

# Comprehensibility of UML-based software product line specifications

## A controlled experiment

Iris Reinhartz-Berger · Arnon Sturm

© Springer Science+Business Media New York 2012  
**Editor:** Murray Wood

**Abstract** Software Product Line Engineering (SPLE) deals with developing artifacts that capture the common and variable aspects of software product families. Domain models are one kind of such artifacts. Being developed in early stages, domain models need to specify commonality and variability and guide the reuse of the artifacts in particular software products. Although different modeling methods have been proposed to manage and support these activities, the assessment of these methods is still in an inception stage. In this work, we examined the comprehensibility of domain models specified in ADOM, a UML-based SPLE method. In particular, we conducted a controlled experiment in which 116 undergraduate students were required to answer comprehension questions regarding a domain model that was equipped with explicit reuse guidance and/or variability specification. We found that explicit specification of reuse guidance within the domain model helped understand the model, whereas explicit specification of variability increased comprehensibility only to a limited extent. Explicit specification of both reuse guidance and variability often provided intermediate results, namely, results that were better than specification of variability without reuse guidance, but worse than specification of reuse guidance without variability. All these results were perceived in different UML diagram types, namely, use case, class, and sequence diagrams and for different commonality-, variability-, and reuse-related aspects.

**Keywords** Variability management · Software product line engineering · Domain models · Empirical evaluation · UML

---

I. Reinhartz-Berger (✉)  
Department of Information Systems, University of Haifa, Haifa 31905, Israel  
e-mail: iris@is.haifa.ac.il

A. Sturm  
Department of Information Systems Engineering, Ben-Gurion University of the Negev,  
Beer Sheva 84105, Israel  
e-mail: sturm@bgu.ac.il

## 1 Introduction

*Software product line engineering (SPLE)* deals with developing artifacts for families of software products, called *product lines*, and adjusting them to the particular needs of applications or systems, called *software products* or *products* for short (Clements and Northrop 2001). Two main activities are identified in SPLE (Pohl et al. 2005): *domain engineering*, during which families of applications or software products are analyzed, yielding sets of analysis, design, and implementation artifacts that can be (re)used by different family members, and *application engineering* in which these artifacts are customized and adapted to the needs of the particular applications. The outcomes of the domain engineering activity are *domain artifacts*, also called *core assets*, which are built to be used by more than one product in the product line. An important type of domain artifact is a *domain model*, which is an analysis or a design model of the product line. *Application artifacts*, also called *product artifacts*, are the outcomes of the application engineering activity and include adaptations of the domain artifacts to the specific requirements of the product in hand. The adaptation is done by selecting relevant domain artifacts, modifying and customizing them to satisfy the particular requirements, and extending the created application artifacts in a way that does not directly resemble the domain artifacts that spawned or generated them, e.g., by performing product-specific adaptations (Halmans et al. 2008).

In order to be reusable and suitable to a wide variety of products, domain artifacts in general, and domain models in particular, have to specify both the commonality that exists and the variability that is allowed among different products in the product line. While *commonality* refers to the kernel that is common to all (or most) members in a given product line, *variability* is defined as the ability of a domain artifact to be efficiently extended, changed, customized or configured for use in a particular application artifact (Coplien et al. 1998; Sinnema and Deelstra 2007). Svahnberg et al. (2005) claimed that “development of software product lines relies heavily on the use of variability to manage the differences between products”, but “variability is not trivial to manage”. Thus, methods and techniques for managing the allowed variability are desired (Sinnema and Deelstra 2008; Ziadi et al. 2004).

The usage of domain artifacts, and especially their variable parts, for carrying out application engineering activities often follows “reuse guidelines”, which are commonly called variability or variation mechanisms (Jacobson et al. 1997; Bachmann and Clements 2005), coding or implementing variability approaches (Anastasopoulos and Gracek 2001) or variability realization techniques (Svahnberg et al. 2005). These mechanisms (or techniques) introduce ways to utilize domain artifacts in order to create particular application artifacts. Becker (2003) classified these mechanisms into three classes: *selection*, in which a domain artifact is selected and adapted to the particular context, *generation*, in which the domain artifact is only used as a template for generating the application artifact, and *substitution*, which supports the creation of application artifacts by externally provided solutions (potentially outside the domain artifacts).

The development and utilization of domain artifacts are not simple tasks, as they require taking into consideration different, sometimes conflicting, concerns. Taking into account all these concerns may result in complicated artifacts that include numerous, potentially implicit, information which is relevant to different products in the line. Yet, these artifacts are expected to guide application engineering activities and, hence, need to be adaptable and usable for a variety of contexts. Thus, the comprehensibility of these artifacts is very important. The comprehensibility of domain models is even more important, as they are the basis of different early stage activities in both domain and application engineering.

Having comprehension errors or inaccuracies in these stages can lead to expensive and undesirable outcomes.

