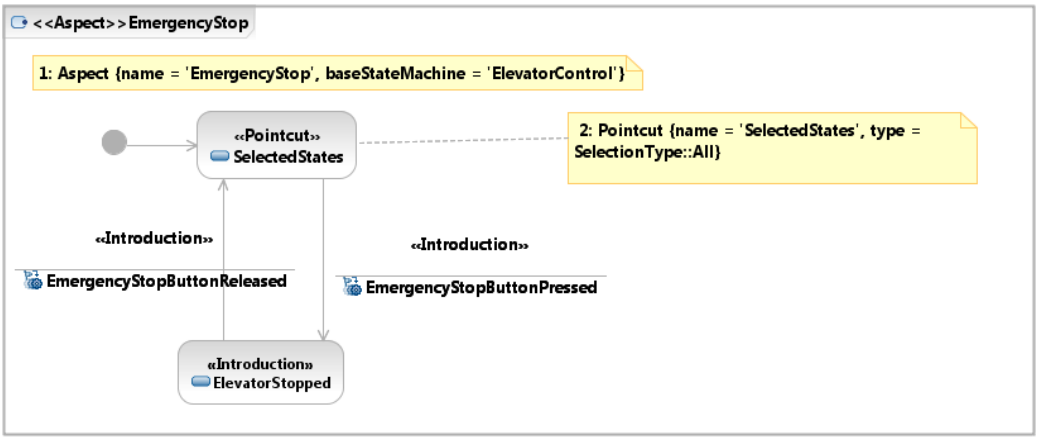

Figure 2. An aspect state machine for crosscutting behavior EmergencyStop

H^2_0: Error rate in inspecting aspect state machines to detect defects is the same as flat state machines.

RQ2: Does the use of AspectSM reduce errors in comprehending aspect state machines when compared to hierarchical and flat state machines?

Similar to the previous research question, we compare the comprehensibility of AspectSM with the two different types of state machines capturing crosscutting behavior (hierarchical and flat state machines) based on the scores to answer a comprehension questionnaire. We defined the following two-tailed null hypotheses accordingly.

H^3_0: The error rate in comprehending aspect state machines is the same as hierarchical state machines.

H^4_0: The error rate in comprehending aspect state machines is the same as flat state machines.

RQ 3: Does the use of AspectSM reduce conformance errors of designing aspect state machines with respect to reference state machines, when compared to flat state machines and/or hierarchical state machines?

We evaluate AspectSM from the perspective of how well it can support the activity of modeling crosscutting behaviors by measuring conformance error rates of aspect state machines against their reference state machines. We further calculate conformance error rates of using flat state machines and/or hierarchical state machines to model the same crosscutting behaviors against their respective reference state machines. Finally, we compare the conformance error rates of aspect state machines with the conformance error rates of the flat and/or hierarchical state machines for the same crosscutting behaviors. Since the differences in results can be in either direction, we defined a two tailed null hypothesis as follows:

H^5_0: Conformance error of aspect state machines is the same as flat/hierarchical state machines.

RQ 4: To which extent do the woven state machines derived by subjects conform to the reference state machines automatically produced by our weaver?

This question calculates the errors made by the subjects to weave aspect state machines into their corresponding base state machine for the purpose of evaluating how well the subjects can understand the aspect state machines through this weaving activity. In our previous work [1], we developed a weaver for AspectSM, which automatically weaves aspect state machines into base state machines and produces a woven state machine. To answer this research question, we compared woven state machines developed by a subject with the woven state machines automatically produced by our weaver.

3.2 Experiment Subjects

Our controlled experiment was conducted at the Pir Mehr Ali Shah Arid Agriculture University, Rawalpindi, Pakistan. The subjects in the experiment are 25 graduate students taking a graduate course in 'Advanced Software Architecture' at the University Institute of Information Technology (UIIT). Our motivation was to find a group of subjects with adequate background that could be trained to use our AOM approach over a short period of time. Most practitioners have very little knowledge of AOP and even less of AOM. Ensuring they have the required background is also difficult. This is why we relied on a group of mature graduate students.

The course is offered in the Master of Science program. The students in this degree already hold a Bachelor in Computer Science or Information Technology and have already been exposed to the UML notation and extensions in the form of UML profiles. On average, each student went through five development and two modeling courses. Eighteen students (out of twenty-five) have used the UML notation for their final year projects before the experiment was conducted. Twenty students gained experience in development work in IT companies or as teaching staff in computer science.

The subjects were free to choose to participate or not into the experiment and were told their choice would have no effect on their course grades.

3.3 Material

In this section, we provide the material we used to conduct the experiment.

3.3.1 Case Study Systems

Two case study systems were used for the experiment: elevator control system (ECS) and automated teller machine (ATM). Detailed description of ECS/ATM and the crosscutting behaviors used in the experiment is presented in [21]. The complexity of the state machines in terms of numbers of model elements of the two case study systems is shown in Table 1. Below, we provide a brief but necessary discussion of these two case study systems.

Elevator Control System The first system used for the experiment is an elevator control system (ECS) that controls movements of an elevator in a building. For our experiment, we extended the specification of the elevator given in [19] with two additional crosscutting behaviors so that the AspectSM profile could be used to model them. These two crosscutting behaviors are:

Emergency call behavior (Call) The behavior of an elevator, when an emergency call is made.

