Softw Syst Model (2014) 13:1189–1221
DOI 10.1007/s10270-012-0293-5

REGULAR PAPER

# Does aspect-oriented modeling help improve the readability of UML state machines?

**Shaukat Ali · Tao Yue · Lionel C. Briand**

Received: 30 November 2011 / Revised: 16 July 2012 / Accepted: 18 October 2012 / Published online: 16 November 2012


**Abstract** Aspect-oriented modeling (AOM) is a relatively recent and very active field of research, whose application has, however, been limited in practice. AOM is assumed to yield several potential benefits such as enhanced modularization, easier evolution, increased reusability, and improved readability of models, as well as reduced modeling effort. However, credible, solid empirical evidence of such benefits is lacking. We evaluate the "readability" of state machines when modeling crosscutting behavior using AOM and more specifically AspectSM, a recently published UML profile. This profile extends the UML state machine notation with mechanisms to define aspects using state machines. Readability is indirectly measured through defect identification and fixing rates in state machines, and the scores obtained when answering a comprehension questionnaire about the system behavior. With AspectSM, crosscutting behavior is modeled using so-called "aspect state machines". Their readability is compared with that of system state machines directly modeling crosscutting and standard behavior together. An initial controlled experiment and a much larger replication were conducted with trained graduate students, in two different institutions and countries, to achieve the above objective. We use two baselines of comparisons—standard UML state machines without hierarchical features (flat state machines) and standard state machines with hierarchical/concurrent features (hierarchical state machines). The results showed that defect identification and fixing rates are significantly better with AspectSM than with both flat and hierarchical state machines. However, in terms of comprehension scores and inspection effort, no significant difference was observed between any of the approaches. Results of the experiments suggest that one should use, when possible, aspect state machines along with hierarchical and/or concurrent features of UML state machines to model crosscutting behaviors.

**Keywords** Aspect-oriented modeling · UML state machines · Controlled experiment · Defect identification and fixing · Comprehension

S. Ali (✉) · T. Yue · L. C. Briand
Certus Software V&V Center, Simula Research Laboratory, P.O. Box 134, 1325 Lysaker, Norway
e-mail: shaukat@simula.no

T. Yue
e-mail: tao@simula.no

L. C. Briand
SnT Centre, University of Luxembourg, Luxembourg, Luxembourg
e-mail: lionel.briand@uni.lu

## 1 Introduction

Aspect-orientation provides enhanced modularization by separating out crosscutting concerns as separate entities called aspects. Aspect-orientation is a very active field [1,2], which not only has mainly focused on aspect-oriented programming (AOP), but also led to significant progress in the realms of design and modeling, denoted as aspect-oriented modeling (AOM) [3,4]. Crosscutting concerns, for example, related to robustness or security behavior, are modeled as aspect models and are subsequently woven into a primary/base model capturing nominal functional behavior. AOM is expected to yield benefits such as improved readability, enhanced modularization, easier evolution, and increased reusability of models, as well as reduced modeling effort [4]. However, there is very little empirical evidence of such benefits. Empirical investigations, such as controlled experiments, are required to support the above claims about AOM and better understand its limitations. For example, an initial search on the IEEE, ACM, Science Direct,

Wiley Interscience, and Springer digital libraries yielded 517 papers on AOM; however, none of them reported any empirical study to evaluate its benefits. This paper is a first step in that direction and reports on the first two controlled experiments assessing the benefits of AOM.

In industrial models, such as state machines, one must capture not only nominal behavior but also robustness behavior, for example, describing how the system should react to abnormal environmental conditions. Such robustness is considered very critical in many standards such as in the IEEE Standard Dictionary of Measures of the Software Aspects of Dependability [5], the ISO's Software Quality Characteristics standard [6], and the Software Assurance Standard by NASA [7]. This is, for example, needed to support the automated robustness testing of embedded or communication systems [8] based on models. Focusing on UML state machines, as it is the most widely used notation in practice for the specification of control and communication systems [8,9], crosscutting (e.g., robustness) behavior can result in cluttered and redundant UML state machines. As a result, modeling such crosscutting behavior directly on UML state machines can be error-prone and is expected to require significant extra modeling effort.

In a recent paper we reported on AspectSM [10], a UML profile which was defined to model crosscutting behavior on UML state machines using *extended* UML state machines, to facilitate the use of AOM and limit its associated learning curve. The focus of AspectSM was on model-based test case generation for control and communication systems [8,9], though it can potentially be applied for other purposes. Comparable approaches in the literature do not use UML extension mechanisms to provide complete AOM support: they make use of specific notations for aspect-related features that do not follow any standard. With our industrial partners, and generally in most industrial settings, AOM support should be based on the UML standard to facilitate adoption. Also, support for modeling robustness behavior as a crosscutting behavior in state invariants and guards is not supported by any existing AOM approach, though they are important features in many applications, such as the generation of automated test oracles and data generation. A detailed comparison of the AspectSM profile with other related profiles can be found in [10]. AspectSM was successfully applied to model the robustness behavior of video conferencing systems for the purpose of model-based robustness testing at Cisco Systems, Norway [10]. Results suggested that on average 98 % of the modeling effort could potentially be saved. Consistent with AOM broader claims, using AspectSM to model crosscutting behavior on UML state machines as aspects should reduce cluttering and redundancy in models.

In this paper, we report the first two controlled experiments that were conducted to evaluate the "readability" of state machine modeling crosscutting behavior using AOM, in our case AspectSM. We aim to study readability via defect identification and fixing rates in state machines, as well as the scores obtained when answering a comprehension questionnaire about the system behavior. We evaluate AspectSM models by comparing them with UML state machines modeling crosscutting behavior directly. The first controlled experiment, which was smaller in scale than the second, was conducted with 27 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. The second experiment, which can be seen as a differentiated replication of the first one, was conducted at the Beijing University of Aeronautics and Astronautics (BUAA) Beijing, China, with 47 graduate students. Half of the students were taking a graduate course titled 'Software Engineering', while the remaining half were taking a course titled 'Software Architecture'. Two case study systems were used for the controlled experiments. The first one is an Elevator Control System (ECS) provided in a well-known textbook [11]— but we had to extend the case study system with two crosscutting behaviors: emergency stop and emergency call. The second case study system is a reduced version of an industrial video conferencing system developed by Cisco Systems, Norway. The readability of state machines is evaluated using three measures. The first measure is based on the ability of subjects to identify design defects seeded in state machines by checking their conformance against their specifications given as English text. The second measure is based on the ability of the subjects to fix the defects seeded in state machines. The third measure is based on subjects' scores to answer a carefully designed comprehension questionnaire. Based on these three measures, we compare the readability and also the effort resulting from using AspectSM, against both standard hierarchical and flat UML state machines. Our motivation is to assess the impact of hierarchy and/or concurrency, which is supposed to address some of the same issues as AOM in state machines (e.g., redundancy), on the relative benefits of using AspectSM.

The results of the experiments show that AspectSM helps significantly increase the identification and fixing of defects. It also leads to significantly better comprehension scores than flat state machines but hierarchical state machines look better in terms of comprehension scores, though these results were not statistically significant. In terms of the inspection effort, no significant difference was observed. For the replication, we observed similar results for defect identification and fixing, but there was no significant difference observed between any of the three approaches regarding comprehension scores.

The rest of the paper is organized as follows: Sect. 2 describes the necessary background to understand the rest of the paper, Sect. 3 provides details on planning of the initial experiment and its replication, and Sect. 4 reports on

results of the initial experiment and replication, respectively. Section 5 discusses the possible threats to validity and Sect. 6 compares existing, related experiments in Aspect-oriented Programming (AOP) to our experiments. Finally, we conclude our paper in Sect. 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 technology being evaluated in our controlled experiments.

### 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 [12] and control systems [9]. Due to the ability of state machines to capture rich and detailed information, they have been used for automatic code generation [13] and the automated generation of test cases [8,14,15]. 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 [16]. 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

This section provides an introduction to the AspectSM profile, which is used to model aspect state machines.

#### *2.2.1 Introduction*

AspectSM is a UML profile described in [10], which supports the modeling of system robustness behavior, which is very common type of crosscutting behavior in many types of systems such as communication and control systems [4]. 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. AspectSM allows modeling UML state machine aspects as UML state machines (aspect state machines). Such an approach, relying on a standard and using the target notation as the basis to model the aspects themselves, is expected to make the practical adoption of aspect modeling easier in industrial contexts. In our previous work [10], we thoroughly compared AspectSM with the similar existing AOM profiles. Our findings showed that only AspectSM is exclusively based on standard UML notation and OCL, thus eliminating the need of learning additional non-standard notations or languages, and therefore making it easy to reuse open source and commercial technology. This is highly important in most industrial contexts and strongly affects the adoption of modeling technologies. In addition, it is easy to train people in the industry for standard languages such as UML and the OCL.

Currently, AspectSM and its weaver have limited support modeling and weaving interactions [17] that may occur between different aspects and may lead to conflicts between aspects during weaving. In [18], four classes of aspect conflicts are discussed: conflicts due to crosscutting specification, aspect–aspect conflicts, aspect–base conflicts, and concern–concern conflicts. In our application context, i.e., robustness modeling and testing, the most relevant conflicts are aspect–aspect conflicts, which are related to inconsistent results when weaving aspects in a different order. Ordering conflict is most relevant in our context since, for testing purposes, we focus on modeling, weaving, and testing one or more related aspects at a time. We specify the ordering between aspect state machines in a UML state machine containing all aspect state machines as submachine states, ordered using state machine control structure features: decision, join, and fork. Interested readers may consult [10] about details on modeling the weaving order of aspects. For testing purposes, which is the focus of AspectSM at the current stage, one first has to focus on testing one concern at a time and may eventually at a later stage test several concerns together. For robustness testing, at this stage of the work, we weave faulty behavior of the environment (e.g., network) one concern at a time, as the goal is to test robustness behavior one concern at a time to facilitate debugging.

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

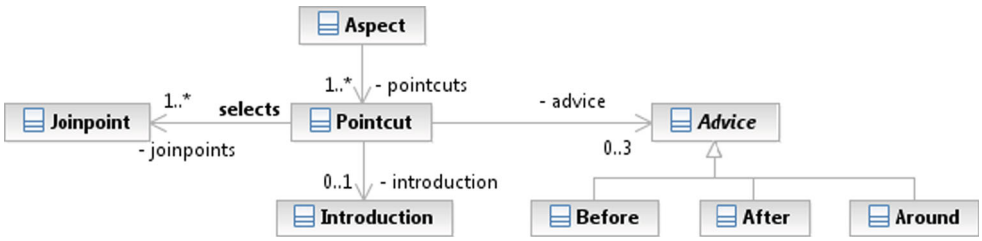


**Fig. 1** Conceptual domain model of the AspectSM profile

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. A weaver [10] 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 [4] are the most important ones. Below, we briefly describe these concepts along with how are they 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 [10]. AspectSM deals with static joinpoints and thus the corresponding weaver [10] supports static weaving or, in other words, pure syntax-based weaving. We can see from Fig. 1 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 experiments.

AspectSM was initially developed to support model-based robustness testing as it was required by the needs of our project with Cisco Systems. However, we have not investigated whether other non-functional crosscutting concerns such as security and dependability can be successfully modeled using AspectSM, but we plan to do that in the future. In addition, models developed using AspectSM (or its extensions) may be used for other purposes such as code generation and other analysis such as performance and safety, which will also be investigated in the future.

### *2.2.2 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 [4]. In the context of UML, all modeling elements in UML can be possibly joinpoints. AspectSM supports static joinpoints, which correspond to modeling elements in UML state machines for example, *State*, *Activity, Constraint, Transition, Behavior, Trigger*, and *Event*. However, we do not deal with dynamic joinpoints as for instance supported by aspect-oriented programming languages such as Aspect C++.

*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) [16] on the UML 2.0 metamodel [16]. 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 [19], OCLE 2.0 [20], and EyeOCL [21]. Furthermore, writing pointcuts as OCL expressions does not require the modeler to learn a notation that is not part of the UML standard. In the literature, several alternatives are proposed to write pointcuts [17,22–25], 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 (joinpoints) 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

**Table 1** Definition of before, around, and after advice

| State machine modeling element | Before advice | Around advice | After advice |
|---|---|---|---|
| State | Adding an OCL constraint that will be evaluated before entry to one or more states selected by a pointcut | Replacing one or more states selected by a pointcut with a new state | Adding an OCL constraint that will be evaluated on leaving one or more states selected by a pointcut |
| Transition | Adding a guard to one or more transitions selected by a pointcut. If a guard already exists, the additional constraint is conjuncted to the existing guard | Replacing one or more transitions selected by a pointcut with a new transition | Adding an effect with one or more actions to one or more transitions selected by a pointcut |
| Trigger | Not applicable | Replacing one or more triggers on transitions selected by a pointcut with new triggers | Not applicable |
| Effect | Adding a new behavior to the effect | Replacing one or more effects on transitions selected by a pointcut with a new effect | Same as before advice |
| Guard and state invariant | Add an additional constraint (conjunct) to the guards (or state invariants) selected by a pointcut | Replacing one or more guards on transitions (or state invariants) selected by a pointcut with a new guard (or a state invariant) | Same as before advice |
| Do, entry, and exit activities of a state | Adding a behavior to the activities selected by a pointcut | Replacing one or more activities in states selected by a pointcut with a new activity | Same as before advice |

by the *Pointcut*. 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*). Table 1 summarizes each type of advice for some of the key UML 2.0 state machine modeling elements. Recall that AspectSM supports static joinpoints and all these advice types are on the syntax of UML state machines. Examples of advices on various UML state machines can be found in [10].

*Introduction* An *Introduction* is similar to the inter-type declaration concept in AspectJ [26]. Using *Introduction* in our context, new modeling elements (e.g., state or transition) can be introduced into a UML state machine.

### *2.2.3 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 Fig. 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 Fig. 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* 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 a subset of states from the base state machine. There are three attributes of ≪*Pointcut*≫, whose values are shown in the note labeled as '2' in Fig. 2. The *name* attribute indicates the name of the pointcut and *type* denotes the type of the pointcut, which is *Subset* in this case. In AspectSM, different types of pointcuts can defined, and a complete list of other types of pointcuts is presented in [10]. The third attribute *selectionConstraint* contains a query in OCL on the UML state machine metamodel, which selects all states of the base state machine except *ElevatorAtFloor* and *Idle*. 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 except *ElevatorAtFloor* and *Idle* (via *SelectedStates*) 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 Experiments planning

This section discusses the planning of the experiments according to the definition and reporting template defined by Wohlin et al. [27].

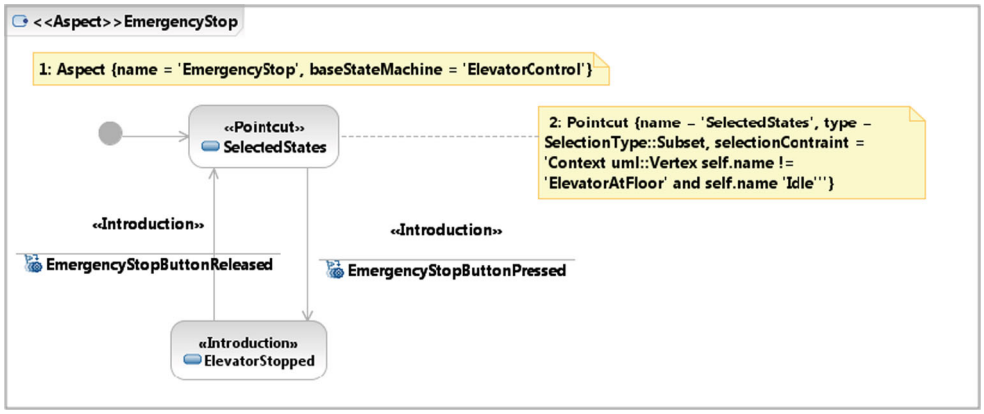


**Fig. 2** An aspect state machine for crosscutting behavior EmergencyStop. In the note numbered 2, the *not equal* operator is shown as != and not as <> (OCL syntax). This is due to the reason that we used IBM Rational Software Architect (RSA) for modeling and it cannot display <> in a note

### 3.1 Goal, research questions and hypotheses

The objective of our experiments is to assess the AspectSM profile with respect to the readability of resulting UML state machines. Readability will be looked at from three complementary points of view: model comprehensibility, the ease of detecting defects, and the ease of fixing defects for designers inspecting the models.

Based on the objective of our experiments, we defined the following four research questions.

- *RQ1: Does the use of AspectSM lead to better defect identification rate when inspecting state machines as compared with hierarchical and flat state machines?*

We wish to compare the readability of AspectSM with two different types of state machines where crosscutting behavior is modeled directly: hierarchical and flat state machines. 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.