In this work, we examined the comprehensibility of domain models specified utilizing the standard modeling language – UML. UML and its profiles are largely used for specifying analysis and design artifacts in SPLE (Chen and Babar 2011). However, as far as we know, the comprehensibility of these profiles for conducting different domain and application engineering tasks has not yet been examined. Following Shanks et al. (2003), we refer here to comprehension in the context of problem-solving tasks, namely, tasks that assess the subjects' ability to use the knowledge represented in the schema, where the subjects are requested to determine whether and how certain information is available from the schema. In particular, we analyzed whether explicit modeling of reuse guidance and variability specification improves comprehensibility of domain models and if so to what extent. For this purpose, we conducted a controlled experiment with 116 third year information systems engineering undergraduate students who were expected to simulate our target audience – non-experienced software and information system developers. The subjects were required to answer comprehension questions about a domain model expressed in ADOM (Application-based DOMain Modeling), a UML-based method which supports the specification of both reuse guidance and variability (Reinhartz-Berger and Tsoury 2011a, b).

The remainder of this paper is organized as follows. Section 2 reviews related work and provides the background for the experiment. Section 3 elaborates on the method used in the experiment, namely, ADOM. Section 4 describes the experiment goals and settings, while Section 5 presents the experiment results. Section 6 discusses the experiment results and Section 7 elaborates on threats to validity. Lastly, Section 8 summarizes our work and discusses future research directions.

## 2 Background

In this section we first discuss the relatively low number of studies related to evaluation of SPLE techniques in general and variability management methods in particular (Section 2.1). We then discuss the various aids for specifying domain models (Section 2.2) and their uses in different UML-based methods (Section 2.3).

### 2.1 Evaluation of SPLE Methods

In a review of 97 papers describing variability management approaches in SPLE from 1990 to 2007, Chen and Babar (2011) conclude that the main corpus of these approaches refers to modeling and utilizes feature models (33 works) and/or UML and its extensions (25 works). Other directions, such as using natural languages, applying formal (mathematical) techniques, defining domain-specific languages, and using ontology-based techniques, are also explored, but the number of studies in each such category in the examined time period is very low. Another observation of Chen and Babar's work is that relatively little attention is allocated for the evaluation of these methods; most of them are evaluated using example applications (58.76 %), experience reports (17.53 %), and case studies (13.4 %), although a large majority of them "are quite amenable to empirical evaluation". Qualitative, textual evaluation through discussion on sets of criteria also exists, e.g., Djebbi and Salinesi (2006), Haugen et al. (2005), Matinlassi (2004), and Schmid et al. (2011). However, the main problem with this kind of evaluation is its subjectivity and opinion-oriented nature.

Only a few studies employ an empirical evaluation to analyze different aspects of SPLE, such as product derivation (Sinnema and Deelstra 2008), maintainability (Bagheri and Dasevic 2011), quality assurance (Denger and Kolb 2006), and architecture process activities (Ahmed and Capretz 2008). Regarding comprehensibility and understandability of domain artifacts, we found the work of Bagheri and Dasevic (2011), which assesses the maintainability of feature models in terms of three characteristics: (1) analyzability, which is “the capability of the conceptual model of a software system to be diagnosed for deficiency”; (2) changeability, which is “the possibility and ease of change in a model when modifications are necessary”; and (3) understandability, which is “the prospect and likelihood of the software system model to be understood and comprehended by its users or other model designers”. Bagheri and Dasevic used a 7-point scale to gather the subjective opinions of participants on the relevant characteristics, but did not objectively check the participants’ performance.

Reinhartz-Berger and Sturm (2008) empirically examined the comprehensibility of domain artifacts and their impact on the creation of application artifacts. They found out that the availability of domain artifacts helps achieve more complete application artifacts without reducing the comprehensibility of those artifacts. Yet, this experiment referred only to commonality aspects, neglecting variability- and reuse-related aspects that might reduce comprehensibility when introduced to domain models.

Reinhartz-Berger and Tsoury (2011a, b) empirically compared the comprehensibility of commonality- and variability-related aspects in domain models specified using two methods: CBFM (Czarnecki and Kim 2005), which is a feature-oriented method, and ADOM (Reinhartz-Berger and Sturm 2008, 2009), which is based on a UML profile. They conclude that developers may better understand the locations at which variability occurs and how to realize variability in ADOM than in CBFM. However, reuse-related issues were not examined in this particular work, nor were different combinations of the same specification aids. Furthermore, the number of subjects was low, preventing the possibility of performing a statistical analysis.

## 2.2 Specification Aids in SPLE

Many studies have previously explored SPLE concepts. Some of the studies are assets-derived, e.g., Becker (2003), Bachmann and Clements (2005), and Oliveira Junior et al. (2005). Others, such as Bachmann et al. (2004), Pohl et al. (2005), and Salicki and Farcet (2002), concentrate on variability concepts. The rest discuss different aspects of SPLE, including commonality and variability specification, e.g., the studies of Weiler (2003), Moon et al. (2005), and Haugen et al. (2008). Reviewing these studies, three sets of specification aids can be identified: commonality specification, variability specification, and reuse guidance.