Table 1. Complexity of the state machines

Case Study	Crosscutting Behavior	ASM				HSM			FSM		
		#S	#T	#P	Total	#S	#T	Total	#S	#T	Total
ECS	Emergency Call	16	18	1	35	17	21	38	15	27	42
	Emergency Stop	14	17	1	32	14	17	31	13	23	36
ATM	Cancel Transaction	7	15	2	24	-	-	-	6	15	21
	Network Failure	8	16	2	26	-	-	-	7	22	29
	Power Failure	8	15	2	25	-	-	-	7	21	28
ATM	Cancel Transaction	-	-	-	-	-	-	-	6	15	21
	Network Failure	-	-	-	-	-	-	-	7	22	29
	Power Failure	-	-	-	-	-	-	-	7	21	28

ASM = Aspect State Machine, HSM = Hierarchical State Machine, FSM = Flat State Machine
#S = Number of states, #T = Number of transitions, #P = Number of pointcuts

Emergency stop behavior (Stop) the behavior of an elevator, when the emergency stop button is pressed.

Automated Teller Machine The second system used for the experiment is the popular automated teller machine (ATM) system from [19].

"A bank has several (ATMs), which are geographically distributed and connected via a wide area network to a central server. Each ATM machine has a card reader, a cash dispenser, a keyboard/display, and a receipt printer. By using the ATM machine, a customer can withdraw cash from either a checking or a saving account, query the balance of an account, or transfer funds from one account to another. A transaction is initiated when a customer inserts an ATM card into the card reader. Encoded on the magnetic strip on the back of the card is recognized, the system validates the ATM card to determine that the expiration date has not passed, that the user-entered PIN matches the PIN maintained by the system, and that the card is not lost or stolen. The customer is allowed three attempts to enter the correct PIN; the card is confiscated if the third attempt fails."

The following crosscutting behaviors were defined for ATM.

Cancel Transaction A customer may cancel a transaction at any time except when the ATM is closed down or not idle. Whenever a cancel request is made, the transaction is terminated and then the card is ejected.

Network Failure An important robustness behavior of ATM is the behavior in the presence of network failure. Whenever network connection fails during the operation of ATM except when it is closed down, it saves the current transaction in its local memory and then tries to recover the network connection. If the network connection is established, the saved transaction is loaded and ATM continues the transaction. Otherwise, it will simply be closed down.

Power Failure Another robustness behavior of ATM is that in the case of power failure it starts its Uninterruptable Power Supply (UPS) and continues the operation.

3.3.2 Design Defect Classification

To answer RQ1 (Section 3.1), the experiment subjects were asked to identify defects seeded in state machines through checking the conformance of the state machines against their corresponding specifications (Section 3.4).

To systematically inspect state machines for various types of defects, a classification of different types of design defects is required. The classification we used in the experiment is given below and was adapted from Binder's book [6]. It was provided to the subjects of the experiment as part of the answer sheet (Section 3.3.5) to systematically collect their answers.

Incorrect Transition (IT) A transition that comes from or leads to a wrong state or the transition has wrong guard, trigger, and/or events.

Missing Transition (MT) According to the specification of a state machine, there is a transition missing.

Extra Transition (ET) A transition is subsumed by another transition in a state machine. Such a transition is redundant in the sense that removing it still keeps the state machine in conformance to its specification.

Missing State (MS) A state that should be modeled in a state machine according to its specification but is missing.

Incorrect State (IS) A state is incorrect if it has a wrong state invariant, do, entry and/or exit activities.

Extra State (ES) A state is subsumed by another state. This state is considered as an extra state in the sense that removing it still keeps the state machine in conformance to its specification.

3.3.3 Seeded Defects

It is important to note that to answer RQ1, we were interested in studying errors made by the subjects while inspecting crosscutting behaviors to detect defects; therefore we seeded defects only in the crosscutting behaviors. Different types of defects were selected after we carefully examined the base and aspect state machines and identified possible independent defects. This resulted in three types of defects for the first crosscutting behavior (Call): missing transition (MT), incorrect transition (IT), and incorrect state (IS), and a missing transition (MT) for the second crosscutting behavior (Stop). Table 2 shows the distribution of these defects that were seeded in the compared state machines.

Because aspects model crosscutting behavior, it is expected that one defect in an aspect often corresponds to several defects in the corresponding hierarchical state machine. Similarly, because hierarchical states factor out common behavior, one defect in a hierarchical state machine often leads to several defects in its corresponding flat state machine. As a result, different numbers of defects were seeded in the three state machines in order to conceptually correspond to equivalent defects.

Table 2. Distribution of seeded defects in state machines

Approach	Call			Stop
	MT	IT	IS	MT
Aspect	1	1	1	1
Hierarchical	4	2	1	1
Flat	11	9	1	10

3.3.4 Comprehension Questionnaire

For RQ2, we want to evaluate how error prone it is to comprehend various types of state machines. To this effect, a comprehension questionnaire was designed to evaluate, in a repeatable and objective way, the extent to which a subject can understand the state machines. For example, questions concern what scenario is triggered when an event happens in a certain state. The subjects were asked the same ten questions on two crosscutting behaviors together for all three types of state machines. The subjects had to answer each question by inspecting the state machines assigned to them and error scores were computed by accounting for partially correct answers. For example, if the answer to a question entailed to list four transitions, then wrongly pointing out each transition contributed 0.25 to the full mark of the question.