$H_0^1$: The defect identification rate in aspect state machines is the same as that for hierarchical state machines.
$H_0^2$: The defect identification rate in aspect state machines is the same as that for flat state machines.

- *RQ2: Does the use of AspectSM lead to better defect fixing rate when inspecting state machines as compared to hierarchical and flat state machines?*

Similar to the previous question, we wish to compare the ease of defect fixing when using AspectSM with two different types of state machines directly capturing crosscutting behavior: hierarchical and flat state machines. Again, none of the expected differences between them can a priori be certain to be in a specific direction, hence leading to the definition of the following two-tailed hypotheses.

$H_0^3$: The defect fixing rate in aspect state machines is the same as that for hierarchical state machines.
$H_0^4$: The defect fixing rate in aspect state machines is the same as that for flat state machines.

- *RQ3: Does the use of AspectSM improve the ease of comprehension when compared with hierarchical and flat state machines?*

Similar to the previous research questions, we wish to compare the comprehensibility of AspectSM with the two different types of state machines directly 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.

$H_0^5$: The comprehensibility of aspect state machines is the same as that for hierarchical state machines.
$H_0^6$: The comprehensibility of aspect state machines is the same as that for flat state machines.

- *RQ4: Does the use of AspectSM reduce the required inspection effort for defect identification and answering the comprehension questionnaire?*

While the two previous research questions looked at the effectiveness of using alternative models, this research question is concerned with the effort required to inspect crosscutting behavior for defect identification and answering the comprehension questionnaire. This leads again to the following two-tailed null hypotheses:

$H_0^7$: The effort to identify defects in aspect state machines is the same as that for hierarchical state machines.
$H_0^8$: The effort to identify defects in aspect state machines is the same as that for flat state machines.
$H_0^9$: The effort to answer the comprehension questionnaire for aspect state machines is the same as that for hierarchical state machines.
$H_0^{10}$: The effort to answer the comprehension questionnaire for aspect state machines is the same as that for flat state machines.

## 3.2 Participants

The first controlled experiment was conducted at the Pir Mehr Ali Shah Arid Agriculture University, Rawalpindi, Pakistan. The subjects in the experiment were 27 graduate students taking a graduate course in 'Advanced Software Architecture' at the University Institute of Information Technology (UIIT). The course is offered in the Master of Science program. The students in this degree already hold a Bachelor's degree 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 25) have used the UML notation for their final year projects before the experiment was conducted. Twenty students had development experience in IT companies or as teaching staff in computer science.

The replication of the above experiment was conducted at the Beijing University of Aeronautics and Astronautics (BUAA), Beijing, China. The subjects in the replication are 47 graduate students. Half of the students were taking a graduate course titled 'Software Engineering', whereas the remaining half students were taking a graduate course titled 'Software Architecture'. Both courses rely on similar teaching materials and methods and we therefore can assume that all students have a similar education background regarding software engineering. These two courses are offered in the Master of Computer Software and Theory program. The students in this degree already hold a Bachelor's degree in Computer Science and had all already been exposed to UML. On average, each student went through two software development courses and one modeling course. All of the students had at least 1-year experience in development work in various industry sectors such as maritime and aerospace. In conclusion, the subjects have roughly the same background, although the subjects were in different years of their study. Their seniority was taken into the consideration while forming experimental groups as we will discuss in Sect. 3.4.

Our motivation in selecting these two groups of subjects was to find participants with adequate background (e.g., UML modeling) that could be trained to use our AOM approach over a short period of time. Our goal was to assess AspectSM with fully trained, competent participants to assess the maximum potential benefits of the approach. 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 and trained graduate students. The subjects were free to choose to participate or not in the experiments and were told their choice would have no effect on their course grades. All students underwent a specific, additional training for the experiments (Sect. 3.7). For the initial experiment, two students decided not to participate in the experiment.

## 3.3 Material

In this section, we provide details on the material we used for the experiments.

### *3.3.1 Case study system*

For the initial experiment, we only used an Elevator Control System (ECS), whereas for the replication of the experiment we used a second system as well: Video Conferencing System (VCS). Differences between the initial experiment and replication are summarized in Sect. 3.8. Information regarding the complexity of the three resulting state machines is provided in Table 2, measured using number of states and transitions for each system. For aspect state machines, we also provide the number of Pointcuts, which also contribute to modeling complexity. In Appendix A, we provide partial models of ECS to illustrate various models specified using different modeling approaches.

**Elevator control system:** It controls movements of an elevator in a building. For our experiments, we extended the specification of the elevator given in [11] with two additional crosscutting behaviors so that the AspectSM profile could be used to model them. These two crosscutting behaviors are: (1) Emergency call behavior (*Call*): the behavior of an elevator, when an emergency call is made, and (2) Emergency stop behavior (*Stop*): the behavior of an elevator, when the emergency stop button is pressed. Note that in Table 2 for ECS, in the replication, we improved the design of *Flat* and *Hierarchical* such that there are fewer states and transitions when compared with the design in the initial experiment.

**Video conferencing system:** It is a core subsystem of a video conference system called Saturn developed by Cisco Systems, Norway. The core functionality to be modeled manages the sending and receiving of multimedia streams. Audio and video signals are sent through separate channels. For the replication, we used a reduced model of Saturn that is related to establishing and disconnecting videoconferences. In addition to the core functionality, we used the following three crosscutting behaviors:

**Table 2** Complexity of the state machines modeling the crosscutting behaviors of the case study system

| System | Experiment | Crosscutting behavior | Base state machine | | Flat approach | | Hierarchical approach | | Aspect approach | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | **#S** | **#T** | **#S** | **# T** | **#S** | **# T** | **#S** | **# T** | **# P** |
| ECS | Experiment | Call | 12 | 15 | 15 | 27 | 14 | 18 | 16 | 18 | 1 |
| | | Stop | 12 | 15 | 15 | 27 | 12 | 15 | 14 | 17 | 1 |
| ECS | Replication | Call | 12 | 15 | 15 | 27 | 17 | 21 | 16 | 18 | 1 |
| | | Stop | 12 | 15 | 13 | 23 | 14 | 17 | 14 | 17 | 1 |
| VCS | Replication | AQ | 5 | 9 | 8 | 17 | 10 | 19 | 8 | 13 | 1 |
| | | DnD | 5 | 9 | 6 | 15 | 8 | 20 | 7 | 13 | 1 |
| | | Standby | 5 | 9 | 5 | 11 | 5 | 14 | 7 | 13 | 1 |

*S* states, *T* transitions, *P* pointcuts

1. *Audio Quality Loss (AQ):* An important robustness behavior of Saturn is to recover from audio quality loss. Whenever Saturn is in a video conference, it checks audio quality at regular intervals. If the quality is within threshold it continues the normal operation; otherwise, it tries to recover audio quality. If it successfully recovered the audio quality it continues its normal operation; otherwise, it restarts the VCS.
2. *Do Not Disturb (DnD):* Whenever the *Do Not Disturb* feature is on, Saturn ignores all incoming calls. If Saturn is already in a call, it will remain in the call, but ignores any new incoming calls.
3. *Standby:* The *Standby* behavior of Saturn becomes active when it is idle for *m* minutes. When any activity is performed on Saturn while it is in Standby mode, it becomes active.

The crosscutting behaviors for both systems can be modeled in three different ways: (1) by applying AspectSM to derive an aspect state machine (*Aspect* Approach), (2) by directly adding states and transitions on the base state machine (*Flat* Approach), and (3) by using hierarchical/orthogonal states (*Hierarchical* approach) to avoid redundant modeling and reduce complexity to the maximum extent. It is, however, not always possible to use the hierarchical approach successfully. For instance, separating out constraints modeling non-functional properties (e.g., video or audio quality) from state invariants is not possible using hierarchical state machines without introducing accidental complexity and redundancy as we demonstrated in [10].

### *3.3.2 Design defect classification*

Given that the correctness and completeness of defect identification through inspections are part of our evaluation criteria to compare state machines, experiment participants were asked to identify defects seeded in state machines by checking their conformance against their corresponding specifications (Sect. 3.4).

To help 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 experiments is given below and was adapted from Binder's book [8]. It was provided to the participants of the experiments as part of the answer sheet (Sect. 3.3.5) to systematically collect their answers.

*Incorrect Transition (IT):* A transition that comes from or leads to a wrong state or has an incorrect guard, trigger, and/or event.

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

*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):* According to its specification, a state that should be modeled in a state machine but is missing.

*Incorrect State (IS):* A state is incorrect if it has an incorrect state invariant, do, entry and/or exit activity.

*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 in our experiments, we are interested in studying the readability of crosscutting behaviors when applied to base behaviors since AspectSM is specifically designed for that purpose. Moreover, the readability of base behaviors that are independent of aspects is expected to be the same with or without AspectSM. For these reasons we only seeded defects in the crosscutting behaviors. Different types of defects were selected after we carefully examined the base and aspect state machines and identified possible

**Table 3** Distribution of seeded defects in state machines

| Experiment | System | Crosscutting behavior | Aspect | | | | Hierarchical | | | | Flat | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | MT | IT | MS | IS | MT | IT | MS | IS | MT | IT | MS | IS |
| Experiment | ECS | Stop | 1 | – | – | – | 1 | – | – | – | 10 | – | – | – |
| | | Call | 1 | 1 | – | 1 | 4 | 2 | – | 1 | 11 | 9 | – | – |
| Replication | ECS | Stop | 1 | – | – | – | 1 | – | – | – | 10 | – | – | – |
| | | Call | 1 | 1 | – | 1 | 1 | 1 | – | 1 | 10 | 10 | – | 1 |
| | VCS | AQ | 1 | 1 | 1 | – | 1 | 1 | 1 | – | 1 | 4 | 1 | – |
| | | DnD | – | 2 | 1 | 1 | – | 2 | 1 | 1 | – | 8 | 1 | 1 |
| | | Standby | 2 | 1 | | 1 | – | – | – | – | 2 | 1 | – | 1 |

independent defects. Table 3 shows the distribution of these defects that were seeded in the compared state machines. Note that seeded defects in ECS are different for the initial experiment and its replication since we improved the models in the latter.

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 to conceptually correspond to equivalent defects and have semantically equivalent models. To determine the number of defects in hierarchical and flat state machines that correspond to one defect in an aspect state machine, we manually wove the aspect state machine with a defect in the base state machine. Note that in Table 3, a '–' indicates that we did not seed defects from a particular defect class (e.g., MT, IT).

#### *3.3.4 Comprehension questionnaire*

As we discussed above, we also want to compare how easy it is to comprehend the 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, some questions concern what scenario is triggered when an event happens in a certain state. The subjects were asked the same ten questions on crosscutting behaviors together for all three state machines. Participants had to answer each question by inspecting the state machine assigned to them and correctness scores were computed by accounting for partially correct answers. For example, if the answer to a question entailed to list four transitions, then pointing out each correct transition contributed 0.25 to the full mark of the question.

#### *3.3.5 Answer sheets*

Three answer sheets were developed to collect answers for three readability measures (defect identification, defect fixing, and comprehension). The first answer sheet 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 the 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 was developed to collect the state machine corrected by the subjects. The third answer sheet was designed to collect answers to the comprehension questionnaire.

### 3.4 Design

In this section, we present the design of the initial experiment and its replication. In the initial experiment, we used a between-subjects [27] design for reasons discussed in Sect. 3.4.1, whereas in the replication, we used both between-subjects and within-subjects designs for each of the two rounds, respectively (Sect. 3.4.2).

#### *3.4.1 Design of the initial experiment*

The design of our experiment is summarized in Table 4. Our experiment design consists of two rounds and there were three groups denoted Group 1, Group 2, and Group 3. Given the number of the subjects, this led, respectively, to 8, 8, and 9 subjects in each group. In each round, one group was given a different type of state machines (*Aspect, Hierarchical*, or *Flat*). During the training sessions (Sect. 3.7), each subject was equally trained to understand the three different types of state machines: *Aspect, Flat*, and *Hierarchical*. The subjects were also given a modeling assignment, after

**Table 4** Design of the initial experiment

*DI* defect identification, *AC* answer comprehension questionnaire, *A* aspect, *H* hierarchical, *F* flat

| Round | Case study | Crosscutting behavior | Task | Group 1 | Group 2 | Group 3 |
|---|---|---|---|---|---|---|
| 1 | ECS | Stop | DI | A | H | F |
| | | Call | | A | H | F |
| 2 | | Stop and Call | AC | F | A | H |

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. The two rounds of the experiments were conducted in sequence on the same day.

This initial experiment used a between-subjects design, where different groups of subjects are compared when using different state machine modeling techniques. As shown in Table 4, in the first round, each group was asked to identify defects in two separate tasks corresponding to the *Call* and *Stop* crosscutting behaviors. Group 1 was given state machines modeled using the *Aspect* approach. The subjects in Group 1 were given one base state machine and one aspect state machine modeling *Call* in *Task 1*, whereas in *Task 2*, Group 1 was given the same base state machine and one aspect state machine for the *Stop* crosscutting behavior. Group 2 was given one hierarchical state machine for *Call* and one hierarchical state machine for *Stop* for *Task 1* and *Task 2*, respectively. Similarly, Group 3 was given one flat state machine for *Call* and one flat state machine for *Stop* for *Task 1* and *Task 2*, respectively. Seeded defects for each type of state machines (*Aspect, Hierarchical*, and *Flat*) are presented in Table 3. For each task, the subjects were allowed to take as much time as they needed, but when they finished the first task, their answer sheets for this task were collected and then they were handed the description of the second task and a new answer sheet. The starting and completion times were noted on each answer sheet by the subjects and were checked for correctness by the instructors while collecting the solutions.

For the second round, the three groups were rotated: Group 1 was asked to answer comprehension questionnaire for flat state machine, Group 2 for aspect state machines, and Group 3 for hierarchical state machines. This rotation was performed only for pedagogical reasons such that each group can be exposed to a different type of state machines than the previous round. However, since we had only two tasks due to time constraints, it was not possible for all of the groups to experience all three approaches. The starting and completion times for this task were collected following the same procedure as for Round 1.

#### *3.4.2 Design of the replication*

The design of the replication is summarized in Table 5. Our replication design consists once again of two rounds (Round 1 and Round 2) and each round was conducted on a separate day. During the training session (Sect. 3.7), each subject was equally trained to understand the three different types of state machines: *Aspect, Flat*, and *Hierarchical*. The subjects were divided to form blocks (i.e., groups of students of equivalent skills) based on their seniority in their graduate programs. Notice that in the initial experiment, skill level was determined based on scores of assignments, whereas in the replication skill level was determined based on the seniority of students due to practical limitations. The groups were then formed through randomization and blocking to obtain three comparable groups with similar proportions of students from each skill block. We divided the subjects into three groups: Group 1, Group 2, and Group 3. For Round 1, there were 17, 15, and 15 subjects, respectively. For Round 2, due to practical reasons such as time clash with courses and exams, fewer students participated than in Round 1. In Round 2, we had 14, 10, and 15 in Group 1, 2 and 3, respectively.

In Round 1, the ECS system and a between-subjects [27] design were used. We did not have a third crosscutting behavior to opt for a balanced, within-subjects design, as for the second round that is described next. Every participant was exposed to only one modeling approach. Group 1 was given state machines modeled using the *Aspect* (*A*) approach, Group 2 with the *Hierarchical* (*H*) approach and Group 3 with the *Flat* approach (*F*).

In Round 2, regarding detecting and fixing defects, we used a within-subjects design [27] since we have three crosscutting behaviors and three treatments (*Aspect, Hierarchical*, or *Flat*). A within-subjects design offers two main advantages. First, with this type of design, we can reduce the error variance due to individual differences in human performance, which is quite common in software engineering tasks. This is due to the fact that the same group of students is exposed to all modeling approaches across the different crosscutting behaviors (e.g., *Call* and *Stop*). Second, within-subjects designs provide more statistical power as compared with a between-subjects design [27] as it leads to more observations for each treatment. Potential threats from within-subjects designs are "carryover" effects. To address this, for each of the three crosscutting behaviors, each group was given a different

**Table 5** Design of the replication

| Round | Case study | Aspect | Task | Group 1 | Group 2 | Group 3 | Time (min) |
|---|---|---|---|---|---|---|---|
| 1 | ECS | Stop | DI | A | H | F | 15 |
| | | | DF | | | | 15 |
| | | Call | DI | A | H | F | 15 |
| | | | DF | | | | 15 |
| | | Stop and Call | AC | A | H | F | 30 |
| 2 | VCS | DnD | DI | A | H | F | 15 |
| | | | DF | | | | 15 |
| | | Standby | DI | F | A | N/A | 15 |
| | | | DF | | | | 15 |
| | | AQ | DI | H | F | A | 15 |
| | | | DF | | | | 15 |
| | | DnD, Standby, AQ | AC | A | H | F | 30 |

*DI* defect identification, *DF* defect fixing, *AC* answer comprehension questionnaire, *A* aspect, *H* hierarchical, *F* flat

treatment in such a way that ordering effects were counterbalanced: each of the three modeling approaches occurred once in a different order across the three groups. For example, as shown in Table 5, for aspect *DnD*, each group was asked to detect and fix defects and Group 1, Group 2, and Group 3 were given treatment *Aspect, Hierarchical*, and *Flat*, respectively. For *Standby*, the three groups were rotated: Group 1 was asked to identify and fix defects for flat state machines, Group 2 used aspect state machines, and Group 3 used hierarchical state machines. Similarly, the groups were rotated again for *AQ*. With a within-subjects design, a matched pair analysis can be applied by comparing the performance of subjects with themselves across treatments.