*Commonality specification* aims at describing the elements in domain artifacts which may be either mandatory or optional in the product line as well as the dependencies among them that define the restrictions for selecting groups of optional elements (e.g., couples of domain elements and mutually exclusive domain elements). Typically, common elements are characterized by their multiplicity ranges, i.e., whether they are mandatory or optional and whether several clones of them may appear in a single product in the family.

*Variability specification* aids include *variation points*, which identify locations in the domain artifact where variable parts may occur and *variants*, which realize possible ways to create particular application artifacts at certain variation points. A variation point, which defines the insertion point for the variants and determines the characteristics (attributes) of

the variability, can be *open*, allowing the specification of new variants which are not associated to the variation point in the domain artifact, or *closed*, requiring that all possible variants will be listed in the domain artifact and associated with the variation point. Furthermore, a variation point may specify rules for selecting variants in the given point, e.g., one variant at most has to be selected in this point. The relationships between variation points and variants may take various forms (Clotet et al. 2008), the most important of which are: (1) inheritance, in which the different variants are specializations of the variation point, (2) instantiation, in which the variants are instances of the variation point, receiving different values for the parameters defined or induced by the variation point, and (3) general dependencies, which may take the form of realization, inclusion, etc.

*Reuse guidance* aids include different mechanisms which can be used for creating or producing application artifacts utilizing domain artifacts (Jacobson et al. 1997; Anastasopoulos and Gracek 2001; Bachmann and Clements 2005; Svahnberg et al. 2005). Examples of such mechanisms are: (1) configuration that enables choosing alternative functions and implementations, (2) inheritance that enables specializing and adding selected features, usually behavioral ones, while keeping others, (3) parameterization that allows changing the values of the attributes in an artifact, affecting its behavior according to the set values, and (4) template instantiation that enables type adaptation or selection of alternative pieces of code.

### 2.3 UML-Based Domain Modeling Methods

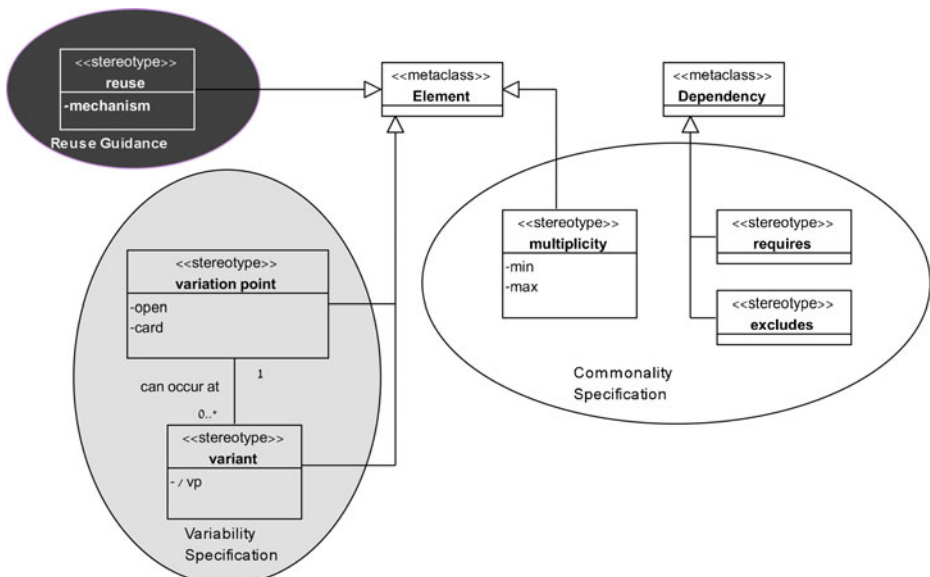
As mentioned earlier, UML-based domain modeling methods are the second largest group of SPLE and variability management methods (Chen and Babar 2011). Indeed, different UML-based SPLE methods have been presented in the last dozen years. Table 5 in Appendix A summarizes most of those published since 2000, along with the specification aids they support. It can be seen that most UML-based methods define stereotypes or profiles for specifying domain artifacts, or more accurately domain models, e.g., John and Muthig (2002), Halmans and Pohl (2003), Gomaa (2004), Ziadi and Jézéquel (2006), Sun et al. (2010), and Reinhartz-Berger and Tsoury (2011a, b), while some of them suggest small modifications to the UML metamodel, e.g., Maßen and Lichter (2002) and Bragança and Machado (2006). Commonality is usually specified using dedicated stereotypes for differentiating mandatory (sometimes called kernel) and optional elements. Some works explicitly specify variability using both «variation point» and «variant» stereotypes, including Ziadi et al. (2004), Sun et al. (2010), and Reinhartz-Berger and Tsoury (2011a, b). Others, including John and Muthig (2002), Halmans and Pohl (2003), and Gomaa (2004), specify only variation points or variants, while the other concept is implicitly specified from its relationships with the explicitly specified concept. Several works explicitly refer to dependencies between elements in the form of stereotypes, tagged values or constraints, e.g., Oliveira Junior et al. (2005), Korherr and List (2007), and Reinhartz-Berger and Tsoury (2011a, b). Only nine of the reviewed studies (see Appendix A) refer to reuse guidance. Yet, all these works, except for Reinhartz-Berger and Tsoury (2011a, b), introduce processes that guide application engineering but do not provide explicit specification aids related to reuse in the domain models. Furthermore, they refer to limited sets of mechanisms that can mainly be classified as selection or generation.