3.3.5 Answer Sheets

There were four answer sheets developed for the experiment. The first answer sheet (for RQ1) was developed to collect information about classes of defects that were identified by each subject, the number of defects in each class, and the location of identified defects. A table was provided to subjects for each crosscutting behavior. The rows of the table were labeled with each defect class, whereas the columns featured two pieces of information about defects: number of defects identified in each class and location of each identified defect. The second answer sheet (for RQ2) was designed to collect answers to the comprehension questionnaire. The third answer sheet (for RQ3) was developed to collect answers for two groups modeling crosscutting behaviors:

Table 3. Design of the experiment

Round	Task	Task Type	Case Study	Crosscutting behavior	Group 1		Group 2		Group 3	
					Approach	Data points	Approach	Data points	Approach	Data points
1	R1T1	Identify Defects	ECS	Emergency call	ASM	8	HSM	8	FSM	9
				Emergency stop	ASM	8	HSM	8	FSM	9
	R1T2	Comprehend state machines		Emergency call and stop	FSM	9	ASM	8	HSM	8
2	R2T1	Model crosscutting behaviors	ATM	Cancel transaction	ASM	10	FSM	10	-	-
				Network failure	ASM	10	FSM	10	-	-
				Power failure	ASM	10	FSM	10	-	-
3	R3T1	Weave crosscutting behaviors	ATM	Cancel transaction	-	15	-	-	-	-
				Network failure	-	15	-	-	-	-
				Power failure	-	15	-	-	-	-

ASM: Aspect state machine, FSM: Flat state machine, and HSM: Hierarchical state machine
R1T1: Round 1 Task 1, R1T2: Round 1 Task 2, R2T1: Round 2 Task 1, and R3T1: Round 3 Task 1

one for the group using standard UML state machines to directly model crosscutting behaviors on the base state machine and the second for the group modeling crosscutting behaviors using aspect state machines. The answer sheet was designed so that subjects can provide their solution one after another and provide time required to model each crosscutting behavior. The fourth answer sheet (for RQ4) was developed to collect woven state machines from subjects. The answer sheet contained a base state machine and three aspect state machines corresponding to the three crosscutting behaviors (Section 3.3) of the ATM system. The answer sheet was designed such that the subjects can provide their solutions one after another and provide time they spent to weave each aspect state machine into the base state machine.

3.4 Design

The design of our experiment is summarized in Table 3. Our experiment design consists of three rounds. For the first round, we divided the subjects into three groups: Group1, Group2, and Group 3. Given the number of subjects, this led respectively to 8, 8, and 9 subjects in each group. In the first round, there were two tasks. In each task, each group was given a different type of state machines (Aspect, Hierarchical, or Flat). During the training sessions (Section 3.6), each subject was equally trained to understand all three different types of state machines. The subjects were also given a modeling assignment, after the training sessions, for them to practice before the actual experiment tasks. This assignment was marked by the first author of this paper and grades were used to form blocks (i.e., groups of students of equivalent skills). The experiment groups were then formed through randomization and blocking to obtain three comparable groups with similar proportions of students from each skill block.

As shown in Table 3, for R1T1 (Round 1 Task 2), each group was asked to sequentially identify defects in the state machines modeling two crosscutting behaviors: Emergency call and Emergency stop. Group 1 was given state machines modeled using AspectSM. The subjects in Group 1 were given one base state machine and one aspect state machine (ASM) modeling Emergency Call and one ASM for the Emergency Stop crosscutting behavior. Group 2 was given one hierarchical state machine (HSM) for Emergency Call and one HSM for Emergency Stop. Similarly, Group 3 was given one flat state machine (FSM) for Emergency Call and one FSM for Emergency Stop. Seeded defects for each type of state machines (Aspect, Hierarchical, and Flat) are presented in Table 2. For R1T1, we used a between-subjects design, which offers several advantages. First, a between-subjects design counterbalances learning effects since each subject is exposed to just one type of treatment (e.g., the Aspect approach) and one task (e.g., one crosscutting behavior) and thus the performance of subjects is not influenced with the experience gained while working with other treatments with the same crosscutting behavior. Second, between-subjects design counterbalances fatigued effect, which is common while working with several treatments for many tasks.

In R1T2, the three groups divided for R1T1 were rotated for pedagogical reasons so that each group can experience difference types of state machines. For example, Group 1 in R1T2 was given flat state machines instead of the aspect state machines which is the case in R1T1.

In R2T1, the subjects were divided into two groups: one group (the FSM group) was asked to use flat state machines to design crosscutting behavior and the other (the ASM group) was asked to apply ASM. Both groups were asked to model the same set of crosscutting behaviors, based on the same base state machine specifying the core behavior of the ATM system. The ASM group was asked to model crosscutting behaviors as aspect state machines, whereas the FSM group was asked to model crosscutting behaviors directly on the base state machine. Notice that in this round, 20 subjects participated in the experiment; therefore we have 10 subjects per group. Again in this round, we used a between-subjects design because of the same reasons as we discussed previously.

In R3T1, there was just one group of subjects and they were asked to sequentially perform three tasks by weaving one aspect state machine at a time and provide the time they spent to each weaving activity. The three tasks were to weave the following three aspect state machines: Cancel transaction, Network failure, and Power failure. A brief description of these aspect state machines is provided in Section 3.3.1. In this round, 15 subjects participated in the experiment.

Note that in Table 3 for different rounds, different number of subjects participated. This is due to the reason that each round was conducted on a separate day and some subjects couldn't participate, for instance, due to practical reasons such as clashes with courses and exams.