In both rounds, the subjects were presented with all three crosscutting behaviors together and were asked to answer questions from a comprehension questionnaire for one type of state machine. For each crosscutting behavior, the subjects were given a fixed time as shown in Table 5. Fixing the time for task execution tends to yield more differences in task effectiveness, but then results cannot be used to study time differences across treatments [27]. Note that in the replication, we ordered the crosscutting behaviors based on their complexity as measured by their number of modeling elements (Table 2) from simple to complex, to enable the subjects to tackle increasingly more complex models and thus smooth the learning curve.

### 3.5 Dependent variables

*Defect Identification Rate (DIR) and Defect Fixing Rate (DFR):* These variables capture whether a subject accurately identifies/fixes seeded defects. Based on the information collected in the answer sheet described in Sect. 3.3.5, there are several different ways to measure DIR and DFR, which we discuss below.

1. *Average DIR/DFR*
   For each type of defect, Average DIR/DFR (DIR_Average/DFR_Average) is measured as the percentage of identified/fixed defects over the total number of seeded defects:

   $$number\ of\ identified\ or\ fixed\ defects / total\ number\ of\ seeded\ defects$$

2. *DIR and DFR on binary scale with minimum defect identification and fixing*
   As discussed in Sect. 3.3.1, one defect seeded in aspect state machines may correspond to more than one defect in hierarchical or flat state machines. Therefore, to allow for a meaningful combination of observations across tasks and state machines, we use a binary measure indicating whether at least one defect was found (*DIR_Binary*) or fixed (*DFR_Binary*). As long as at least one defect is identified/ fixed in a given task by a subject in hierarchical and flat state machines, value 1 is assigned to *DIR_Binary/DFR_Binary*. For example, as shown in Table 3, the flat state machine modeling the *Call* crosscutting behavior contains 10 MT defects, 10 IT defects, and one IS defect. If at least any one of these defects is identified by a subject, then *DIR_Binary* = 1; otherwise *DIR_Binary* = 0. It is important to note that we developed this measure such that comparisons across the three approaches are made possible. This is due to the reason that different numbers of defects are introduced in three types of state machines corresponding to a single defect in a crosscutting behavior.
3. *DIR and DFR on binary scale with maximum defect identification and fixing*
   This measure (*DIR_Binary_Max/DFR_Binary_Max*) is a variation of *DIR_Binary/DFR_Binary*—which is also

**Table 6** Dependent variables corresponding to each research question

| Research question | Dependent variables |
|---|---|
| RQ1 | DIR_Average, DIR_Binary, DIR_Binary_Max |
| RQ2 | DFR_Average, DFR_Binary, DFR_Binary_Max |
| RQ3 | SCQ |
| RQ4 | Effort |

comparable across state machines—and is assigned value 1 when all defects seeded in a crosscutting behavior are identified/fixed by a subject in a task. For instance, in Table 3, the hierarchical state machine modeling the *Call* crosscutting behavior has 10 MT defects. *DIR_Binary_Max* = 1, if all these defects are identified by a subject; otherwise, it is assigned 0. In comparison with the measure *DIR_Binary*, this measure is stricter in the sense that it requires all the seeded defects in each of the three types of state machines to be identified to obtain a value 1. None of these measures are perfect but such binary measures are necessary to combine all observations in one data set. We will interpret differences in results of binary measures if they arise. The purpose of defining this measure is the same as for the previous measure: render possible comparisons across the three approaches, but in a different way.

4. *Score of the responses to the comprehension questionnaire (SCQ):* Correctness of the responses to the comprehensive questionnaire is calculated as follows:

$$Sum\ of\ scores\ of\ all\ questions/10$$

In the above-mentioned formula, the score for each question is calculated based on the marking procedure discussed in Sect. 3.3.4 and 10 is the total number of questions in the questionnaire.

5. *Required effort (Effort):* This is measured in minutes taken by a subject to identify/fix defects in each crosscutting behavior. Similarly, we also measure effort in minutes taken by a subject to answer the comprehension questionnaire.

Table 6 summarizes which dependent variables are used to answer research questions presented in Sect. 3.1.

### 3.6 Data collection

For the initial experiment, the solutions were collected from the subjects and were marked by the first author of this paper. In the replication, the solutions were marked by the second author of this paper. The data were encoded into a JMP [28] data file to perform the statistical analysis.

For the experiment, data integrity was checked using the following rule: for the same subjects and for each step, the starting time should precede the completion time, and the completion time of the current task must precede the starting time of the next task. For the replication, since the time for each task was fixed (Sect. 3.4.2), the answer sheets for a task were collected before handing over the next task to the subjects to ensure that each subject used exactly the same time. In addition, to avoid mistakes in marking the solutions, the first two authors double-checked the solutions marked by the other. Moreover, for a sample of randomly selected solutions, the first two authors also checked the consistency of the entries in the JMP file with the marks on the answer sheets and no inconsistencies were detected.

### 3.7 Training

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

For the replication, the subjects were trained by the second author of this paper. One 3-h session was given on the same topics as the ones used in the initial experiment. However, in this case, there were no class assignments given to the students due to practical constraints.

### 3.8 Replication

There are several potential reasons why replications of experiments are necessary in software engineering [29]. Our replication was motivated by the following reasons: (1) to reduce the validity threats that were observed in the intitial experiment, (2) to increase the sample sizes and improve the statistical power of results, and (3) to address the problems identified in the design and material. The differences between the initial experiment and its replication are summarized below:

#### *3.8.1 Reduced external validity threats*

In the replication, we reduced external validity threats by doing the following. (1) We added an additional case study,

which is a reduced version of an industrial videoconferencing system developed by Cisco, Norway. In addition, we included three real crosscutting behaviors of the videoconferencing system. (2) We replicated the experiment in a different geographical area with graduate students from a different education system.

#### *3.8.2 Improved hierarchical modeling of ECS*

The ECS system was used in both the initial experiment and its replication. For the replication, we improved the design for *Hierarchical*. The *Stop* crosscutting behavior of ECS in the replication is modeled with a reduced number of modeling elements as compared to its design in the initial experiment.

#### *3.8.3 Improved assignments of subjects to treatments*

In the initial experiment (Sect. 3.4.1), we rotated the groups for two tasks (defect identification and answering comprehension questionnaire) such that each group can inspect the state machines modeled with a different approach. Though this rotation was done for pedagogical reasons, since we had only two tasks for ECS (Sect. 3.3.1), not all of the groups could experience state machines modeled with all three approaches. In the replication, in contrast, we used a within-subjects design for the VCS system, where each group was exposed to all treatments exactly once. As discussed above, this also led to higher statistical power and a reduction in variance associated with individual differences by enabling the use of matched pair analysis.

#### *3.8.4 Other differences*

In the initial experiment, we measured readability from two perspectives: defect identification and answering a comprehension questionnaire. In the replication, we added another perspective: defect fixing. In the initial experiment, we gave subjects as much time as they wanted to perform each task. The results did not, however, reveal any significant differences between various approaches in terms of time (Sect. 4.4). In the replication, we fixed the time for each task and this expectedly led to most subjects using most of the allocated time. As expected, the differences across treatments, if any, are in such a context only visible in terms of effectiveness (e.g., defect identification/fixing rates) [27]. Exact times for tasks in the replication were estimated based on the time taken by the subjects for various tasks in the initial experiment.

### 3.9 Overview of statistical tests

In this section, we provide justifications for the statistical tests run for our data analysis.

#### *3.9.1 Statistical tests*

Using statistical testing, we check whether the differences between modeling approaches are statistically significant to determine if we can reject the null hypotheses stated in Sect. 3.1. For all statistical tests reported in this section, we used a significance level of $\alpha = 0.05$, though exact $p$ values are also reported. To check if, overall, there exist significant differences among the three approaches under investigation, we performed the one-way ANOVA test [30] on each dependent variable defined on an interval scale, i.e., *DIR_Average, DFR_Average, SCQ, and Effort*. Our samples for all dependent variables meet all assumptions of the ANOVA test, which are as follows: (1) the samples should be approximately normal, (2) the samples must be independent, and (3) variances of populations must be equal. To check for normality, we performed the Shapiro–Wilk W test [30] for each dependent variable. The results showed that their distributions do not strongly depart from normality. The second assumption also holds since our samples are collected on different groups of the subjects, working independently. To check the equivalence of variances, we performed the Bartlett's test [30] which showed that the variances across samples are equal for all dependent variables. In addition to the one-way ANOVA test, we also performed the Kruskal–Wallis one-way analysis of variance test [30], which is a non-parametric equivalent of the one-way ANOVA test. The results of both tests turned out to be consistent.

For those dependent variables for which one-way ANOVA results were significant, we performed a pair-wise comparison of the distributions obtained for the three state machines using Tukey_Kramer HSD [30], which is the ANOVA post-hoc test. As an adaptation of $t$ test, the Tukey_Kramer HSD test is designed to handle the increase in Type-I error resulting from multiple comparisons. It assumes normally distributed samples and requires samples of equal or comparable size, or otherwise yields conservative results [30]. We have (nearly) equal sample sizes (see Sect. 3.4) and our dependent variable distributions do not strongly depart from normality as the results of the Shapiro–Wilk W test [30] showed. We also report the mean differences between pairs of approaches indicating the direction in which the result is significant. We also performed the Wilcoxon Signed-Rank test [30], which is a non-parametric equivalent of Tukey_Kramer HSD. The results of both tests were consistent.

For Round 2 in the replication, since our design is a within-subjects design, we performed the matched pairs $t$ test, in addition to the one-way ANOVA and pair-wise comparisons with Tukey_Kramer HSD since matched pairs analysis improves statistical power over independent sample testing, as discussed in Sect. 3.4. In our context, a pair is the same student performing the same type of task (e.g., defect identification) on different crosscutting

behaviors (e.g., *DnD* and *Standby*) on state machines designed with different approaches (e.g., *Aspect* and *Hierarchical*). We double checked the results of the matched pairs *t* test with a Wilcoxon matched pairs test, which is an equivalent, non-parametric test. The results of the tests turned out to be consistent. Since the results of both parametric and non-parametric tests are consistent, we only report the results of the parametric tests in this paper.

For *DIR_Binary, DFR_Binary, DFR_Binary_Max,* and *DIR_Binary_Max*, we performed the Fisher's exact test [30] to compare the defect identification/fixing proportions for the various state machines. These four measures are binary and observations can be therefore classified into two categories (either 0 or 1 showing 'not found' or 'found', respectively), which is exactly what the Fisher's exact test is designed for. For these binary variables, for Round 2 in the replication, since our design is a within-subjects design, we also performed the McNemar's Test [31] for matched pairs analysis. This test is specifically designed for matched pairs analysis of binary data.

We performed all the tests mentioned in this section using JMP [28] except for the McNemar's Test [31], for which we used the web-based application [31].

#### *3.9.2 Power analysis*

Power analysis can be used during the design stage of an experiment to determine how many subjects are likely to be needed, or after to help interpret non-significant results. The latter may be due to small sample sizes and effect sizes that are smaller than expected. Power analysis is particularly important for controlled experiments in software engineering that involve human subjects, as they normally suffer from small sample sizes because of the limited availability of trained subjects and the high cost of conducting experiments. In our context, like in most software engineering experiments, the number of subjects is imposed by external constraints and a retrospective power analysis, as suggested in [32], helps interpret non-significant results in such conditions. For each statistical test considered, such an analysis estimates the minimum effect size at which we can observe an acceptable level of power (typically 80 %). This means that above that minimum, we can probably interpret a non-significant result as an absence of effect. Below this threshold the effect might be present but remain undetected.

In our experiments, we are interested in comparing the *Aspect* approach with *Hierarchical* and *Flat* approaches. We perform power analysis for the dependent variables that did not yield significant results and followed the method of calculating power as reported in [32], which requires a fixed sample size, a set significance level (0.05), and power level (80 %) and uses the observed variance to calculate the corresponding, minimum effect size. We did not use standardized effect sizes as suggested by Cohen [33] since those cannot be easily interpreted in a software engineering context.

## 4 Results and discussion

We analyze and present our experiment results in this section. We present the results for the four research questions in Sects. 4.1, 4.2, 4.3, and 4.4, respectively. Within each section, we provide results for both the initial experiment and its replication, and a plausible explanation of the results. In Sect. 4.5, we provide concluding remarks on the results and discussions.

### 4.1 Results and analysis for defect identification (RQ1)

In this section, we report results for RQ1 presented in Sect. 3.1. As shown in Table 6, we will answer this research question based on the *DIR_Average, DIR_Binary*, and *DIR_Binary_Max* dependent variables, for both the initial experiment (Sect. 4.1.1) and its replication (Sect. 4.1.2). We provide individual discussions of the results for each experiment in Sect. 4.1.3 and an overall discussion in Sect. 4.1.3.3.

#### *4.1.1 Results for the initial experiment*

Regarding *DIR_Average*, from Table 7 we can observe higher values for *Aspect* than for *Hierarchical* and *Flat*. More specifically, the *Aspect* group performed 56 and 62 % better than the *Hierarchical* and *Flat* groups. These results show that it is easier to correctly detect the defects seeded in aspect state machines than in the flat and hierarchical state machines. The most plausible explanation is that the number of model elements (Sect. 3.3.1) for aspect state machines is lower than in the other two types of state machines (Table 2) and complexity of pointcuts written as OCL queries does not override this effect. In addition, we checked whether the differences observed for *DIR_Average* are statistically significant to determine if we can reject the null hypotheses stated in Sect. 3.1. As shown in Table 8, we observed significant differences for *DIR_Average*. Since the results were statistically significant, we further performed Tukey_Kramer HSD for a pair-wise comparison of modeling approaches. The results showed that *Aspect* significantly outperformed both *Flat* and *Hierarchical* in terms of *DIR_Average* as $p$ values are lower than $\alpha$ (Table 9).

For *DIR_Binary* as shown in Table 7, for *Aspect*, 93.7 % of the subjects managed to catch at least one defect from any of the defect types seeded in both tasks. This is 37.5 and 27 % higher than for *Hierarchical* and *Flat*, respectively. For *DIR_Binary_Max*, we observed a pattern similar to *DIR_Binary* for both tasks, as shown in Table 7. *DIR_Binary_Max* is higher for *Aspect* than that of

**Table 7** Descriptive statistics for various DIR measures

| Experiment | System | Measure | Crosscutting behavior | Approach | | |
|---|---|---|---|---|---|---|
| | | | | Aspect | Hierarchical | Flat |
| Experiment | ECS | DIR_Average | Call and Stop | 0.81 | 0.25 | 0.19 |
| | | DIR_Binary | | 0.94 | 0.56 | 0.67 |
| | | DIR_Binary_Max | | 0.69 | 0.13 | 0.28 |
| Replication | ECS | DIR_Average | Stop | 0.29 | 0.46 | 0.40 |
| | | DIR_Binary | | 0.29 | 0.46 | 0.73 |
| | | DIR_Binary_Max | | 0.29 | 0.46 | 0.06 |
| | | DIR_Average | Call | 0.27 | 0.53 | 0.16 |
| | | DIR_Binary | | 0.52 | 0.93 | 0.73 |
| | | DIR_Binary_Max | | 0.05 | 0.06 | 0 |
| | | DIR_Average | Call and Stop | 0.28 | 0.5 | 0.28 |
| | | DIR_Binary | | 0.41 | 0.7 | 0.73 |
| | | DIR_Binary_Max | | 0.17 | 0.26 | 0.03 |
| | VCS | DIR_Average | DnD | 0.30 | 0.1 | 0.18 |
| | | DIR_Binary | | 0.71 | 0.4 | 0.6 |
| | | DIR_Binary_Max | | 0 | 0 | 0 |
| | | DIR_Average | Standby | 0.6 | – | 0.33 |
| | | DIR_Binary | | 0.6 | – | 0.64 |
| | | DIR_Binary_Max | | 0.6 | – | 0 |
| | | DIR_Average | AQ | 0.25 | 0.19 | 0.26 |
| | | DIR_Binary | | 0.66 | 0.57 | 1 |
| | | DIR_Binary_Max | | 0 | 0 | 0 |
| | | DIR_Average | DnD, Standby, and AQ | 0.36 | 0.15 | 0.25 |
| | | DIR_Binary | | 0.66 | 0.5 | 0.71 |
| | | DIR_Binary_Max | | 0.15 | 0 | 0 |

*Hierarchical* and *Flat*, i.e., for the *Aspect* group, 68.7 % of the subjects managed to find all the defects seeded in both tasks, which is 56.2 % more than for the *Hierarchical* group and 40.9 % more than for *Flat* (see Table 7). As discussed in Sect. 3.9, we performed the Fisher's exact test to check statistical significance of difference in binary variables and the results are provided in Table 10. For *DIR_Binary, Aspect* significantly outperformed *Hierarchical*, but there were no significant differences observed for *Aspect* versus *Flat*. In the case of *DIR_Binary_Max, Aspect* significantly outperformed both *Hierarchical* and *Flat*.