Based on Table 5 in Appendix A, we chose ADOM for our empirical study for the following reasons. Firstly, it supports the selection and addition of variants in certain variation points through the ‘card’ and ‘open’ tagged values, respectively, which will be explained and exemplified later. Secondly, it enables explicit specification of both variation

points and variants. Thirdly, it enables association of commonality- and reuse-related aspects for all domain elements and not just to variation points and variants. It further supports the specification of a wide range of reuse mechanisms through the «reuse» stereotype, which will be also explained and exemplified later. Lastly, it coherently and consistently incorporates commonality specification, variability specification, and reuse guidance aids in all UML diagram types.

### 3 The ADOM Method

The Application-based Domain Modeling (ADOM) method (Reinhartz-Berger and Sturm 2008, 2009; Reinhartz-Berger and Tsoury 2011a, b) is based on a UML profile comprising of six stereotypes: «multiplicity», «requires», «excludes», «variation\_point», «variant», and «reuse». Figure 1 depicts this profile, while the rest of the section explains these stereotypes and demonstrates them on a Check-In Check-Out (CICO) product line. The aim of the CICO product line is to develop applications for checking in and out items (e.g., hotel reservation systems, libraries, renting agencies, and version control services). Appendix B further provides the formalism of ADOM in the form of definitions and constraints among the different stereotypes. As formally explained there, ADOM can be used with different modeling languages and in particular with different UML diagram types. For convenience, we demonstrate ADOM in this section on class diagram fragments of the CICO product line. However, no significant differences exist between applying ADOM to class diagrams and applying it to other UML diagram types. Nevertheless, in order to cover other aspects of product lines, such as the functional and dynamic ones, Appendix C includes the full domain model of the CICO product line, which demonstrates applying ADOM to use case and sequence diagrams as well.



**Fig. 1** The UML profile of ADOM

### 3.1 Commonality Specification in ADOM

The main modeling aid for specifying commonality in ADOM is the «multiplicity» stereotype. This stereotype is used for specifying the range of elements in the application artifact, i.e., the application elements that can be classified as the same domain element. Two tagged values, min and max, are used for defining the lowest and upper-most boundaries of that range. For clarity purposes, four commonly used multiplicity groups are defined on top of this stereotype: «optional many», where min=0 and max=∞; «optional single», where min=0 and max=1; «mandatory many», where min=1 and max=∞; and «mandatory single», where min=max=1. Nevertheless, any multiplicity interval constraint can be specified using the general stereotype «multiplicity min=m<sub>1</sub> max=m<sub>2</sub>».

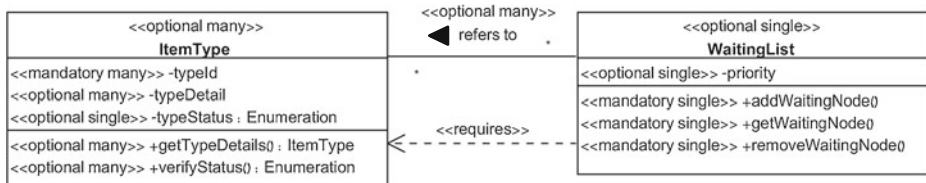
As an example of the «multiplicity» stereotype, consider two concepts in the CICO product line, described in Fig. 2: Item Type and Waiting List. Both concepts are optional but a specific software product may have several different Item Type classes while it must have at most one Waiting List class (note that the product may have several different actual waiting list objects, which are instances of the same Waiting List class).

Each Item Type class should have: at least one attribute specifying its type identifier, zero or more attributes describing different details on the type, at most one attribute specifying the type status, zero or more operations for getting the type details (or part of them), and zero or more operations for verifying different conditions on the type status. Some of the signatures of these attributes and operations are constrained in the model (see Fig. 2). For example, each status attribute should be of an enumeration type and getTypeDetails operations should return objects of type Item Type. Each Waiting List class, if it appears, includes at most one priority attribute (as the waiting list may be prioritized or not) and exactly one operation in each of the following three categories: adding, getting, and removing nodes to or from the waiting list. The classes Waiting List and Item Type may be connected with “refers to” associations, although these relations are not mandatory in the CICO product line, meaning, for example, that a waiting list can refer to an item rather than to an item type.