3.5 Dependent Variables

Error Rate in Identifying Defects (ERID) This variable calculates the percentage of seeded defects not identified by a subject and is calculated as:

Number of unidentified defects by a subject / Total number of seeded defects

Error rate of answering comprehension questionnaire (ERC) This calculates the error rate of answering the comprehension questionnaire and is calculated as:

Sum of the scores of the wrong answers of the questions / Total number of questions (i.e. 10)

In the formula above, the score for each answer was calculated based on the marking procedure discussed in Section 3.3.4 and 10 is the total number of the questions in the questionnaire.

Conformance error of state machine (CERSM) This variable measures the conformance error of a subject's state machine by comparing it with a reference state machine. It is determined by the conformance error of states and transitions-two main modeling elements of a state machine. Note that, since we have two sets of results with respect to two different treatments (for R2T1): flat state machines and aspect state machines, two sets of measures were designed to evaluate the completeness of two different types of state machines derived by the subjects given different treatments.

The formula for *CERT* (i.e., conformance error in transitions) shown in Table 4, calculates the conformance error of a subject's state machine by looking at the transitions of the subject's solution with the reference solution (this holds for each group) that do not match. Matching of transitions is determined by looking at whether the source and target states of a transition match to the source and target states of any transition in the reference model. Three model elements constituting a transition (i.e., guard, trigger, and effect) are further assessed to evaluate the conformance error of a transition. Matching of the trigger, guard, and effect of a transition is determined whether their names are the same or similar to the corresponding elements of the matched transition in the reference state machine. The conformance error of a transition is calculated based on the proportion of the trigger, guard, and effect of the transition that do not match with the reference solution. For instance, if only the guard and trigger of a transition do not match with a transition in the reference solution, then this means that the transition conformance error is 66% (2/3) to the reference transition. In other words, 33% (1/3) of modeling elements of the transition are missing. For flat state machines, we compare only the guard, trigger, and effect of a transition; while for aspect state machines, in addition we also check whether required stereotypes are applied to transitions. This is so because AspectSM requires applying stereotypes on states and transitions in aspect state machines (Section 2.2). For each transition *k* in a subject's solution, we check if its guard, trigger, or effect is missing with respect to the matched transition in the reference solution (Table 5). For each missing guard, trigger, and effect, we assign value 1 to the corresponding variable (E_{guard_k}, $E_{trigger\_k}$ or E_{effect_k}), otherwise 0.

Similarly, we calculate *CERS* as shown in Table 4, whereas Table 5 shows the measures needed for the calculation. Since state and transition are two main types of model elements of a UML state machine, the overall conformance error (*CERSM*) of the state machine is therefore calculated based on the conformance error of states and transitions, as shown in Table 4. A simpler way to do so is to just simply take average of *CERT* and *CERS*. However, the numbers of states and transitions in a

Table 4. Conformance error rate measures for a state machine diagram

Category	Measure	Formula	Formula
CERSM	CERS	E_S	$\frac{(N_{t\_r} * CERT + N_{s\_r} * CERS)}{N_{t\_r} + N_{s\_r}}$
	CERT	E_T	

N_{t_r} is the total number of the transitions in the reference model. N_{s_r} is the total number of the states in the reference model.

Table 5. Collected state machine diagram data

Measure	Specification
E_{Sname_k}	Missing name of the k_{th} state in a subject's state machine diagram
$E_{Sstereotype\_k}$	Missing stereotype of the k_{th} state in a subject 's aspect state machine diagram
E_{guard_k}	Missing guard of the k_{th} transition in a subject 's state machine diagram
$E_{trigger\_k}$	Missing trigger of the k_{th} transition in a subject 's state machine diagram
E_{effect_k}	Missing effect of the k_{th} transition in a subject's state machine diagram
$E_{Tstereotype\_k}$	Missing stereotype of the k_{th} transition in a subject's aspect state machine diagram
E_S	For a standard state machine: $\frac{\sum_{k=1}^{n}(E_{Sname\_k})}{n}$
	For an aspect state machine: $\frac{\sum_{k=1}^{n}(E_{Sname\_k}+E_{Sstereotype\_k})/2}{n}$
E_T	For a standard state machine: $\frac{\sum_{k=1}^{n}(E_{Tguard\_k}+E_{Ttrigger\_k}+E_{Teffect\_k})/3}{n}$
	For an aspect state machine: $\frac{\sum_{k=1}^{n}(E_{Tguard\_k}+E_{Ttrigger\_k}+E_{Teffect\_k}+E_{Tstereotype\_k})/4}{n}$

E_{Sname_k}, $E_{Sstereotype\_k}$, E_{guard_k}, $E_{trigger\_k}$, E_{effect_k}, and $E_{Tstereotype\_k}$ are Boolean measures that take value 0 and 1 only.
'n' refers to the number of matched states or transitions.

Table 6. Statistical tests for measures

Round	Task	Measure	Approach	Mean difference	p-value (α = 0.05)
1	R1T1	ERID	ASM vs HSM	-0.56	**0.0011**
			ASM vs FSM	-0.62	**0.0001**
			HSM vs FSM	-0.06	0.8590
	R1T2	ERC	ASM vs HSM	2.17	**0.0419**
			ASM vs FSM	-1.89	0.0821
			HSM vs FSM	-4.06	**0.0038**
2	R2T1	CERT	ASM vs FSM	-0.05	**0.0237**
		CERS	ASM vs FSM	0.07	**0.0127**
		CERSM	ASM vs FSM	-0.14	**0.0368**

solution might be different, so taking average of them is unfair. So, to be fair in the calculation and considering each modeling element (state or transition) having same weight, we calculate the overall conformance error based on the proportions of states and transitions in a state machines. To achieve this, first we obtain the overall conformance error of transitions by multiplying *CERT* with the total number of the transitions in the reference model (N_{t_r}). Similarly, we calculate overall completeness of states by multiplying CERS with the total number of the states N_{s_r}. Finally, we take sum of both and divide it with the sum of the numbers of states and transitions in the reference state machine.