**Table 8** Results for one-way ANOVA for DIR_Average

| Experiment | System | Crosscutting behavior | *p* value |
|---|---|---|---|
| Experiment | ECS | Call and Stop | **0.0001** |
| Replication | ECS | Stop | 0.55 |
| | | Call | **0.003** |
| | | Stop and Call | **0.04** |
| | VCS | DnD | 0.10 |
| | | Standby | 0.12 |
| | | AQ | 0.64 |
| | | DnD, Standby, and AQ | **0.02** |

### *4.1.2 Results for the replication*

In this section, we provide results for the replication for defect identification. Sect. 4.1.2.1 provides the results for ECS, whereas Sect. 4.1.2.2 provides the results for VCS.

*4.1.2.1 Results for the ECS System* Table 7 shows descriptive statistics for various measures of *ECS*. For *Stop, DIR_Average* for *Hierarchical* (0.46) and *Flat* (0.40) is better than *Aspect* (0.29). For *Call*, again *Hierarchical* has higher *DIR_Average* (0.53) than *Aspect* (0.27). However, in this case *Aspect* has higher *DIR_Average* than *Flat* (0.16). For *Stop* and *Call* together *Hierarchical* has higher *DIR_Average* (0.53) than *Aspect* and *Flat*, and *DIR_Average* is tied between *Aspect* and *Flat*. For *Stop, DIR_Binary* is higher (0.73) for *Flat* than *Hierarchical* and *Aspect*, which are 0.46 and 0.29, respectively. *DIR_Binary* of *Call* for *Hierarchical*

**Table 9** Comparisons of all pairs for DIR_Average using Tukey_Kramer HSD

| Experiment | System | Crosscutting behavior | Aspect versus Hierarchical | | Aspect versus Flat | |
|---|---|---|---|---|---|---|
| | | | Mean difference (Aspect−Hierarchical) | *p* value | Mean difference (Aspect−Flat) | *p* value |
| Experiment | ECS | Stop and Call | 0.36 | **0.02** | 0.44 | **0.005** |
| Replication | ECS | Call | −0.25 | **0.04** | 0.11 | 0.48 |
| | | Stop and Call | −0.21 | **0.03** | 0.003 | 0.99 |
| | VCS | DnD, Standby, and AQ | 0.20 | **0.01** | 0.10 | 0.27 |

**Table 10** Two-tailed Fisher's exact test for DIR binary measures at $\alpha = 0.05$

| Experiment | System | Measure | Aspect versus Hierarchical | | Aspect versus Flat | |
|---|---|---|---|---|---|---|
| | | | Difference in proportion (Aspect−Hierarchical) | *p* value | Difference in proportion (Aspect−Flat) | *p* value |
| Experiment | ECS | DIR_Binary | 0.375 | **0.03** | 0.27 | 0.09 |
| | | DIR_Binary_Max | 0.56 | **0.003** | 0.40 | **0.03** |
| Replication | ECS | DIR_Binary (Stop) | −0.17 | 0.46 | −0.43 | **0.03** |
| | | DIR_Binary_Max (Stop) | −0.17 | 0.46 | 0.22 | 0.17 |
| | | DIR_Binary (Call) | −0.40 | **0.01** | −0.20 | 0.29 |
| | | DIR_Binary_Max (Call) | −0.0007 | 1 | 0.05 | 1 |
| | | DIR_Binary | −0.28 | **0.02** | −0.32 | **0.01** |
| | | DIR_Binary_Max | −0.09 | 0.54 | 0.14 | 0.10 |
| | VCS | DIR_Binary (DnD) | 0.31 | 0.21 | 0.11 | 0.69 |
| | | DIR_Binary_Max (DnD) | 0 | – | 0 | – |
| | | DIR_Binary (Standby) | – | – | −0.04 | 1 |
| | | DIR_Binary_Max (StandBy) | – | – | 0.6 | **0.001** |
| | | DIR_Binary (AQ) | 0.09 | 0.71 | −0.33 | 0.06 |
| | | DIR_Binary_Max (AQ) | 0 | – | 0 | – |
| | | DIR_Binary | 0.16 | 0.28 | −0.05 | 0.80 |
| | | DIR_Binary_Max | 0.15 | 0.07 | 0.15 | **0.02** |

(0.93) is higher than *Aspect* (0.52) and *Flat* (0.73), respectively. For *Call* and *Stop* together, *Flat* (0.73) has higher *DIR_Binary* than *Hierarchical* (0.7) and *Flat* (0.41). For *DIR_Binary_Max* in *Stop*, *Hierarchical* (0.46) outperformed *Aspect* (0.15) and *Flat* (0.40), but for *Call, Hierarchical* and *Aspect* show values for *DIR_Binary_Max* of 0.06 and 0.5, respectively, whereas *Flat* has *DIR_Binary_Max* of 0. For *Stop* and *Call* together, *Hierarchical* is better than both *Aspect* and *Flat*.

In addition, we checked the statistical significance of DIR_Average using one-way ANOVA, as discussed in Sect. 3.9. Table 8 shows the ANOVA results for ECS, where the *p* values are made bold when below than our chosen significance level (0.05). For ECS, we observed significant differences in *DIR_Average* for *Call* individually and *Stop* and *Call* together. We then performed a pair-wise comparison of the distributions obtained for the three state machines using Tukey_Kramer HSD [30]. The results are presented in Table 9. For *DIR_Binary* and *DIR_Binary_Max*, we performed the two-tailed Fisher's Exact test, whose results are also summarized in Table 10.

*4.1.2.2 Results for the VCS system* Table 7 shows the descriptive statistics for various measures of VCS. For *DnD, Aspect* outperformed both *Hierarchical* and *Flat* for *DIR_Average* and *DIR_Binary*; however, *DIR_Binary_Max* is 0 for all groups. Note that we could not model the *Standby* crosscutting behavior with *Hierarchical*. Again, for *Standby, Aspect* outperformed *Flat* for *DIR_Average* and *DIR_Binary_Max*, whereas we observed the reverse for *DIR_Binary* (Table 7). In case of *DIR_Average* and *DIR_Binary* for *AQ, Flat* outperformed *Aspect*, which in turn outperformed *Hierarchical*. For all three crosscutting behaviors together, *Aspect* outperformed *Hierarchical* and *Flat* in terms of *DIR_Average*

**Table 11** Results of the matched pairs for VCS at $\alpha = 0.05$ for various DIR measures

| Measure | Test | Pair of approaches | Mean difference | $p$ value |
|---|---|---|---|---|
| DIR_Average | $t$ test | Aspect-Hierarchical | 0.27 | **0.002** |
| DIR_Average | | Aspect–Flat | 0.19 | **0.02** |
| DIR_Binary | McNemar's test | Aspect–Hierarchical | 0.16 | **0.03** |
| DIR_Binary | | Aspect–Flat | −0.05 | **0.02** |
| DIR_Binary_Max | | Aspect–Hierarchical | 0.15 | **0.001** |
| DIR_Binary_Max | | Aspect–Flat | 0.15 | **5.42e−07** |

**Table 12** Estimation of the effect size corresponding to 80 % power for ECS

| Experiment | Measure | $p$ value | Observed effect size | Minimum effect size | Average | Minimum effect size/average |
|---|---|---|---|---|---|---|
| Experiment | DIR_Binary (A vs. F) | 0.09 | 0.14 | 0.20 | 0.80 | 0.24 |
| Replication | DIR_Average (A vs. F) | 0.99 | 0.001 | 0.13 | 0.29 | 0.46 |
| | DIR_Binary_Max (A vs. H) | 0.54 | 0.04 | 0.15 | 0.22 | 0.69 |

*A* aspect, *H* hierarchical, *F* flat

and *DIR_Binary_Max*, whereas for *DIR_Binary, Flat* (1.0) outperformed *Aspect* (0.66), which in turn outperformed *Hierarchical* (0.57).

In addition, the ANOVA results (Table 8) showed significant differences in *DIR_Average* with Call individually and Stop and Call together. We, therefore, performed a pairwise comparison of the distributions obtained for the three state machines using Tukey_Kramer HSD [30]. The results of the test are reported in Table 9. For VCS, in addition we performed the matched pairs $t$ test (Sect. 3.9) as reported in Table 11. We observed that *Aspect* significantly outperformed *Hierarchical* and *Flat* with $p$ values of 0.002 and 0.02 (Table 11), respectively. Hence, it shows that *Aspect* has a high likelihood of having higher *DIR_Average* than both *Flat* and *Hierarchical*. For *DIR_Binary* and *DIR_Binary_Max*, we performed the two-tailed Fisher's Exact test, whose results are also summarized in Table 10. The results of the McNemar's test for matched pairs analysis of these binary measures are reported in Table 11. For *DIR_Binary_Max, Aspect* significantly performed better than both *Hierarchical* and *Flat*. For *DIR_Binary, Aspect* significantly outperformed *Hierarchical*, but *Flat* significantly performed better than *Aspect*.

### *4.1.3 Discussion*

In this section, we discuss the results reported in Sect. 4.1.1 and Sect. 4.1.2. First, we provide discussion of the results for each experiment (Sect. 4.1.3.1 and Sect. 4.1.3.2) individually followed by an overall discussion in Sect. 4.1.3.3.

*4.1.3.1 Analysis of results for the initial experiment* Based on the experiment results reported in Sect. 4.1.1, we conclude that overall, *Aspect* state machines are better than *Flat* and *Hierarchical* ones in terms of the overall defect identification rate, even though the difference between *Aspect* and *Flat* for one of the binary measures (*DIR_Binary*) is not statistically significant given our selected $\alpha$ (0.05) and sample size. One reasonable explanation is that, when compared with flat and hierarchical state machines, aspect state machines are much less complex in terms of number of states and transitions (Table 2); therefore it is expected to be much easier to identify defects in aspect state machines. It is also interesting to note that the additional complexity introduced by pointcuts in *Aspect* does not have any visible negative effect on defect identification.

We further analyzed non-significant results using the power analysis reported in Table 12. The table shows the estimated effect size thresholds corresponding to 80 % power for *DIR_Binary* (*Aspect* vs. *Flat*) that yielded non-significant results in the previous section (*Minimum effect size*). This means that for effect sizes less than these thresholds, power is less than 80 % thus entailing a significant risk of error (type II) in not rejecting the null hypotheses. In other words, for effect sizes below these thresholds, we cannot draw conclusions with confidence from the statistical test results in Table 12. The *Average* column in Table 12 shows the average values for the dependent variables, when combining all the observations being compared. The last column shows the percentage of *Average* that corresponds to the minimum effect size. The result of power analysis for *DIR_Binary* regarding *Aspect* vs. *Flat* (Table 12) shows an estimated effect size of 0.20 (24 % of average) to achieve 80 % power. The observed effect size is 0.14, which is lower than this estimated effect size thus explaining the lack of significance. This suggests that we need to collect more

**Table 13** Summary of statistically significant results for DIR measures

| Dependent variable | Approach pair | Round 1 | Round 2 | | |
|---|---|---|---|---|---|
| | | ECS | VCS (Tukey_Kramer HSD) | VCS (Matched pairs) | |
| | | Crosscutting behavior (X) | Crosscutting behavior (X) | Test | Crosscutting behavior (X) |
| DIR_Average | A > X | – | All (H) | *t* test | All (H), All (F) |
| | X > A | Call (H), All (H) | – | | – |
| DIR_Binary | A > X | – | – | McNemar's test | All (H) |
| | X > A | Stop (F), Call (H), All (F), All (H) | – | | All (F) |
| DIR_Binary_Max | A > X | – | Standby (F), All (F) | | All (H), All (F) |
| | X > A | – | – | | – |

*X* either H (Hierarchical) or F (Flat), *A* aspect, *H* hierarchical, *F* flat, '–' non-significant results

observations, if we want to draw conclusions with confidence for effect sizes below 24 % of the average, regarding which approach (*Aspect* or *Flat*) is better in terms of *DIR_Binary*.

*4.1.3.2 Analysis of results for the replication* In this section, we provide a discussion on DIRs for each crosscutting behavior individually and all crosscutting behaviors together for the replication. Recall that DIRs are measured with three dependent variables: *DIR_Average, DIR_Binary*, and *DIR_Binary_Max*. Results for all those variables for which the results were statistically significant are summarized in Table 13. The first column lists the dependent variables which are used to answer RQ1 (Table 6). The second column represents a pair of approaches being compared and each dependent variable has two rows in this column: $A > X$ and $X > A$, denoting whether *Aspect* (A) is significantly better than *Hierarchical* or *Flat* (X), and *Hierarchical* or *Flat* (X) are significantly better than *Aspect* (A), respectively. The third column (labeled "Crosscutting Behavior (X)") presents two pieces of information: (1) name(s) of the crosscutting behavior(s) for which the results were significant for ECS, (2) name of the approach in brackets against which the results were significant, i.e., the approach is either significantly better than *Aspect* if it is in the row $X > A$ or vice-versa if it is in the row A>X. For example, in case of *DIR_Average* in the row labeled "$X > A$", *Call* (*H*) means that *Hierarchical* is significantly better than *Aspect* for the *Call* crosscutting behavior. If results were significant for all crosscutting behaviors together, for instance in the case of ECS, when the observations are combined for *Call* and *Stop* for a dependent variable (e.g., *DIR_Average*), we denote it as *All* in the table. The fourth column is similar to the third column except that it presents the results of VCS. The sixth column is similar to the fourth column, but the only difference is that the sixth column represents the results of the matched pairs tests, whereas the fourth column shows the results of Tukey_Kramer HSD for VCS. The fifth column represents the type of the matched pairs tests applied to each dependent variable. For instance, the McNemar's test is applied to the two binary dependent variables. Non-significant results are indicated by "–" in Table 13.

From Table 13, we can see that in the case of the ECS system, we observed a significance difference across the three approaches for *Call* and for *Stop* and *Call* together in terms of *DIR_Average*, where *Hierarchical* fared significantly better than *Aspect*. This could be due to the reason that in this first round, students were more familiar with standard UML state machines as compared with aspect state machines. For VCS, in case of *DIR_Average, Aspect* has significantly higher *DIR_Average* than *Hierarchical* for all crosscutting behaviors together (column 4, row 1, in Table 13). The results of the matched pairs *t* test on VCS show consistent results with Tukey_Kramer HSD, since in both cases *Aspect* significantly outperformed *Hierarchical* and*Flat*.