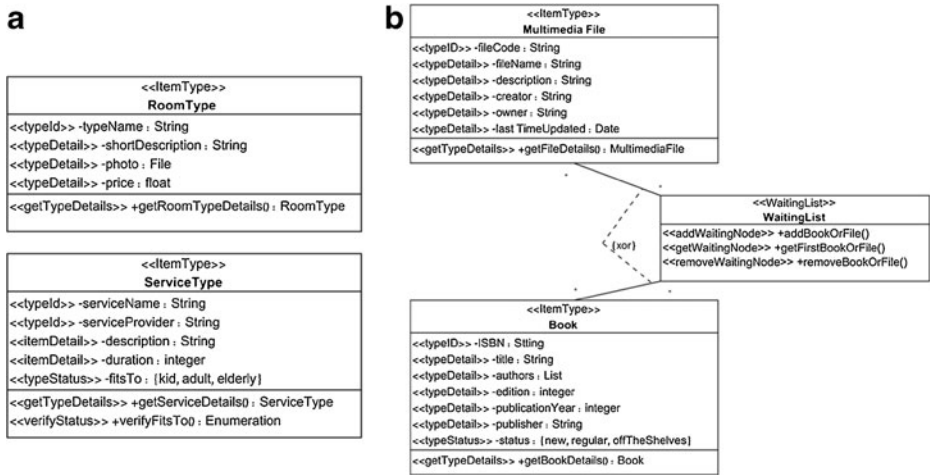
In order to constrain valid configurations, two stereotypes are defined in ADOM for determining dependencies between optional elements: «requires» and «excludes». A «requires» B, where A and B are two optional elements, implies that if A appears in a particular product, then B should appear too. In the same spirit, A «excludes» B implies that if A is included in a particular product, then B should not.

In Fig. 2, for example, if the Waiting List class is selected in a particular software product (at least once), then at least one Item Type class needs to be defined in the same product, although these classes may not be directly connected via associations.

In order to better explain the semantics of the aforementioned stereotypes, Fig. 3 presents two possible uses of the domain model depicted in Fig. 2. In the first case (a), the domain model is used for specifying a hotel reservation system in which two types of items can be reserved: rooms and services. This system does not support waiting lists at all. The second



**Fig. 2** Part of the CICO domain model, exemplifying the «multiplicity» and «requires» stereotypes



**Fig. 3** Exemplifying the semantics of the «multiplicity» and «requires» stereotypes in two applications: **a** a hotel reservation system and **(b)** a library system

case (b) refers to a library system which supports waiting lists of either books or multimedia files. Note that the domain elements appear as stereotypes in those models, providing anchors for verifying the domain constraints. Indeed, all the domain constraints discussed above are maintained in the two models, including the constraints on the classes' attributes, operations, and associations.

### 3.2 Variability Specification in ADOM

For specifying variability, each domain element may be defined as a variation point. This is done using the stereotype «*variation\_point*» in addition to the «multiplicity» stereotype. A variation point can be optional, mandatory or located several times in the same application. The «*variation\_point*» stereotype has the following tagged values: (1) *open*, specifying whether the variation point is open or closed, i.e., whether or not product-specific variants that are not specified in the domain models can be added at this point, and (2) *card*, which stands for “cardinality”, indicating a variants' selection rule in the form of the range of variant types that need to be chosen for the given variation point. The common cardinalities are 1..1 (XOR), 1..\* (OR), 0..1 (optional XOR), and 0..\* (optional OR). Note that there are differences between the «multiplicity» stereotype and the cardinality tagged values. A variation point, for example, can be optional (e.g., «optional many»), while its cardinality specification is mandatory (e.g., '1..\*'), indicating that this variation point may not be included in a particular application, but if it is, then at least one of its variants (as specified in the domain model) has to be selected. Similarly, an open variation point can be mandatory (e.g., «mandatory many») while its cardinality specification is optional (e.g., '0..\*'), indicating that this variation point has to be included in a particular product, but possibly uses particular, product-specific variants (not specified in the domain models).

Each variant is specified using the «*variant*» stereotype in addition to the «multiplicity» stereotype. A variation point and its variants should be of the same type (e.g., classes, attributes, associations). Since variation points may specify structural and behavioral aspects that are relevant to all their variants, we model the relationships between variants and the relevant variation point via inheritance relationships. When

not applicable, i.e., for variation points and variants that are not classifiers, such as attributes, operations, and combined fragments, the relationships between variants and variation points are specified using a tagged value, *vp*, associated with the «variant» stereotype; *vp* specifies the name of the corresponding variation point. Note that a domain element can be stereotyped by both «variation point» and «variant», enabling the specification of hierarchies of variants.