3.6 Procedure (Training)

The subjects were trained by the first author of this paper. Two three-hour sessions were given on the following topics: 1) Recap of UML state machines since subjects were already familiar with this topic preceding the training (Section 3.2), 2) Introduction to the Object Constraint Language (OCL), 3) Introduction to aspect-oriented software development (AOSD), and 4) Aspect-oriented modeling (AOM) using the AspectSM profile. Each topic was accompanied with several examples and interactive class assignments. As previously discussed, the subjects were given a home assignments after the training sessions to practice the three state machine modeling approaches and groups were later formed based on the grades of this assignment.

4. Results and Discussion

We analyze and present our experiment results in this section. Descriptive statistics and statistical tests are presented in Section 4.1 and Section 4.2, respectively. The discussion of results is provided in Section 4.3.

4.1 Descriptive Statistics

We report descriptive statistics for each dependent variable designed to answer each research question in Table 7. Descriptive statistics for each variable is characterized using their minimum, maximum, median, mean and standard deviation.

Recall from Section 3.5 that *ERID* aims to measure the percentage of defects not identified by the subjects in R1T1. We see that on average *ERID* for ASM is 19%, which is much lower than HSM and FSM, which are 74% and 81%, respectively. Regarding *ERC* (the variable that measures the error rate of answering the comprehensive questionnaire in R1T2), mean *ERC* for HSM is 1.44, which is lower than both ASM and FSM, which are 3.61 and 5.5, respectively.

To answer RQ2 and RQ3, variables *CERT*, *CERS*, and *CERSM* were designed to measure the conformance errors for ASM and FSM. As shown in Table 7, for R2T1, mean *CERT* for FSM (37%) is higher than ASM (23%). Mean *CERS* is 13% and 19% for FSM and ASM, respectively. Total conformance error rates (*CERSM*) for FSM and ASM are 36% and 22%, respectively. For R3T1, *CERT* for FSM is 31% and *CERS* is very low (i.e., 5%). Total mean conformance error (*CERSM*) is 29%.

4.2 Statistical Tests

In this section, we check whether the differences observed in the previous section are statistically significant to determine if we can reject the null hypotheses stated in Section 3.1. For all the statistical tests reported in this section, we used a significance level of α=0.05.

To check if, overall, there exist significant differences among the three approaches under investigation for round 1, we performed the Kruskal–Wallis one-way analysis of variance [22] on ERID and ERC. We performed this test since our samples are not normal as we checked with the Shapiro–Wilk W test [22]. We

Table 7. Descriptive statistics for measures

Round	Task	Measure	Approach	Min	Median	Max	Mean	Std
1	R1T1	ERID	ASM	0	0	1	0.19	0.32
			HSM	0	0.83	1	0.74	0.34
			FSM	0.1	0.9	1	0.81	0.25
	R1T2	ERC	ASM	1	3.5	6.25	3.61	2.08
			HSM	0	1.5	3.5	1.44	1.24
			FSM	2.25	6.13	8.5	5.5	2.45
2	R2T1	CERT	FSM	0	0.33	1	0.37	0.28
			ASM	0	0.13	1	0.23	0.30
		CERS	FSM	0	0	1	0.13	0.34
			ASM	0	0.25	0.5	0.19	0.18
		CERSM	FSM	0	0.33	0.89	0.36	0.24
			ASM	0	0.15	0.8	0.22	0.21
3	R3T1	CERT	FSM	0	0.34	0.84	0.31	0.28
		CERS	FSM	0	0	0.5	0.05	0.15
		CERSM	FSM	0	0.32	0.78	0.29	0.26

obtained p-values of 0.0001 and 0.0047 for ERID and ERC respectively, which are less than 0.05 hence are significant (Table 8). This encouraged us to perform a pairwise comparison of the distributions obtained for the three state machines using Mann–Whitney U for ERID and ERC. It doesn't assume normally distributed samples as it is our case as the results of the Shapiro–Wilk W test [22] showed. Pairwise p-values and mean differences across pairs for each measure are reported in Table 6. Bold p-values highlight statistically significant results. The mean differences between pairs of approaches indicate the direction in which the result is significant. For instance, in row 1, for measure ERID, between ASM vs HSM, the mean difference is negative and p-value is less than 0.05 (our selected significance level). This means that ERID for ASM is significantly lower than HSM.

For *CERT*, *CERS*, and *CERSM*, since we have two treatments, we performed the Wilcoxon signed-rank test [22], which is non-parametric equivalent of t-test. We performed this test because our samples are not normally distributed as we checked with the Shapiro–Wilk W test [22]. Again, bold values highlights that the results are significant and mean differences indicate the direction in which the results are significant.

4.3 Discussion

The above results showed that (Section 4.1 and Section 4.2), aspect state machines have significantly lower error rates than flat and hierarchical ones in terms of ERID given our selected α (0.05) and sample size (Table 3). This indicates that for the activity of identifying defects, the subjects given ASM made fewer errors than the subjects inspecting FSM and HSM. This can be explained from the fact that ASM is much simpler than HSM and FSM in terms of number of modeling elements (Table 1) and therefore the subjects with ASM had high possibility to identify defects. We did not observe significant difference between HSM and FSM in terms of ERID.