In case of *DIR_Binary*, for E CS, again *Hierarchical* and *Flat* significantly performed better than *Aspect*, whereas for VCS we did not observe significant differences between approaches using the Tukey_Kramer HSD test. However, based on the results of matched pairs analysis with the McNemar's test for *DIR_Binary*, we observed that *Aspect* significantly outperformed *Hierarchical*, whereas *Flat* significantly outperformed *Aspect*. Regarding the latter, it could be due to an inherent bias of *DIR_Binary* towards *Flat* as finding just one defect out of all seeded defects will give *Flat* a maximum score (Sect. 3.5). For *DIR_Binary_Max* in ECS we did not observe any significant differences. With VCS, *Aspect* significantly outperformed *Flat* for *Standby* and *Aspect* significantly performed better than *Flat* for all crosscutting behaviors together. Similar results were observed for the McNemar's test for *DIR_Binary_Max*, where *Aspect* outperformed both *Flat* and *Hierarchical*. In conclusion, a plausible explanation for the results presented above is that, when compared with flat and hierarchical state machines, aspect state

**Table 14** Estimation of the effect size corresponding to 80 % power for VCS

| Dependent variable | *p* value | Observed effect size | Minimum effect size | Average | Minimum effect size/average |
|---|---|---|---|---|---|
| DIR_ Average (A vs. F) | 0.27 | 0.05 | 0.1 | 0.31 | 0.32 |
| DIR_Binary (A vs. H) | 0.28 | 0.1 | 0.11 | 0.28 | 0.39 |
| DIR_Binary (A vs. F) | 0.80 | 0.05 | 0.11 | 0.31 | 0.35 |
| DIR_Binary_Max (A vs. H) | 0.54 | 0.08 | 0.18 | 0.6 | 0.30 |

*A* aspect, *H* hierarchical, *F* flat

**Table 15** Summary of statistically significant results for both experiments

| Dependent variable | Pair of approaches | Experiment | Replication | | | |
|---|---|---|---|---|---|---|
| | | ECS | ECS | VCS (Tukey_Kramer HSD) | VCS (Matched Pairs) | |
| DIR_Average | A > X | H, F | – | H | *t* test | H, F |
| | X > A | – | H | – | | – |
| DIR_Binary | A > X | H | – | – | McNemar's test | H |
| | X > A | – | H, F | – | | F |
| DIR_Binary_Max | A > X | H, F | – | F | | H, F |
| | X > A | – | – | – | | – |

*X* either H (Hierarchical) or F (Flat), *A* aspect, *H* hierarchical, *F* flat, '–' non-significant results

machines are much less complex in terms of number of states and transitions (Table 2); therefore, it is expected to be easier to identify defects in aspect state machines. The fact that the reader has to mentally weave the aspects with the base state machines to get the full picture does not seem to be a severe hindrance for these defect identification tasks.

By looking at the above results, it is also interesting to note that the results of Round 2 are different than those of Round 1 since all the results in Round 2, as opposed to Round 1, are in favor of AspectSM, except the McNemar's test results for *DIR_Binary* between *Aspect* and *Flat*. This could be due to the reason that AspectSM entails a steep learning curve as the experience gained by the subjects of the *Aspect* group in Round 1 helped them in performing significantly better than the subjects in other groups in Round 2.

To discuss non-significant results, we performed power analysis, which results are summarized in Tables 12 and 14. Note that we did so only for those cases where the results were not significant when observations were combined for all crosscutting behaviors. In case of *DIR_Average* (*Stop* and *Call*) regarding the three approaches (Table 12), an estimated minimum effect size of 0.13 (46 % of average) is required to achieve 80 % power as shown in Table 12. The observed effect size is 0.001, which is much lower than 0.13. Since this is a quite large effect size threshold, to draw useful conclusions with confidence regarding which approach (*Aspect* or *Flat* or *Hierarchical*) is better in terms of *DIR_Average* for *Stop* and *Call*, we probably need more observations. Similar results are obtained for other dependent variables for which the results were not significant, as shown in Tables 12 and 14.

*4.1.3.3 Overall discussion* In this section, we discuss the results of RQ1 for the initial experiment and the replication together. Table 15 summarizes the statistically significant results of the initial experiment and its replication. The first column represents dependent variables for defect identification, i.e., *DIR_Average, DIR_Binary*, and *DIR_Binary_Max*. The second column denotes the pair of approaches being compared, e.g., $A > X$ reports on whether the *Aspect* (A) approach is significantly better than *Hierarchical* and/or *Flat* (X). In our particular case, we have three approaches *Aspect, Hierarchical*, and *Flat* denoted as $A$, $H$, and $F$, respectively, in the table. In addition, each dependent variable has two corresponding rows: $A > X$ and $X > A$, reporting on whether *Aspect* is significantly better than *Hierarchical* or *Flat*, and *Hierarchical* or *Flat* are significantly better than *Aspect*, respectively. The third column tells the name(s) of the approaches(s) for which the results were significant for ECS in the experiment. For example, for RQ1, in the case of *DIR_Average* in the row labeled "$A > X$", $H$ means that *Aspect* is significantly better than *Hierarchical*. The fourth and fifth columns are similar to the third column, but the only difference is that these columns represent the results for ECS and VCS for the replication using Tukey_Kramer HSD. The seventh column presents the results of matched pairs for VCS, whereas the sixth column lists tests being applied for matched pairs analysis for all the dependent variables. In the table, "–" indicates non-significant results.

For *DIR_Average*, in the experiment, for ECS, *Aspect* performed significantly better than both *Flat* and *Hierarchical*. In contrast, in the replication, we observed that *Hierarchical*

**Table 16** Descriptive statistics for various DFR measures

| System | Measure | Crosscutting behavior | Approach | | |
|---|---|---|---|---|---|
| | | | Aspect | Hierarchical | Flat |
| ECS | DFR_Average | Stop | 0.64 | 0.66 | 0.49 |
| | DFR_Binary | | 0.64 | 0.66 | 0.93 |
| | DFR_Binary_Max | | 0.64 | 0.66 | 0 |
| | DFR_Average | Call | 0.64 | 0.63 | 0.31 |
| | DFR_Binary | | 0.94 | 0.86 | 0.93 |
| | DFR_Binary_Max | | 0.47 | 0.4 | 0.13 |
| | DFR_Average | Call and Stop | 0.64 | 0.65 | 0.40 |
| | DFR_Binary | | 0.79 | 0.76 | 0.93 |
| | DFR_Binary_Max | | 0.55 | 0.53 | 0.06 |
| VCS | DFR_Average | DnD | 0.5 | 0.125 | 0.16 |
| | DFR_Binary | | 0.71 | 0.4 | 0.26 |
| | DFR_Binary_Max | | 0.28 | 0 | 0 |
| | DFR_Average | Standby | 0.4 | – | 0.59 |
| | DIR_Binary | | 0.4 | – | 0.85 |
| | DFR_Binary_Max | | 0.4 | – | 0.07 |
| | DFR_Average | AQ | 0.62 | 0.16 | 0.43 |
| | DFR_Binary | | 1 | 0.5 | 0.7 |
| | DFR_Binary_Max | | 0.33 | 0 | 0 |
| | DFR_Average | DnD, Standby, and AQ | 0.52 | 0.14 | 0.38 |
| | DFR_Binary | | 0.74 | 0.45 | 0.58 |
| | DFR_Binary_Max | | 0.33 | 0 | 0.02 |

outperformed *Aspect* for *DIR_Average*. This can be explained from the fact that the subjects in the initial experiment had more training and previous experience in modeling as compared with the subjects in the replication (Sect. 3.7). This can be further seen from the results of the VCS system in the replication, where *Aspect* significantly performed better than *Hierarchical* and *Flat* using matched pairs analysis for *DIR_Average*, consistent with those for the ECS system in the initial experiment.

We observed similar results for *DIR_Binary*. In the initial experiment, *Aspect* significantly outperformed *Hierarchical* for ECS but for the replication, we observed that *Flat* and *Hierarchical* performed significantly better than *Aspect*. Again, this is probably due to the differences in training that the subjects received in the initial experiment and replication. For *DIR_Binary_Max*, we observed consistent results for the initial experiment and the replication, in which *Aspect* outperformed *Flat* and *Hierarchical*.

## 4.2 Results and analysis for defect fixing (RQ2)

In this section, we present results for defect fixing (RQ2) based on the *DFR_Average, DFR_Binary*, and *DFR_Binary_Max* dependent variables (Sect. 3.5). Recall from Sect. 3.8 that defect fixing was only conducted in the replication.

### *4.2.1 Results for the ECS system*

For ECS, in case of the *Stop* crosscutting behavior (Table 16), *Hierarchical* scored 0.66 for *DFR_Average* outperforming *Aspect* (0.64) and *Flat* (0.49). For *DFR_Binary*, *Flat* (0.93) outperformed *Hierarchical* (0.66) and *Aspect* (0.64). For *DFR_Binary_Max* (Table 16), *Hierarchical* (0.66) outperformed *Aspect* (0.64) and *Flat* (0). In *Call, Aspect* (0.64) outperformed both *Hierarchical* (0.63) and *Flat* (0.31) and similar results were observed for *DFR_Binary* and *DFR_Binary_Max* for *Stop*. For *Call* and *Stop* taken together, *Hierarchical* scored 0.65 for *DFR_Average*, outperforming *Aspect* (0.64) and *Flat* (0.40). For *DFR_Binary, Flat* (0.93) outperformed *Aspect* (0.79) and *Hierarchical* (0.76) respectively, whereas *Aspect* (0.55) outperformed *Hierarchical* (0.53) and *Flat* (0.06) for *DFR_Binary_Max* (Table 16).

The one-way ANOVA results presented in Table 17 show that there are significant differences for *DFR_Average* (*Call*) and *DFR_Average* (*Stop* and *Call*). For these variables, we performed a pair-wise comparison of the distributions obtained for the three state machines using Tukey_Kramer HSD [30], reported in Table 18. The results of the two-tailed Fisher exact test for binary variables (*DFR_Binary* and *DFR_Binary_Max*) are shown in Table 19, where

**Table 17** Results of One-way ANOVA for DFR_Average at $\alpha = 0.05$

| System | Crosscutting behavior | $p$ value |
|---|---|---|
| ECS | Stop | 0.48 |
| | Call | **0.03** |
| | Stop and Call | **0.02** |
| VCS | DnD | **0.008** |
| | Standby | 0.23 |
| | AQ | **0.0003** |
| | DnD, Standby, AQ | **0.0003** |

$p$ values are given in bold when below our selected level of significance.

#### 4.2.2 Results for the VCS system

For VCS, in case of *DnD, Aspect* outperformed both *Hierarchical* and *Flat* for all three defect fixing measures as it can be seen from the means reported in Table 16. For the *Standby* crosscutting behavior, for *DFR_Average* and *DFR_Binary, Flat* outperformed *Aspect*, whereas *Aspect* outperformed *Flat* for *DFR_Binary_Max*. Recall that for *Standby*, we did not have a solution using the *Hierarchical* approach. For *AQ, Aspect* outperformed *Hierarchical* and *Flat* for all three defect fixing measures. For all three crosscutting behaviors together, *Aspect* outperformed *Hierarchical* and *Flat* for all three defect fixing dependent variables.

The results of one-way ANOVA presented in Table 17 show that there are significant differences for *DFR_Average* (*DnD*), *DFR_Average* (*AQ*), and *DFR_Average* (*DnD, Standby*, and *AQ*). For these variables, since one-way ANOVA results were significant, we performed a pair-wise comparison of the distributions obtained for the three state machines using Tukey_Kramer HSD [30] and the results are given in Table 18. For binary variables *DFR_Binary* and *DFR_Binary_Max*, we report the results of the Fisher exact test in Table 19. In all these tables, bold $p$ values highlight statistically significant results and the mean differences between pairs of approaches indicate the direction of the effect.

The results for the matched pairs $t$ test for *DFR_Average* are shown in Table 20. For all three crosscutting behaviors together, *Aspect* significantly outperformed *Hierarchical*; however, there is no significant difference between *Aspect* and *Flat*. For matched pairs analysis of the binary dependent variables, the results of the McNemar's test are shown in Table 20, where *Aspect* significantly outperformed *Hierarchical* and *Flat* regarding *DFR_Binary_Max*. For *DFR_Binary*, a significant difference is once again observed between *Aspect* and *Flat*, but not between *Aspect* and *Hierarchical*.

#### 4.2.3 Discussion

In this section, we provide a discussion on DFRs for each crosscutting behavior individually and all crosscutting behaviors together. DFRs are measured based on three dependent variables: *DFR_Average, DFR_Binary*, and *DFR_Binary_Max*. Statistically significant results are summarized in Table 21. The first column lists the dependent variables which are used to answer RQ2 (Table 6). The second column denotes pairs of approaches being compared and each dependent variable has two rows in this column: $A > X$ and $X > A$ denoting whether *Aspect* (A) is significantly better than *Hierarchical* or *Flat* (X), and *Hierarchical* or *Flat* (X) are significantly better than *Aspect* (A), respectively. The third column (labeled "Crosscutting Behavior (X)") presents two pieces of information: (1) name(s) of the crosscutting behavior(s) for which the results were significant for ECS, (2) name of the approach in brackets against which the results were significant, i.e., the approach is either significantly better than *Aspect* if located in row X > A or vice versa if located in row A > X. For example, in case of *DIR_Average* in the row labeled "$X > A$", *Call* (*H*) means that *Hierarchical* is significantly better than *Aspect* for the *Call* crosscutting behavior. If results were significant for all crosscutting behaviors together, for instance in the case of ECS, when the observations are combined for *Call* and *Stop* for a dependent variable (e.g., *DIR_Average*), we denote it as *All* in the table. The fourth column is similar to the third column except that it presents the results of VCS. The fifth column is similar to the fourth column, but the only difference is that it reports the results of the matched pairs $t$ test, whereas

**Table 18** Comparisons of all pairs using Tukey_Kramer HSD for DFR_Average

| System | Measure | Aspect versus Hierarchical | | Aspect versus Flat | |
|---|---|---|---|---|---|
| | | Mean difference (Aspect−Hierarchical) | $p$ value | Mean difference (Aspect−Flat) | $p$ value |
| **ECS** | DFR_Average (Call) | 0.01 | 0.99 | 0.33 | **0.04** |
| | DFR_Average | −0.0002 | 0.99 | 0.24 | **0.04** |
| **VCS** | DFR_Average (DnD) | 0.37 | **0.02** | 0.34 | **0.02** |
| | DFR_Average (AQ) | 0.46 | **0.0002** | −0.26 | 0.06 |
| | DFR_Average | 0.37 | **0.0002** | 0.13 | 0.18 |

**Table 19** Two tailed Fisher's exact test for DFR binary measures

| System | Measure | Aspect versus Hierarchical | | Aspect versus Flat | |
|---|---|---|---|---|---|
| | | Difference in proportion (Aspect−Hierarchical) | *p* value | Difference in proportion (Aspect−Flat) | *p* value |
| ECS | DFR_Binary (Stop) | −0.01 | 1 | −0.28 | 0.08 |
| | DFR_Binary_Max (Stop) | −0.01 | 1 | 0.64 | **0.0001** |
| | DFR_Binary (Call) | 0.07 | 0.58 | 0.0007 | 1 |
| | DFR_Binary_Max (Call) | 0.07 | 0.73 | 0.33 | 0.06 |
| | DFR_Binary | 0.02 | 1 | −0.13 | 0.15 |
| | DFR_Binary_Max | 0.02 | 1 | 0.49 | **0.0001** |
| VCS | DFR_Binary (DnD) | 0.31 | 0.21 | 0.44 | **0.02** |
| | DFR_Binary_Max (DnD) | 0.28 | 0.11 | 0.28 | **0.04** |
| | DFR_Binary (Standby) | – | – | −0.45 | **0.03** |
| | DFR_Binary_Max (Standby) | – | – | 0.32 | 0.12 |
| | DFR_Binary (AQ) | 0.50 | **0.002** | 0.3 | 0.05 |
| | DFR_Binary_Max (AQ) | 0.33 | **0.04** | 0.33 | 0.06 |
| | DFR_Binary | 0.28 | **0.03** | 0.15 | 0.22 |
| | DFR_Binary_Max | 0.33 | **0.001** | 0.30 | **0.0005** |

**Table 20** Results of matched pairs for VCS for various DFR measures at $\alpha = 0.05$

| Dependent variable | Pair of approaches | Mean difference | Test | *p* value |
|---|---|---|---|---|
| DFR_Average | Aspect–Hierarchical | 0.31 | *t* test | **0.004** |
| DFR_Average | Aspect–Flat | 0.14 | | 0.11 |
| DFR_Binary | Aspect–Hierarchical | 0.33 | McNemar's test | 0.832 |
| DFR_Binary | Aspect–Flat | 0.15 | | **0.03** |
| DFR_Binary_Max | Aspect–Hierarchical | 0.33 | | **2.98e−08** |
| DFR_Binary_Max | Aspect–Flat | 0.31 | | **4.17e−07** |

**Table 21** Summary of statistically significant results

| Dependent variable | Approach pair | Round 1 | Round 2 | |
|---|---|---|---|---|
| | | ECS | VCS (Tukey_Kramer HSD) | VCS (Matched Pairs *t* test) |
| | | Crosscutting behavior (X) | Crosscutting behavior (X) | Crosscutting behavior (X) |
| DFR_Average | A > X | Call (F), All (F) | DnD (H), DnD (F), AQ (H), All (H) | All (H) |
| | X > A | – | – | – |
| DFR_Binary | A > X | – | DnD (F), Standby (F), AQ (H), All (H) | All (F) |
| | X > A | – | – | – |
| DFR_Binary_Max | A > X | Stop (F), All (F) | DnD (F), AQ (H), All (H), All (F) | All (H), All (F) |
| | X > A | – | – | – |

*X* either H (Hierarchical) or F (Flat), *A* aspect, *H* hierarchical, *F* flat, '–' non-significant results

the fourth column shows the results of Tukey_Kramer HSD for VCS. In Table 21, a "–" indicates non-significant results.