As an example of the specification of variation points and variants in ADOM, consider the Item concept, described in Fig. 4. Items, as opposed to Item Types, are the actual elements that are checked out. Thus, they must have attributes which specify their unique identifiers, overdue periods, and fees. In addition, items may also have attributes which specify their check-in details, check-out details, and statuses. Items are primarily divided into virtual and physical items, each of which is a variant of the 'Item' concept. Handling fees in this model differ according to the item classification; physical items are likely to handle overdue, damage, and loss fees. Physical items must also have location details. Virtual items, on the other hand, typically have no location details and no overdue, damage, or loss fees. They have to handle loan fees and may need to support reservation fees. According to the tagged values of the 'Item' variation point (open=false and card='1..2'), a particular product in the line cannot include items which are neither physical nor virtual. However, an item can be both virtual and physical, e.g., books in a library that are loaned as both physical copies and files.

To exemplify the usage of the above model in the context of a hotel reservation system, consider the two variants of items depicted in Fig. 5. Rooms are physical items with location

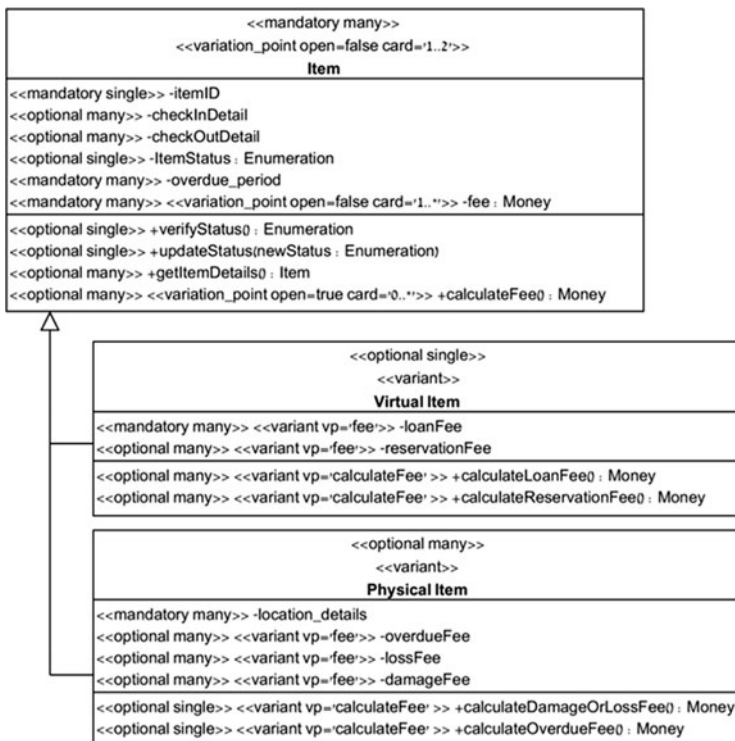


Fig. 4 Part of the CICO domain model, exemplifying the «variation\_point» and «variant» stereotypes

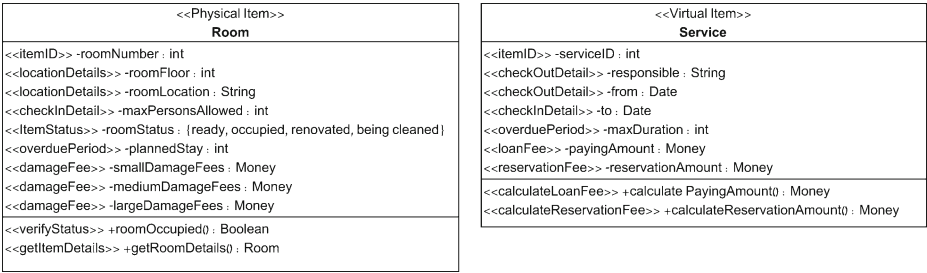


Fig. 5 Exemplifying the semantics of the «variation\_point» and «variant» stereotypes in the context of a hotel reservation system

details and three types of damage fees, while services are virtual items with loan and reservation fees. Here again the domain elements appear as stereotypes in order to support the verification of all domain constraints.

3.3 Reuse Guidance in ADOM

The «reuse» stereotype in ADOM aims to guide the reuse of domain elements while developing a particular software product. It has an associated tagged value ‘mechanism’, which can take different values that represent the different applicable mechanisms. In this paper, we refer only to the following three values of this tagged value: s – usage (a representative of the selection reuse mechanisms), e – extension (a representative of the substitution reuse mechanisms), and se – not constrained (i.e., either usage or extension).