In terms of ERC, we observed that HSM achieved statistically lower error rate than ASM and FSM. The mean error rate of ASM is lower than FSM, though no significant differences were observed. Recall that, answering the comprehensive questionnaire requires the subjects to comprehensively comprehend the given state machines. For Aspect state machines, the subjects were required to understand for example Pointcut specifications (written in OCL) in order to correctly answer the comprehensive questions. A plausible explanation is that due to insufficient training given to the subjects on understanding OCL expressions (as part of the training given to AspectSM (Section 3.6)), the solutions of the subjects with ASM exhibit more errors as compared with HSM.

Recall that for the activity of modeling crosscutting behaviors using state machines (R2T1) we have only two treatments: ASM and FSM. For CERT (transitions), we found that ASM has significantly lower error rate than FSM, whereas for CERS (states), FSM has significantly lower error rate than ASM. The reason why significant differences were observed for transitions (CERT) (in favor of ASM) instead of states is that (as shown in Table 1) the aspect state machines contain comparable number of states as the flat state machines modeling the same set of crosscutting behaviors. Aspect state machines additionally require applying stereotypes, which is not the case for flat state machines. Therefore, more modeling effort is required to apply AspectSM and actually for this specific set of crosscutting behaviors, more states were required to be modeled in aspect state machines than flat state machines. Hence in such case, aspect state machines are more error-prone. Regarding CERT (transitions), ASM achieved significantly lower error rate than FSM. This is because AspectSM helped to reduce the number of transitions in aspect state machines (Table 1) for the three crosscutting behaviors. It is worth noticing that with more complicated crosscutting behaviors, one can expect that AspectSM will reduce the number of states in aspect state machines and therefore lower error rate for states in aspect state machine can be expected. Total conformance error rate (CERSM) is significantly lower in ASM than FSM (Table 6). Again this is due to the fact that ASM are much simpler that FSM in terms of total number of modeling elements (states and transitions).

In round 3, recall that we didn't have multiple treatments as the subjects were required to weave aspect state machines into a base state machine. We observed that on average CERT (transitions) is 31%, CERS (states) is 5%, whereas CERSM is 29%, as shown in Table 7. Notice that CERT is much higher than CERS. This is due to the fact that the subjects had to weave more transitions in the base state machine than states (Table 1). We assume that more training will further reduce these percentages of weaving errors. However, it is important to note that the errors made by the subjects may be for example, accidentally missing modeling a transition and not due to the actual understandability of aspect state machines. This might happen during the weaving process when many model elements, especially transitions, have to be added to the base state machine. It is very important to notice that the subjects were asked to weave state machines, which is an indirect way to study how the subjects understand aspect state machines; weaving aspect state machines to their base state machines requires the subjects to understand both state machines and how their composition takes place. Of course, the weaving process can be automated but the objective of this task is to test the understandability of AspectSM. Using our automated weaver [1], a correct woven state machine can be automatically derived from aspect state machines and their base state machine.

5. Threats to Validity

Below, we discuss the threats to validity of our controlled experiment based on template in [20].

As with most controlled experiments in software engineering, our main conclusion validity threat is related to the sample size on which we base our analysis. We designed and conducted the experiment to maximize the number of observations within time constraints. For instance, in round 3, we combined observations from three different crosscutting behaviors.

Through our experiment design, we have tried to minimize the chances of other factors being confounded with our primary independent variable: the use of aspect state machines. For example, blocking was used based on assignment marks to form the groups of subjects. Furthermore, we used a between-subjects design, i.e., each subject is exposed to one type of approach once to counterbalance the learning effects [26], i.e., since each subject is exposed to just one type of treatment (e.g., the Aspect approach) and one task (e.g., one crosscutting behavior) and thus the performance of subjects is not influenced with the experience gained while working with other treatments with the same crosscutting behavior.

One possible construct validity threat in our experiment is that

Table 8. Kruskal–Wallis test results for ERID and ERC

Round	Task	Measure	p-value (α= 0.05)
1	R1T1	ERID	**0.0001**
	R1T2	ERC	**0.0047**

we were not able to investigate all features of aspect-orientation (such as all types of basic advice) in this experiment due to the nature of our crosscutting concerns. The other concern is that the conformance error rate measures for a state machine (CERSM) can be interpreted in various ways, depending on the application domain or one's subjective opinion. However we made the effort to devise a set of objective metrics to measure conformance error rates by comparing with reference models, such that subjective perceptions and application specific measurement can be reduced to a minimum and hence the comparison of models across different case studies (perhaps with different application domains), derived by different subjects (e.g., experts, students) is possible. In addition, these metrics are general so that they are reusable and can be applied to multiple experiments. By doing so, we can therefore make sure that we don't introduce bias (beside the actual choice of those metrics) to the evaluation results which might be a threat to validity.