From Table 21, we can see that overall *Aspect* significantly performed better than *Flat* in terms of *DFR_Average* and *DFR_Binary_Max*, but there are no significant differences between *Aspect* and *Hierarchical*. When compared to the results of DIRs from Round 1, the results are in favor of AspectSM because the students gained experience with AspectSM while identifying defects. In addition, due to the lower complexity of aspect state machines (Table 2), it was easier for the subjects to fix the defects. For Round 2, in the case of VCS, *Aspect* is overall significantly better than *Hierarchical* and *Flat* as it can be seen from the results of Tukey_Kramer HSD for all three DFR variables in Table 21. The results of the matched pairs *t* test for *DFR_Average* and the McNemar's test for the two binary dependent variables

**Table 22** Estimation of the effect size corresponding to 80 % power

| System | Measure (approaches) | *p* value | Observed effect size | Minimum effect size | Average | Minimum effect size/average |
|---|---|---|---|---|---|---|
| ECS | DFR_ Average (A vs. H vs. F) | 0.48 | 0.07 | 0.20 | 0.60 | 0.34 |
| | DFR_ Average (A vs. H) | 0.99 | 0.001 | 0.15 | 0.65 | 0.23 |
| | DFR_Binary(A vs. H) | 1 | 0.01 | 0.15 | 0.78 | 0.19 |
| | DFR_Binary (A vs. F) | 0.15 | 0.06 | 0.12 | 0.85 | 0.14 |
| | DFR_Binary_Max (A vs. H) | 1 | 0.01 | 0.18 | 0.54 | 0.33 |
| VCS | DFR_ Average (A vs. H vs. F) | 0.18 | 0.06 | 0.12 | 0.45 | 0.27 |
| | DFR_Binary (A vs. H vs. F) | 0.22 | 0.07 | 0.15 | 0.66 | 0.22 |

*A* aspect, *H* hierarchical, *F* flat

**Table 23** Descriptive statistics for SCQ

| Experiment | System | Crosscutting behavior | Approach | | |
|---|---|---|---|---|---|
| | | | Aspect | Hierarchical | Flat |
| Experiment | ECS | Call and Stop | 6.38 | 8.56 | 4.50 |
| Replication | ECS | Call and Stop | 5.52 | 7.06 | 5.33 |
| | VCS | DnD, Standby, and AQ | 6.92 | 6.6 | 6.4 |

yielded consistent results. Similar to defect identification, plausible explanation is that, when compared with flat and hierarchical state machines, aspect state machines are much less complex in terms of number of states and transitions (Table 2); therefore, it is expected to be much easier to fix defects in aspect state machines.

To further investigate non-significant results, we performed power analysis, whose results are summarized in Table 22. Note that we did so only for those cases where the results are not even significant when observations are combined for all crosscutting behaviors. In case of *DFR_Average* (*Call* and *Stop*) regarding three approaches, the results of the power analysis shows an estimated minimum effect size of 0.20 (60 % of average) to achieve 80 % power in Table 22. The observed effect size is 0.07, which is much lower than the estimated effect size (0.20) thus explaining lack of significance. Given that 60 % is a large threshold, this suggests that we need to collect more observations to draw conclusions with confidence regarding which approach (*Aspect* or *Flat* or *Hierarchical*) is better in terms of *DFR_Average* for *Call* and *Stop*. Similar results are obtained for other dependent variables for which the results were not significant in Table 22.

## 4.3 Results and analysis for comprehensibility (RQ3)

In this section, we present results for answering comprehension questionnaire (RQ3) based on the *SCQ* dependent variable (Sect. 3.5).

### *4.3.1 Results for the initial experiment*

The descriptive statistics for *SCQ* are presented in Table 23. We observed that *Hierarchical* yields higher correctness than *Aspect* and *Flat*. More specifically, *Hierarchical* performed 21.8 and 40 % better than *Aspect* and *Flat*, respectively. We checked the significance of the results by applying one-way ANOVA to *SCQ* (Table 24), which shows significant differences between the approaches as the *p* value is 0.002. Since one-way ANOVA results are significant, we performed a pair-wise comparison of the distributions obtained for the three state machines using Tukey_Kramer HSD. The results showed that differences are not significant between *Aspect* vs. *Hierarchical* and *Aspect* vs. *Flat*. However, the results are significant between *Hierarchical* and *Flat*, but we do not report them here since this is not the focus of our study.

### *4.3.2 Results for the replication*

For the replication with ECS, we observed that *Hierarchical* yields higher comprehensibility (*SCQ*) than *Aspect* and *Flat* as it can be seen from the results reported in Table 23. For VCS, we observed that *Aspect* scored on average 6.92, which is higher than *Hierarchical* (6.6) and *Flat* (6.4) in Table 23. A one-way ANOVA with *SCQ* for ECS is reported in Table 24 and shows a significant difference. However, the results of a pair-wise comparison using Tukey_Kramer HSD shows, once again, significant differences only between *Hierarchical* and *Flat*. For VCS, the result of one-way ANOVA on *SCQ* showed no significant differences (Table 24).

**Table 24** Results of One-way ANOVA for SCQ at $\alpha = 0.05$

| Experiment | System | Crosscutting behavior | $p$ value |
|---|---|---|---|
| Experiment | ECS | Stop and Call | **0.002** |
| Replication | ECS | Stop and Call | **0.02** |
| | VCS | DnD, Standby, and AQ | 0.79 |

#### 4.3.3 Discussion

Overall, the differences between *Aspect* vs. *Hierarchical* and *Aspect* vs. *Flat* are not significant. One plausible explanation is that for *Aspect* the subjects needed to carefully read and understand *Pointcut* specifications in the *Aspect* state machines. With more training and practice on AspectSM, subjects would be expected to gain better comprehension of aspect state machines as compared with flat and hierarchical state machines, for which they had more prior experience.

The power analysis results for *SCQ* for the initial experiment, when comparing *Aspect vs. Hierarchical* and *Aspect vs Flat,* revealed that we need minimum effect sizes of 1.27 (17 % of average) and 1.65 (30 % of average), respectively, to achieve 80 % power (Table 25). These effect sizes are larger than the observed effect sizes, i.e., 1.08 and 0.94, thus explaining lack of significance. For VCS, power analysis revealed similar results, where we need minimum effect size of 1.10 (17 % of the average) to achieve 80 % of power as reported in Table 25. The minimum effect size, i.e., 1.10 (score out of 10) is much larger than the observed effect size (0.23). Thus, overall, if we want to investigate effects below the minimum thresholds mentioned above, the results of power analysis suggest that we need to collect more observations either by increasing the number of subjects and/or adding more case studies with crosscutting behaviors.

### 4.4 Results and analysis for effort (RQ4)

In this section, we present results for effort (RQ4) based on the *Effort* dependent variable (Sect. 3.5). Recall from Sect. 3.8 that the effort was measured only for the initial experiment and thus in this section we only present results and analysis for the initial experiment.

#### 4.4.1 Results

From Table 26, we can observe that in *Task 1*, the subjects took approximately 34 min on average for *Hierarchical* to identify defects. However, both *Aspect* and *Flat* took the same average time to complete the task: 32 min. *Task 2* took 3 and 7 min less for *Aspect* than for *Hierarchical* and *Flat* to identify defects. For answering the comprehension questionnaire (*Task 3*), the subjects took 10 and 5 min more for *Hierarchical* than *Aspect* and *Flat,* respectively. In summary, there is no practically significant time difference across the three state machines (Table 26).

As discussed in Sect. 3.9.1, we applied the one-way ANOVA test to assess the statistical significance of differences for *Effort* (for each task) distributions across the three approaches. Table 27 shows the results of the test, where significant differences were observed for the *Effort* of Task 3. Since one-way ANOVA results were significant, we performed a pair-wise comparison of the distributions obtained for the three state machines using Tukey_Kramer HSD [30]. The results are presented in Table 28, where *Hierarchical* took significantly more time than *Aspect* ($p$ value $= 0.01$), whereas *Aspect* took less time than *Flat*, though the latter is not significant ($p$value $= 0.39$).

#### 4.4.2 Discussion

There were no significant differences in effort between any pair of approaches for defect identification (*Task1* and *Task2*). This means that the effort spent for identifying defects across the three state machines is roughly the same. Regarding *Task 3* (i.e., answering the comprehension questionnaire), we only observed significant differences for *Effort* between the *Aspect* and *Hierarchical* groups, where the hierarchical group took significantly more time than the *Aspect* group (Table 28). Between *Aspect* and *Flat*, for Task 3, we did not observe significant differences in terms of *Effort*.

The power analysis in Table 29 shows that the minimum effect size corresponding to 80 % power is 3.62 min (i.e., 16 % of the average effort for the combined groups). The observed effect size is 2.14 min, thus explaining lack of significance. Drawing reliable conclusions for effect sizes below 3.62 min would require larger sample sizes. However, note that the

**Table 25** Estimation of the effect size corresponding to 80 % power for SCQ

| Experiment | System | Pair of approaches | $p$ value | Observed effect size | Minimum effect size | Average (score out of 10) | Minimum effect size/average |
|---|---|---|---|---|---|---|---|
| Experiment | ECS | Aspect versus Hierarchical | 0.09 | 1.08 | 1.27 | 7.41 | 0.17 |
| Replication | ECS | Aspect versus Flat | 0.15 | 0.94 | 1.65 | 5.5 | 0.3 |
| | VCS | Aspect versus Hierarchical versus Flat | 0.79 | 0.23 | 1.10 | 6.64 | 0.17 |

**Table 26** Mean values in minutes for effort across three tasks

| Measure | Aspect | Hierarchical | Flat |
|---|---|---|---|
| Effort (Task 1) | 32 | 34 | 32 |
| Effort (Task 2) | 12 | 15 | 19 |
| Effort (Task 3) | 20 | 30 | 25 |

**Table 27** Results for one-way ANOVA test for Effort using Tukey_Kramer HSD

| Measure | Effort (Task1) | Effort (Task2) | Effort (Task3) |
|---|---|---|---|
| *p* value | 0.86 | 0.09 | **0.02** |

difference between the two averages, i.e., 2.14 and 3.62 min, is small and therefore practically negligible.

### 4.5 Concluding remarks

Based on the above results and discussions, we suggest that aspect state machines should be used to model crosscutting behavior, but one should always use, when applicable, hierarchical state machines features within aspect state machines to further improve their comprehensibility. There are cases in which hierarchical state machines (submachines) are not applicable and aspect state machines are then the only option. For example, separating out constraints modeling non-functional properties (e.g., video or audio quality) from state invariants is not possible using hierarchical state machines without introducing accidental complexity and redundancy as we demonstrated in [10]. Easier identification/fixing of defects in aspect state machines also implies that it is easier to ensure their conformance to specifications.

## 5 Threat to validity

Below, we discuss the threats to validity of our controlled experiments based on the guidelines presented in [27].

### 5.1 Conclusion validity threats

Conclusion validity threats are concerned with factors that can influence the conclusion that can be drawn from the results of the experiments. 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. For the initial experiment, we performed a two-round experiment to maximize the number of observations within time constraints. However, the lack of significance of certain differences (e.g., the difference in *SCQ* for *Aspect* vs. *Hierarchical* and *Aspect* vs. *Flat*, effort for answering the comprehension questionnaire (*Aspect* vs. *Flat*), and *DIR_Binary* for *Aspect* vs *Flat*) may be due to low statistical power if actual effect sizes are below a certain threshold (Sect. 4.5). Studying the presence of smaller effect sizes requires replicating the experiment and collecting additional data points. Due to this reason, we replicated the experiment with an additional industrial case study including three crosscutting behaviors (Sect. 3.3.1) and with more subjects (Sect. 3.2) to increase the sample sizes and thereby the power of statistical tests. Statistical conclusions were drawn by applying appropriate statistical tests based on a careful analysis of their assumptions (Sect. 3.9).

### 5.2 Internal validity threats

Internal validity threats exist when the outcome of results are influenced by external factors and are not necessarily due to the application of the treatment being studied. Through our experiment design (between-subjects design) for the initial experiment and Round 1 of the replication, we have tried to minimize the chances of other factors being confounded with our primary independent variable: the use of aspect state machines. We avoided any biased assignment of subjects to groups by using blocking based on assignment marks.

In Round 2 of the replication, regarding identification and fixing defects, we used a within-subjects design and matched pairs*analysis*. The strength of this design is that the variation due to differences in subjects is eliminated as each subject acts as its own control. A within-subjects design may, however, be subject to learning effects, for example, due to using the same material for various tasks (e.g., defect identification and defect correction) that could result in improved performance from one task to the next. To counterbalance such effects, we rotated our groups to each crosscutting behavior for each task (e.g., defect identification) as we discussed in Sect. 3.4.2. In the initial experiment, we gave the subjects as much time as they wished to use for each task, though there was a time limit by which they had to finish all the tasks. No time differences were observed across modeling approaches. In the replication, we gave the subjects a fixed amount of time for each activity. Such an approach only enables, however, an investigation of the effect of the modeling approaches in terms of effectiveness.

### 5.3 Construct validity threats

Construct validity threats are related to the degree to which the construct being studied (i.e., readability in our context) is affected by experiment settings, which include the coverage of modeling elements of three types of state machines: the coverage of different types of defects, and the coverage of features of aspect-orientation. Regarding readability based on defect identification rates, due to time and resource

**Table 28** Comparisons of all pairs for Effort (Task 3) using Tukey_Kramer HSD

| Aspect versus Hierarchical | | Aspect versus Flat | |
|---|---|---|---|
| Mean difference (Aspect−Hierarchical) | *p* value | Mean difference (Aspect − Flat) | *p* value |
| −9.31 | **0.01** | −4.3 | 0.39 |

**Table 29** Estimation of the effect size corresponding to 80 % power

| Measure | *p* value | Observed effect size | Minimum effect size | Average (min) | Minimum effect size/average |
|---|---|---|---|---|---|
| Effort (Task 3 for Aspect versus Flat) | 0.39 | 2.13 | 3.62 | 22.43 | 16 % |

constraints, we could not seed all types of defects in the defect classification (Sect. 3.3.2). It is also not practically feasible to devise case studies containing all types of defects from the defect classification. Nevertheless, we tried to maximize defect classification coverage based on the available case studies and seeded defects of types MT, IT, MS, and IS to compute defect identification rates. Another threat of construct validity is that we were not able to investigate all features of aspect-orientation (e.g., all types of basic advices) due to the nature of the crosscutting behaviors in our case studies.

In terms of the coverage of UML state machine modeling elements, we cover submachine states, composite states, orthogonal states, signal events, call events, change events, time events, entry and exit points, history states, effects, and guards. In our experiments, we did not study the impact of interactions between aspect state machines. However, it is important to recall that the experiments presented in this paper are, to the best of our knowledge, the very first experiments with AOM, which focus only on studying the readability of models. In the future, we plan to conduct more experiments to study the impact of interactions between aspect state machines.

### 5.4 External validity threats

This is typically the most common threat in controlled experiments. Due to time constraints, case studies and tasks are usually small, and this often tends to minimize the differences among treatments. As we see in Table 2 for ECS, for crosscutting behavior *Call*, the flat state machine has 15 states and 27 transitions. Similarly, for the *Stop* crosscutting behavior, we have 13 states and 25 transitions. Such numbers are at least representatives of the state machines of classes and small components. In addition, we also replicated the experiment on an industrial case study with three crosscutting behaviors and more students to further reduce external validity threats. 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. It is worth noting that we replicated the experiment in a different geographical area and education system to reduce external validity threats. One may also question the use of students as subjects for the experiment. Note that many practitioners have very little knowledge of AOP or AOM in general and hence require significant training and cost to teach them AOM. Due to this reason, we chose a group of experienced graduate students with a suitable educational background (Sect. 3.2). In addition, some studies in [34–36] reported on the performance of trained software engineering students for various tasks when compared with professional developers. These differences turned out not to be statistically significant when compared with junior and intermediate developers, thus suggesting 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

In this section, we compare our controlled experiment with the experiments reported in the literature of Aspect-Orientation (AO) in Sect. 6.1 and experiments related to state machines in the non-AO literature in Sect. 6.2.

### 6.1 Comparison with experiments reported in AO

Most experimentation in Aspect-Oriented Software Development (AOSD) has been conducted to evaluate aspect-oriented programming when compared with object-oriented programming in terms of development time, errors in development, and performing maintenance tasks. An initial search on the IEEE, ACM, Science Direct, Wiley Interscience, and Springer digital libraries yielded 517 papers; however, none of them reported any controlled experiment to evaluate AOM approaches. A controlled experiment [37] 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 [38], 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 [39] 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, Aspect-oriented Programming (AOP) seems to be more beneficial when the crosscutting concern is more separable from the core behavior.