Figure 6 illustrates the «reuse» stereotype through two additional concepts of the CICO product line: Lending and Lending Policy. Both concepts are modeled as classes and are required to select legal sub-sets of characteristics (i.e., attributes and operations that satisfy the multiplicity constraints). The “follows” association, on the

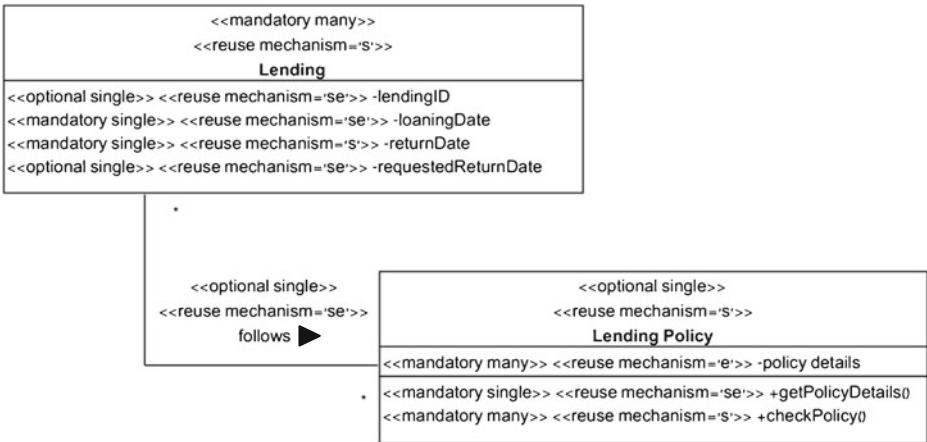


Fig. 6 Part of the CICO domain model exemplifying the «reuse» stereotype

other hand, which connects the classes Lending and Lending Policy, can be substituted in a particular product in the line in order to extend the specification of this relationship. Four uses of the “follows” association in particular applications are depicted in Fig. 7: (a) the association remains as is, (b) an association class is added, (c) the association multiplicity is changed in order to be more restrictive, and (d) the direction of the association is changed (to be unidirectional). These changes are not explicitly guided by the domain model, but are allowed by the «reuse» stereotype with the value of ‘se’ to its mechanism tagged value.

## 4 Experiment Description

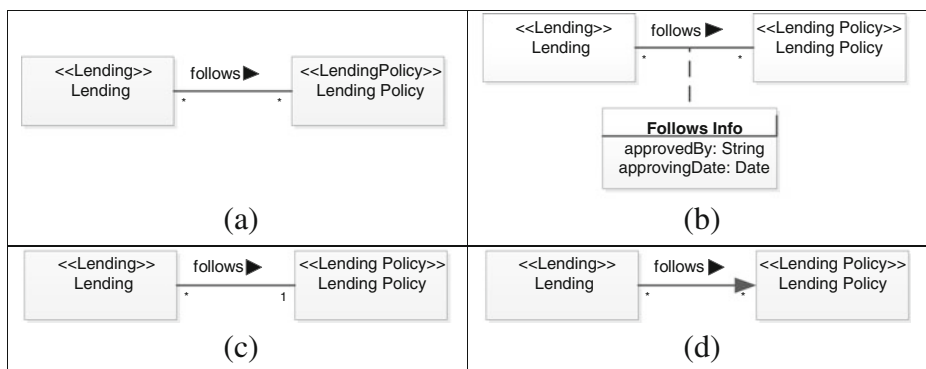
In order to examine the contribution of the different specification aids in the ADOM method to the comprehensibility of UML-based domain models, we conducted a controlled experiment using one factor with multiple treatments. As the commonality specification aids are part of all SPLE methods, we examined how adding variability specification and/or reuse guidance aids affects the comprehensibility of a domain model. The rest of this section elaborates on the experiment goals and hypotheses, subjects, and materials and tasks.

### 4.1 Experiment Goals and Hypotheses

As mentioned earlier, our major goal was to check whether explicit specification of variability aspects and/or reuse guidance affects the comprehensibility of domain models. To this end, we phrased the following six null hypotheses:

$H_{none-var}^0$  : There will be no significant difference in the comprehensibility of domain models in which both variability aspects and reuse guidance are *not explicitly* specified and domain models in which only variability aspects are *explicitly* specified.

$H_{none-reuse}^0$  : There will be no significant difference in the comprehensibility of domain models in which both variability aspects and reuse guidance are *not explicitly* specified and domain models in which only reuse guidance is *explicitly* specified.



**Fig. 7** Exemplifying the semantics of the «reuse» stereotype

$H_{none-both}^0$  : There will be no significant difference in the comprehensibility of domain models in which both variability aspects and reuse guidance are *not explicitly* specified and domain models in which these aspects are *explicitly* specified.

$H_{var-both}^0$  : There will be no significant difference in the comprehensibility of domain models in which only variability aspects are *explicitly* specified and domain models in which both variability aspects and reuse guidance are *explicitly* specified.

$H_{reuse-both}^0$  : There will be no significant difference in the comprehensibility of domain models in which only reuse guidance is *explicitly* specified and domain models in which both variability aspects and reuse guidance are *explicitly* specified.

$H_{reuse-var}^0$  : There will be no significant difference in the comprehensibility of domain models in which only reuse guidance is *explicitly* specified and domain models in which only variability aspects are *explicitly* specified.