External validity threat is the most commonly found threat in any controlled experiment. Due to time constraints, case studies and tasks are usually small. As we see in Table 1, our models are of reasonable size. Such numbers are at least representatives of the state machines of classes and small components. However, because crosscutting concerns are expected to have an even higher impact on large models, we expect the use of AspectSM to be even more beneficial in such cases. One may also question the use of students as subjects for the experiment. Our motivation was to find a group of subjects with adequate background that could be trained to use AspectSM over a short period of time. Most practitioners have very little knowledge of AOP or AOM in general, and significant training would therefore be required. This is why relying on a group of experienced graduate students with the right educational background (Section 2) seemed to be the better option. In addition, some studies are reported in [23-25], where the performance of trained software engineering students for various tasks was compared with professional developers. The differences in performance were not statistically significant when compared to junior and intermediate developers, thus leading to the conclusion that there is no evidence that students trained for the tasks at hand may not be used as subjects in place of professionals.

6. Related Work

Most experimentation in Aspect-Oriented Software Development (AOSD) has been conducted to evaluate AOP when compared to Object-Oriented Programming (OOP) in terms of development time, errors in development, and performing maintenance tasks. An initial search on IEEE resulted in 169 papers on AOM; however, none of them reported any controlled experiment to evaluate AOM approaches. A controlled experiment [26] was performed in industry settings to measure effort and errors using aspect-oriented programming for applying different maintenance tasks related to the tracing crosscutting concern, i.e., the use of logging to record execution of a program. The results showed that aspect-orientation resulted in reducing both development effort and number of errors.

Another experiment is reported in [27], which compares aspect-orientation (AspectJ) with a more traditional approach (Java) in terms of development time for crosscutting concerns. A similar experiment is reported in [28] focusing on development time to perform debugging and change activities on object-oriented programs using AspectJ. Both of these experiments revealed mixed results, i.e., aspect-orientation has positive impact on development time only for certain tasks. For instance, AOP seems to be more beneficial when the crosscutting concern is more separable from the core behavior.

An exploratory study is reported in [29] to assess if AOP has any impact on software maintenance tasks. Eleven software professionals were asked to perform different maintenance tasks using Java and AspectJ. The results of the experiment revealed that AOP performed slightly better than OOP, but there were no statistically significant results observed. Another exploratory study is reported in [30] to measure fault-proneness with AOP. Three evolving AOP programs were used and data about different faults made during their development were collected. The experiment revealed two major findings: 1) Most of the faults were due to lack of compatibility between aspect and base code, 2) The presence of faults in AOP features such as Pointcuts, Advice, and inter-type declarations was as likely as for normal programming features. The results turned out to be statistically significant.

An experiment is reported in [31], where two software development processes based on a same aspect modeling approach (i.e., the Theme approach [32]) are compared to determine their impacts on maintenance tasks such as adding new functionality or improving existing functionality. The first process (aspectual process) involves generating AOP code in AspectJ from Theme models, whereas the second process (hybrid process) involves generating object-oriented code in Java from Theme models. Maintenance tasks are measured based on metrics such as size, coupling, cohesion, and separation of concerns. The results showed that on average the aspectual process took lesser time than the hybrid process.

An exploratory study is reported in [33], which aims to assess if aspects can help reducing effort on resolving conflicts that can occur during model compositions. To do so, they compared AOM with non-AOM in terms of effort to resolve conflicts and number of conflicts resolved on six releases of a software product line. The results of the study showed that aspects improved modularization and hence helped better localize conflicts, which in turn resulted in reducing the effort involved in resolving conflicts.

Our controlled experiment is different from the above experiments from several perspectives. First, our controlled experiment focused on the design of the software development life cycle and aspect-oriented modeling. Most of the experiments in the literature have focused on comparing AOP with OOP. We evaluated the errors in modeling made by subjects when doing different kinds of AOM tasks, i.e., defect identification, answering comprehension questionnaire, designing aspect state machines, and weaving aspect state machines.

7. Conclusion and Future Work

Aspect-oriented Modeling (AOM) has received lots of attention in the recent years, but unfortunately it lacks empirical evaluations to support its proclaimed benefits such as reduced modeling errors and reduced modeling effort. In this paper, we presented a controlled experiment, to assess conformance error rates of an AOM profile: AspectSM. This profile is developed to support state-based robustness testing at Cisco, Norway. However, it is general enough to be applied in any situation where state-based robustness testing is required. Using AspectSM, robustness crosscutting behaviors are modeled as stereotyped state machines termed as aspect state machines. Conformance error rates of applying AspectSM are assessed from four different perspectives

by conducting four modeling activities: 1) identifying modeling defects, 2) comprehending state machines, 3) modeling crosscutting behaviors, and 4) weaving crosscutting behaviors. Results of the experiment show that for most of the activities, the subjects who were given treatment AspectSM achieved significantly lower error rates than the ones given standard UML state machines.

In the future, we are planning to conduct similar controlled experiments to assess if AspectSM supports other benefits declared by AOM such as higher maintainability and changeability of models. We also plan to empirically compare AspectSM with other similar AOM profiles that support modeling crosscutting behaviors on UML state machines. In addition, we plan to compare AspectSM with domain specific languages for AOM that can be used to achieve the similar objective as AspectSM.

8. References

[1] Ali, S., Briand, L. C. and Hemmati, H. Modeling Robustness Behavior Using Aspect-Oriented Modeling to Support Robustness Testing of Industrial Systems. *Systems and Software Modeling (SOSYM)*(2011).

[2] Weigert, T. and Reed, R. *Specifying Telecommunications Systems with UML*. Kluwer Academic Publishers, 2003.

[3] Drusinsky, D. *Modeling and Verification using UML Statecharts: A Working Guide to Reactive System Design, Runtime Monitoring and Execution-based Model Checking*. Newnes, 2006.