An exploratory study is reported in [40] 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 Object-oriented Programming (OOP), but there were no statistically significant results observed. Another exploratory study is reported in [41] 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 [42], where two software development processes based on a same aspect modeling approach (i.e., the Theme approach [43]) 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 AO code in AspectJ from Theme AO 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 [44], which aims to assess if aspects can help reduce 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 experiments are different from the above experiments from several perspectives. First, our controlled experiments focused on the design phase of the software development life cycle and AOM. Most of the experiments in the literature have focused on comparing AOP with OOP. We evaluated the "readability" (i.e., defect identification, defect fixing, and answering comprehension) of crosscutting behaviors modeled as aspect state machines as compared with directly modeling them in UML state machines. We further compared the effort for defect identification and answering comprehension for the experiment. Apart from these differences, we observed results in our experiments to be consistent with some of the results observed in the literature. For instance, similar to the results on development time using AspectJ reported in [41], we did not observe any reduced effort in inspecting state machines developed using our AspectSM approach. Also, similar to results reported in [38] and [39], where they observed inconsistent results for different measures corresponding to different program development and maintenance activities, our results also differed for defect identification/fixing rates and responses to the comprehension questionnaire.

### 6.2 Comparison with non-AO experiments for state machines

A series of experiments is presented in [45] that compares a state-based testing strategy (All round-trip paths coverage) with a code-based structural coverage test strategy (edge coverage) in terms of fault detection. Another series of experiments is reported in [46], which compares All round-trip paths coverage with a black-box functional testing technique called Category Partition in terms of fault detection. Our experiment is different from these experiment since we study the readability of three types of state machines developed for robustness testing and do not focus on studying the effectiveness of test strategies defined on state machines.

Another experiment is reported in [47], which studies the effect of six factors (e.g., representation of state machines, hierarchies and triggering conditions) on the readability of three types of state machine specifications: tabular, textual, and graphical. The experiment was conducted with graduate students and questionnaires were designed to ask students to answer different subjective and objective questions. Interesting results were obtained. For example, in terms of representation, most of the students found graphical and tabular representations more readable than textual ones. For hierarchies, all subjects agreed that hierarchies increase readability. In our experiment, we also measured readability using a questionnaire with subjective questions; however, the following differences can be noted: (1) We evaluate the readability of *crosscutting behaviors* when modeled using aspect, hierarchical, and flat state machines; (2) We evaluate readability by asking subjects to identify/fix seeded defects in crosscutting

behaviors; and (3) We evaluate the readability of aspect state machines, which has not been studied before.

## 7 Conclusion and future work

Aspect-oriented Modeling (AOM) is a very active field of research and can potentially yield several benefits while modeling systems, including enhanced separation of concerns, improved readability, easier model evolution, increased reusability, and reduced modeling effort. However, to authors' knowledge, there is no reported empirical evidence regarding such benefits.

This paper reports the results of the first two controlled experiments in the literature to report on the evaluation of AOM, and more precisely whether AOM can help improve the "readability" of UML state machines in terms of design defect identification, defect fixing, comprehension, and inspection effort. The specific AOM approach under evaluation is a recently published UML profile (AspectSM), which was specifically designed to model crosscutting behavior (e.g., robustness behavior) using standard UML 2 state machines with a lightweight extension for aspect-oriented features. The AspectSM profile has been previously applied to an industrial case study for automated, state-based robustness testing. The readability of state machines modeling crosscutting behavior using AspectSM (aspect state machines) is compared with standard UML 2 state machines using advanced features such as hierarchy and concurrency (hierarchical state machines) and without hierarchical features (flat state machines).

Results show that the defect identification and defect fixing rates of aspect state machines are significantly higher than the ones for the hierarchical and flat state machines. For instance, for the industrial case study in the replication, aspect state machines show, on average, increases of 28 and 19 % in defect identification rates when compared with hierarchical and flat state machines, respectively. This is most likely due to the fact that aspect state machines are less complex than hierarchical and flat state machines in terms of modeling elements such as states and transitions. But on the other hand, aspect state machines can be potentially difficult to comprehend in terms of mentally processing how an aspect is woven into its base state machine. This may explain why, based on subjects' responses to a comprehension questionnaire, results show that the subjects that were given hierarchical state machines outperformed the ones that were assigned aspect state machines, though that difference was not statistically significant. No significant difference in effort was observed between any types of state machines in both defect identification and comprehension. Based on the results above, our practical recommendation is to model crosscutting behaviors using aspect state machines in combination with hierarchical/concurrent features of UML state machines, where applicable, in order to improve the overall readability of crosscutting behaviors.

In the future, we are planning to replicate the experiment to study the readability of aspect state machines in the presence of interactions between aspects as well as compare the understandability, modeling effort, and quality of aspect state machines with flat and hierarchical state machines.

**Acknowledgments** Lionel Briand was in part supported by a PEARL grant from the Fonds National de la Recherche, Luxembourg.

## Appendix A: Models for elevator control system (ECS)

In this Appendix, we provide the description and models for one of the case studies that we used in the replication: the Elevator Control System (ECS). Note that we provide this information only for one crosscutting behavior *EmergencyCall* (*Call*) of ECS, to provide an idea of how the models developed using different modeling approaches look like and to demonstrate that these models are semantically equivalent. The crosscutting behavior *EmergencyCall* is an important robustness behavior of an elevator. Whenever the elevator is operating, an emergency call can be made at any time. When a call is made, it is dialed to the control room and if the call is successful, the person in the elevator can talk to a person in the control room. When the person in the elevator is done talking, he/she can disconnect the call from the control room. Notice that all these operations related to the emergency call are performed concurrently to the operation of the elevator. All the diagrams used in the experiments and shown in Figs. 3, 4, and 5 were drawn using IBM Rational Software Architect (RSA) [48] and therefore, the diagrams conform to the RSA graphical notations. In addition, the experiment participants were trained to understand these graphical notations.

The base state machine of ECS is shown in Fig. 3, which controls movements of an elevator in a building. The specification of the elevator is obtained from [11]. From the *Idle* state, call the *RequestUp* trigger (method of the ECS class), and then the elevator goes to the *DoorClosingToMoveUp* state representing the behavior of the system when the door is closing and moving up. Similarly, from the *Idle* state, when the *RequestDown* trigger is fired, the elevator goes to the *DoorClosingToMoveDown* state. From *ElevatorAtFloor*, if no trigger is fired within 5 s, the elevator state machine transits to the *Idle* state (modeled as a time event). Similarly, different states and transitions are modeled in the base state machine.

Aspect state machine for the *EmergencyCall* aspect is shown in Fig. 4. Stereotype ≪*Aspect*≫ is applied to the *EmergencyCall* state machine, indicating that it is an aspect state machine. The attributes for ≪*Aspect*≫ contain the following information: the name of the aspect state machine and the name of the base state machine (*ElevatorControl*

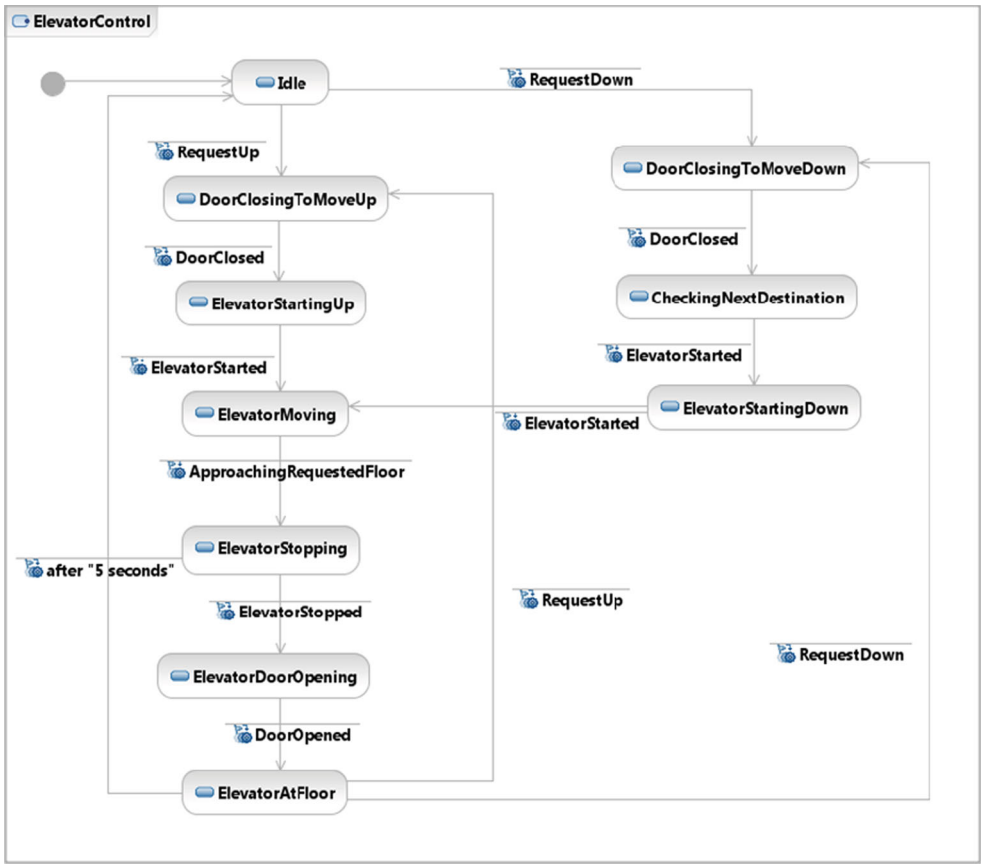


**Fig. 3** Base state machine for ECS

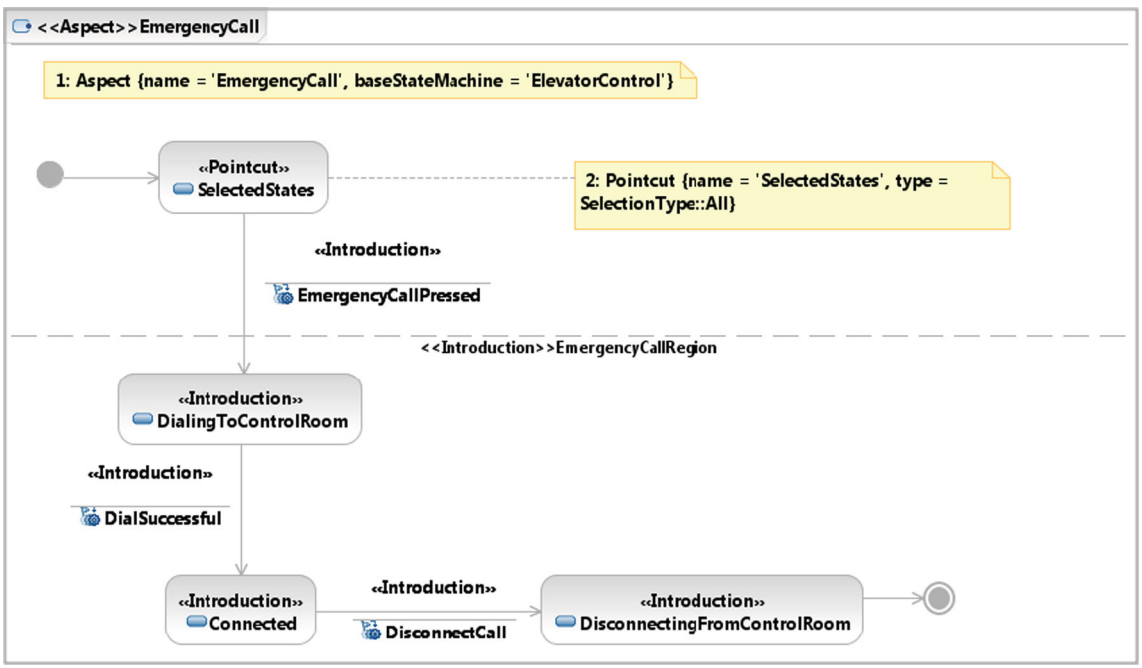


**Fig. 4** Aspect state machine for *EmergencyCall*

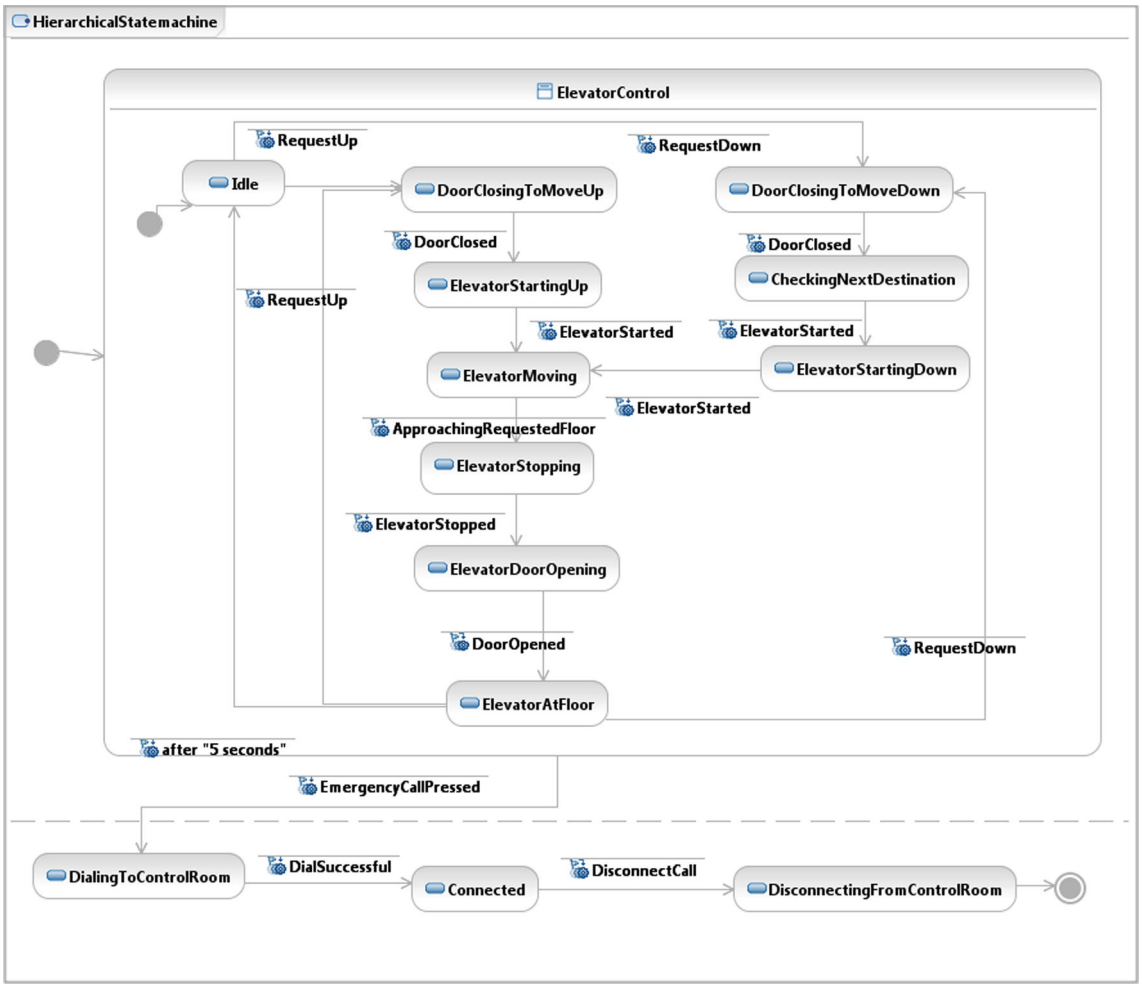


**Fig. 5** *EmergencyCall* modeled with the Hierarchical approach

in this example). State *SelectedStates* is stereotyped as ≪*Pointcut*≫, which shows that this state selects states from the base state machine. Stereotype ≪*Pointcut*≫ has two attributes: the name of the pointcut (*SelectedStates* in this case) and its type (*SelectionType:All* meaning that it selects all states of the base state machine). New transitions are added in the base state machine, with trigger named as *EmergencyCallPressed* stereotyped with ≪*Introduction*≫, from all the states of *ElevatorControl* to a newly introduced state named as *DialingToControlRoom* stereotyped with ≪*Introduction*≫. A new transition with trigger *DialSuccessful* is added from *DialingToControlRoom* to a newly introduced state *Connected*. Finally, from *Connected*, a newly introduced transition with trigger *DisconnectCall* is added to a newly introduced state *DisconnectingFromControlRoom*. Note that in Fig. 4, a new region is introduced: *EmergencyCall*, which is orthogonal to the normal operation of *ElevatorControl*.

The *EmergencyCall* behavior modeled using the Hierarchical approach is shown in Fig. 5. The behavior of *ElevatorControl* is modeled in a composite state (i.e., *ElevatorControl*) in Fig. 5. From the boundary of the *ElevatorControl* composite state, a transition with trigger *EmergencyCallPressed* goes to *DialingToControlRoom*. This means that from any of the states in *ElevatorControl*, whenever *EmergencyCallPressed* is pressed, the emergency call is made. An equivalent design of *EmergencyCall* using flat state machines is shown in Fig. 6.

## Appendix B: Comprehension questionnaire for replication

1. Explain the possible subsequent scenario when *Saturn* is in a videoconference with three endpoints and audio quality is within the allowed threshold value?

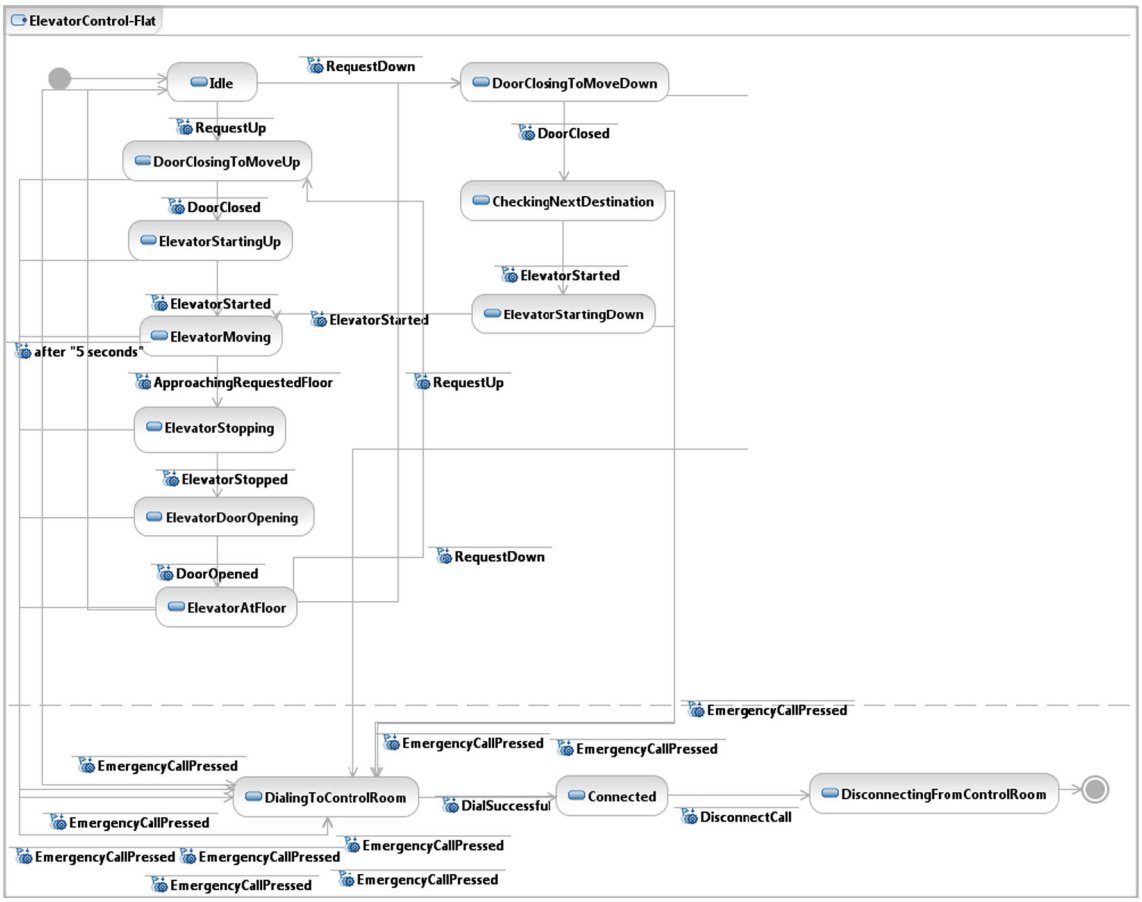


**Fig. 6** *EmergencyCall* modeled with the Flat approach

2. Explain the possible subsequent scenario when *Saturn* is *Idle* and audio quality is greater than the allowed threshold?
3. Explain the possible subsequent scenario when *Saturn* could not recover audio quality and is in a videoconference with one endpoint?
4. Explain the possible subsequent scenario when *Saturn* is connected to one endpoint and *Do Not Disturb mode* is turned *On*?
5. Explain the possible subsequent scenario when *Saturn* is in *Do Not Disturb mode* and is connected to two endpoints and two more endpoints are dialing to *Saturn* at the same time?
6. Explain possible subsequent scenario when *Saturn* is in *Connected* state for *m* minutes?
7. Explain the possible subsequent scenario when *Saturn* is in *StandbyOn* and an endpoint dials to it?
8. Explain the possible subsequent scenario when *Saturn* is *Idle* and an endpoint dials to it with H254 videoconference protocol?
9. Explain the possible subsequent scenario when *Saturn* is in a videoconference with two endpoints with SIP protocol and a third endpoint dials to Saturn with H323 videoconference protocol?
10. Explain the scenario when *Saturn* is in a videoconference with six endpoints with SIP protocol and another endpoint dials to Saturn with H323 videoconference protocol?

## References

1. Ali, M.S., Babar, M.A., Chen, L., Stol, K.-J.: A systematic review of comparative evidence of aspect-oriented programming. Inf. Softw. Technol. **52**, 871–887 (2010)
2. Chitchyan, R., Rashid, A., Sawyer, P., Bakker, J., Alarcon, M.P., Garcia, A., Tekinerdogan, B., Clarke, S., Jackson, A.: Survey of Aspect-Oriented Analysis and Design. Aspect-Oriented Software Engineering Special Interest Group, Lancaster University, AOSD-Europe-ULANC-9 (2005)
3. Filman, R.E., Elrad, T., Clarke, S., Aksit, M.: Aspect-Oriented Software Development. Addison-Wesley Professional, Reading, MA (2004)
4. Yedduladoddi, R.: Aspect Oriented Software Development: An Approach to Composing UML Design Models. VDM Verlag Dr, Müller, Saarbrücken (2009)

5. IEEE Standard Dictionary of Measures of the Software Aspects of Dependability. IEEE Std 982.1-2005 (Revision of IEEE Std 982.1-1988), pp. 1–34 (2006)
6. Standard for Software Quality Characteristics, International Organization for Standardization, ISO-9126-32003
7. Software Assurance Standard, NASA Technical Standard, NASA-STD-8739.82005
8. Binder, R.V.: Testing object-oriented systems: models, patterns, and tools. Addison-Wesley Longman Publishing Co., Inc., Reading, MA (1999)
9. Drusinsky, D.: Modeling and Verification using UML Statecharts: A Working Guide to Reactive System Design, Runtime Monitoring and Execution-based Model Checking, 1st edn. Newnes (2006)
10. Ali, S., Briand, L.C., Hemmati, H.: Modeling robustness behavior using aspect-oriented modeling to support robustness testing of industrial systems. Syst. Softw. Model. (SOSYM) J. (Accepted for publication) (2011)
11. Gomaa, H.: Designing Concurrent, Distributed, and Real-Time Applications with UML. Addison-Wesley Professional, Reading, MA (2000)
12. Weigert, T., Reed, R.: Specifying Telecommunications Systems with UML. In: UML for real: design of embedded real-time systems. Kluwer Academic Publishers, Dordrecht, pp. 301–322 (2003)
13. SmartState. Available: http://www.smartstatestudio.com/. Accessed April (2012)
14. Utting, M., Legeard, B.: Practical Model-Based Testing: A Tools Approach. Morgan-Kaufmann, London (2007)
15. Cavarra, R., Crichton, C., Davies, J., Hartman, A., Mounier, L.: Using UML for Automatic Test Generation Presented at the International Symposium on Software Testing and Analysis (ISSTA '02) (2002)
16. Pender, T.: UML Bible. Wiley, Hoboken (2003)
17. Whittle, J., Moreira, A., AraújJ., Jayaraman, P., Elkhodary, A., Rabbi, R.: An Expressive Aspect Composition Language for UML State Diagrams. In: Engels, G., Opdyke, B., Schmidt, D.C., Weil, F. (eds.) Model Driven Engineering Languages and Systems (2007)
18. Tessier, F., Badri, L., Badri, M.: Towards a Formal Detection of Semantic Conflicts Between Aspects: A Model-Based Approach. Presented at the 5th Aspect-Oriented Modeling Workshop In Conjunction with UML 2004 (2004)
19. Ibm, OCL Parser. Available: http://www-01.ibm.com/software/awdtools/library/standards/ocl-download.html. Accessed April (2012)
20. Chiorean, D., Bortes, M., Corutiu, D., Botiza, C., Câu, A.: OCLE (V2.0 ed.). Available: http://lci.cs.ubbcluj.ro/ocle/. Accessed April (2012)
21. Egea, M.: EyeOCL Software. Available: http://maude.sip.ucm.es/eos/. Accessed April (2012)
22. Zhang, G.: Towards Aspect-Oriented State Machines, Presented at the 2nd Asian Workshop on Aspect-Oriented Software Development (AOASIA'06), Tokyo (2006)
23. Zhang, G., Hö, M.: HiLA: High-Level Aspects for UML-State Machines, Presented at the Proceedings of the 14th Workshop on Aspect-Oriented Modeling (AOM@MoDELS'09) (2009)
24. Zhang G., Hö M.M., Knapp, A.: Enhancing UML State Machines with Aspects. In: Proceedings of the 10th International Conference on Model Driven Engineering Languages and Systems (MoDELS) (2007)
25. Xu, D., Xu, W., Nygard, K.: A State-Based Approach to Testing Aspect-Oriented Programs. Presented at the 17th International Conference on Software Engineering and Knowledge Engineering, Taiwan (2005)
26. Laddad, R.: AspectJ in Action: Practical Aspect-Oriented Programming. Manning Publications, Greenwich (2003)
27. Wohlin, C., Runeson, P., Höst, M.: Experimentation in Software Engineering: An Introduction. Springer, Berlin (1999)
28. JMP. Available: http://www.jmp.com/. Accessed April (2012)
29. Shull, F., Singer, J., Sjø, D.I.K.: Guide to Advanced Empirical Software Engineering. Springer, Berlin (2008)
30. Sheskin, D.J.: Handbook of Parametric and Nonparametric Statistical Procedures. Chapman and Hall/CRC, London (2007)
31. McNemar's Test. Available: http://www.fon.hum.uva.nl/Service/Statistics/McNemars_test.html. Accessed April (2012)
32. Thomas, L.: Retrospective power analysis. Conserv. Biol. **11**, 276–280 (1997)
33. Dybå, T., Kampenes, V.B., Hannay, J.E., Sjøberg, D.I.K.: Systematic review: A systematic review of effect size in software engineering experiments. Inf. Softw. Technol. **49**, 1073–1086 (2007)
34. Höst, M., Regnell, B., Wohlin, C.: Using students as subjects—A comparative study of students and professionals in lead-time impact assessment. Empir. Softw. Eng. **5**, 201–214 (2000)
35. Arisholm, E., Sjoberg, D.I.K.: Evaluating the effect of a delegated versus centralized control style on the maintainability of object-oriented software. IEEE Trans. Softw. Eng. **30**, 521–534 (2004)
36. Holt, R.W., Boehm-Davis, D.A., Shultz, A.C.: Mental representations of programs for student and professional programmers. In: Gary, M.O., Sylvia, S., Elliot, S. (eds.) Empirical Studies of Programmers: Second Workshop, pp. 33–46. Ablex Publishing Corp, Norwood (1987)
37. Durr, P., Bergmans, L., Aksit, M.: A Controlled Experiment for the Assessment of Aspects-Tracing in an Industrial Context. University of Twente, CTIT, Enschede (2008)
38. Hanenberg, S., Kleinschmager, S., Josupeit-Walter, M.: Does Aspect-Oriented Programming Increase the Development Speed for Crosscutting Code? An Empirical Study, Presented at the 2009 3rd International Symposium on Empirical Software Engineering and Measurement (2009)
39. Walker, R.J., Baniassad, E.L.A., Murphy, G.C.: An Initial Assessment of Aspect-Oriented Programming, Presented at the 21st international Conference on Software Engineering. Los Angeles, California (1999)
40. Bartsch, M., Harrison, R.: An exploratory study of the effect of aspect-oriented programming on maintainability. Softw. Qual. Control **16**, 23–44 (2008)
41. 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., Maldonado, J.: An exploratory study of fault-proneness in evolving aspect-oriented programs, presented at the proceedings of the 32nd ACM/IEEE international conference on software engineering? vol. 1, Cape Town (2010)
42. Farias, K., Garcia, A., Whittle, J.: Assessing the impact of aspects on model composition effort, presented at the proceedings of the 9th international conference on aspect-oriented software development, Rennes and Saint-Malo
43. Carton, A., Driver, C., Jackson, A., Clarke, S.: Model-Driven Theme/UML. In: Shmuel, K., Harold, O., Robert, F., Jean-Marc, J., Quel, Z. (eds.) Transactions on Aspect-Oriented Software Development VI, pp. 238–266. Springer, Berlin (2009)
44. Hovsepyan, A., Scandariato, R., Baelen, S.V., Berbers, Y., Joosen, W.: From aspect-oriented models to aspect-oriented code?: the maintenance perspective, presented at the proceedings of the 9th international conference on aspect-oriented software development, Rennes and Saint-Malo
45. Mouchawrab, S., Briand, L.C., Labiche, Y., Penta, M.D.: Assessing, comparing, and combining state machine-based testing and structural testing: a series of experiments. IEEE Trans. Softw. Eng. **37**, 161–187 (2011)
46. Briand, L.C., Penta, M.D., Labiche, Y.: Assessing and improving state-based class testing: a series of experiments. IEEE Trans. Softw. Eng. **30**, 770–793 (2004)

47. Zimmerman, M.K., Lundqvist, K., Leveson, N.: Investigating the readability of state-based formal requirements specification languages, presented at the proceedings of the 24th international conference on software engineering. Orlando, Florida (2002)
48. IBM Rational Software Architect (RSA). Available: http://www.ibm.com/developerworks/rational/products/rsa/. Accessed April (2012)

## Author Biographies

**Shaukat Ali** is currently a research scientist in Certus Software Verification and Validation Center, Simula Research Laboratory, Norway. He has been affiliated to Simula Research Lab since 2007. He has been involved in many industrial and research projects related to Model-based Testing (MBT) and Empirical Software Engineering since 2003. He has experience of working in several industries and academic research groups in many countries including UK, Canada, Norway, and Pakistan.

**Tao Yue** received her BEng degree in the Department of Automation Science and Electrical Engineering, Beihang University, China, the M.A.Sc and PhD degrees in the Department of Systems and Computer Engineering at Carleton University, Ottawa, Canada in 2006 and 2010, respectively. She is now a senior research scientist of Simula Research Laboratory, Oslo, Norway, where she is leading the expertise area of Model Driven Engineering (MDE). She has around 16 years of experience of conducting industry-oriented research with a focus on MDE in various application domains such as Avionics, Maritime and Energy, and Communications in several countries including Canada, Norway, and China. Her main research area is software engineering, with specific interested in requirements engineering, model-based development, model-based configuration and variability modeling, and empirical software engineering.

**Lionel C. Briand** is professor and FNR PEARL chair in software verification and validation at the SnT centre for Security, Reliability, and Trust, University of Luxembourg. Lionel started his career as a software engineer in France (CS Communications & Systems) and has conducted applied research in collaboration with industry for more than 20 years. Until moving to Luxembourg in January 2012, he was heading the Certus center for software verification and validation at Simula Research Laboratory, where he was leading applied research projects in collaboration with industrial partners. Before that, he was on the faculty of the department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, where he was full professor and held the Canada Research Chair (Tier I) in Software Quality Engineering. He has also been the software quality engineering department head at the Fraunhofer Institute for Experimental Software Engineering, Germany, and worked as a research scientist for the Software Engineering Laboratory, a consortium of the NASA Goddard Space Flight Center, CSC, and the University of Maryland, USA. Lionel has been on the program, steering, or organization committees of many international, IEEE and ACM conferences. He is the coeditor-in-chief of Empirical Software Engineering (Springer) and is a member of the editorial boards of Systems and Software Modeling (Springer) and Software Testing, Verification, and Reliability (Wiley). He was on the board of IEEE Transactions on Software Engineering from 2000 to 2004. Lionel was elevated to the grade of IEEE Fellow for his work on the testing of object-oriented systems. He was recently granted the IEEE Computer Society Harlan Mills award for his work on model-based verification and testing. His research interests include: model-driven development, testing and verification, search-based software engineering, and empirical software engineering.