[4] *SmartState,* http://www.smartstatestudio.com/, 2010

[5] Utting, M. and Legeard, B. *Practical Model-Based Testing: A Tools Approach*. Morgan-Kaufmann, 2007.

[6] Binder, R. V. *Testing object-oriented systems: models, patterns, and tools*. Addison-Wesley Longman Publishing Co., Inc., 1999.

[7] Cavarra, R., Crichton, C., Davies, J., Hartman, A. and Mounier, L. Using UML for automatic test generation In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA '02)* (2002).

[8] Pender, T. *UML Bible*. Wiley, 2003.

[9] Yedduladoddi, R. *Aspect Oriented Software Development: An Approach to Composing UML Design Models*. VDM Verlag Dr. Müller, 2009.

[10] *IBM OCL Parser, IBM*, http://www-01.ibm.com/software/awdtools/library/standards/ocl-download.html, 2010

[11] *OCLE,* http://lci.cs.ubbcluj.ro/ocle/, 2010

[12] *EyeOCL Software,* http://maude.sip.ucm.es/eos/, 2010

[13] Zhang, G. Towards Aspect-Oriented State Machines. In *Proceedings of the 2nd Asian Workshop on Aspect-Oriented Software Development (AOASIA'06)* (Tokyo, 2006).

[14] Zhang, G. and Hölzl, M. HiLA: High-Level Aspects for UML-State Machines. In *Proceedings of the In Proceedings of the 14th Workshop on Aspect-Oriented Modeling (AOM@MoDELS'09)* (2009).

[15] Zhang, G., Hölzl, M. M. and Knapp, A. *Enhancing UML State Machines with Aspects*. 2007.

[16] Xu, D., Xu, W. and Nygard, K. A State-Based Approach to Testing Aspect-Oriented Programs. In *Proceedings of the 17th International Conference on Software Engineering and Knowledge Engineering* (Taiwan, 2005).

[17] Whittle, J., Moreira, A., Araújo, J., Jayaraman, P., Elkhodary, A. and Rabbi, R. *An Expressive Aspect Composition Language for UML State Diagrams*. 2007.

[18] Laddad, R. *AspectJ in Action: Practical Aspect-Oriented Programming*. Manning Publications, 2003.

[19] Gomaa, H. *Designing Concurrent, Distributed, and Real-Time Applications with UML*. Addison-Wesley Professional, 2000.

[20] Wohlin, C., Runeson, P. and Höst, M. *Experimentation in Software Engineering: An Introduction*. Springer, 1999.

[21] Ali, S., Yue, T., Briand, L. C. and Malik, Z. I. *Does Aspect-Oriented Modeling Help Improve the Readability of UML State Machines?* Simula Reserach Laboratory, Technical Report(2010-11), 2010.

[22] Sheskin, D. J. *Handbook of Parametric and Nonparametric Statistical Procedures*. Chapman and Hall/CRC, 2007.

[23] Höst, M., Regnell, B. and Wohlin, C. Using Students as Subjects—A Comparative Study of Students and Professionals in Lead-Time Impact Assessment. *Empirical Software Engineering*, 5, 3I (2000), pp. 201-214.

[24] Arisholm, E. and Sjoberg, D. I. K. Evaluating the Effect of a Delegated versus Centralized Control Style on the Maintainability of Object-Oriented Software. *IEEE Transactions on Software Engineering*, 30, 8I (2004), pp. 521-534.

[25] Holt, R. W., Boehm-Davis, D. A. and Shultz, A. C. *Mental Representations of Programs for Student and Professional Programmers*. Ablex Publishing Corp., 1987.

[26] Durr, P., Bergmans, L. and Aksit, M. *A Controlled Experiment for the Assessment of Aspects - Tracing in an Industrial Context*. University of Twente, CTIT, 2008.

[27] Hanenberg, S., Kleinschmager, S. and Josupeit-Walter, M. Does aspect-oriented programming increase the development speed for crosscutting code? An empirical study. In *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement* (2009). IEEE Computer Society.

[28] Walker, R. J., Baniassad, E. L. A. and Murphy, G. C. An initial assessment of aspect-oriented programming. In *Proceedings of the 21st international conference on Software engineering* (Los Angeles, California, United States, 1999). ACM.

[29] Bartsch, M. and Harrison, R. An exploratory study of the effect of aspect-oriented programming on maintainability. *Software Quality Control*, 16, 1I (2008), pp. 23-44.

[30] Ferrari, F., Burrows, R., Lemos, v., Garcia, A., Figueiredo, E., Cacho, N., Lopes, F., Temudo, N., Silva, L., Soares, S., Rashid, A., Masiero, P., Batista, T. and Maldonado, J. An exploratory study of fault-proneness in evolving aspect-oriented programs. In *Proceedings of the Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1* (Cape Town, South Africa, 2010). ACM.

[31] Farias, K., Garcia, A. and Whittle, J. Assessing the impact of aspects on model composition effort. In *Proceedings of the Proceedings of the 9th International Conference on Aspect-Oriented Software Development* (Rennes and Saint-Malo, France). ACM.

[32] Carton, A., Driver, C., Jackson, A. and Clarke, S. *Model-Driven Theme/UML*. Springer-Verlag, 2009.

[33] Hovsepyan, A., Scandariato, R., Baelen, S. V., Berbers, Y. and Joosen, W. From aspect-oriented models to aspect-oriented code?: the maintenance perspective. In *Proceedings of the Proceedings of the 9th International Conference on Aspect-Oriented Software Development* (Rennes and Saint-Malo, France). ACM.