The alternative hypotheses are:

$H_{none-var}^1$  : There will be a significant difference in the comprehensibility of domain models in which both variability aspects and reuse guidance are *not explicitly* specified and domain models in which only variability aspects are *explicitly* specified.

$H_{none-reuse}^1$  : There will be a significant difference in the comprehensibility of domain models in which both variability aspects and reuse guidance are *not explicitly* specified and domain models in which only reuse guidance is *explicitly* specified.

$H_{none-both}^1$  : There will be a significant difference in the comprehensibility of domain models in which both variability aspects and reuse guidance are *not explicitly* specified and domain models in which these aspects are *explicitly* specified.

$H_{var-both}^1$  : There will be a significant difference in the comprehensibility of domain models in which only variability aspects are *explicitly* specified and domain models in which both variability aspects and reuse guidance are *explicitly* specified.

$H_{reuse-both}^1$  : There will be a significant difference in the comprehensibility of domain models in which only reuse guidance is *explicitly* specified and domain models in which both variability aspects and reuse guidance are *explicitly* specified.

$H_{reuse-var}^1$  : There will be a significant difference in the comprehensibility of domain models in which only reuse guidance is *explicitly* specified and domain models in which only variability aspects are *explicitly* specified.

The independent variable in the experiment was the explicitly provided specification aids. As commonality specification aids were always provided, the values of this variable could be one of four possibilities: (1) commonality specification, (2) commonality specification and reuse guidance, (3) commonality specification and variability specification, and (4) commonality specification, variability specification, and reuse guidance. The comprehension level, which was measured as the number of correct answers and correct explanations, is the dependent variable in the experiment.

## 4.2 Subjects

The subjects were 116 third year undergraduate students in an Information Systems Engineering department taking a mandatory course on object-oriented analysis and design. The subjects had background in computer science and information systems engineering through courses they had taken, such as programming languages, procedural analysis and design of information systems, and introduction to databases. The experiment took place

during the final exam of the course. The students had about 90 minutes to answer the part of the exam that directly referred to the experiment.

The subjects were divided into classes by the University Exams Unit without the interference of the authors. In each class the students were provided with alternating form types according to their seating positions so that this arbitrary division into the four experiment treatments closely approximated random division. This way we neutralize possible biases that might affect the division (e.g., age, education, work experience, gender, etc.), although the similarity in the subjects' background was relatively high, as all of them were undergraduate students in the same program. This kind of assignment of participants to treatment groups is often used in empirical evaluations of modeling techniques (Ramesh and Topi 2002).

As noted, each group received a different exam form according to the explicitly provided specification aids. Table 1 presents the students' division into groups. After performing the Kruskal-Wallis test on the average grades of the students in their studies, no significant difference was found between the groups ( $\chi^2=3.147$ ,  $p=0.37$ ).

### 4.3 Experiment Materials and Tasks

As mentioned earlier, the experiment took place during the final exam of an object-oriented analysis and design course. In this course, students studied and practiced ADOM for 12 academic hours and reached a level of understanding which enabled them to perform the experiment's tasks. In order to avoid deviation, we spent a similar amount of time teaching each specification aid and demonstrating each diagram type.

The experiment was designed as one factor with four treatments (see Table 1). All four groups of students received the same task, although the models differed, depending on the provided specification aids. In all cases the models referred to the Check-In Check-Out (CICO) product line and included use case, class, and sequence diagrams, which represented the functional, structural, and dynamic aspects of the product line, respectively. The students in the 'control' group, however, received no explicit variability specification and reuse guidance. Nevertheless, in order to preserve the "information equivalence" principle (Siau 2004; Burton-Jones et al. 2009) among the models of the different groups, the constraints regarding the unsupported aspects were modeled via "regular" UML utilities, such as inheritance and notes. Three experts verified that all models indeed represented the same information on the CICO product line. The models used for the 'reuse and variability' group

**Table 1** The students division into groups

Group name	Size	Specification aids	Utilized stereotypes
Control	29	Commonality specification	«multiplicity», «requires», «excludes»
Reuse	29	Commonality specification and Reuse guidance	«multiplicity», «requires», «excludes», «reuse»
Variability	29	Commonality specification and Variability specification	«multiplicity», «requires», «excludes», «variation point», «variant»
Reuse and variability	29	Commonality specification, Variability specification and Reuse guidance	«multiplicity», «requires», «excludes», «variation point», «variant», «reuse»

can be found in Appendix C and the models of the other three groups can be found with the experimental material.<sup>1</sup>

The subjects received a questionnaire with 15 statements referring to the CICO product line model. The students had to decide whether each statement was true or false, to indicate the diagram or diagrams which allowed them to reach the decision, and to provide an explanation for their decision based on the provided model. The full list of questions can be found in Appendix C.<sup>2</sup> Note that in order to avoid interactions among the different diagram types, the information for answering a particular question correctly resided in a single diagram type. For example, to answer question #1 (see Appendix C), the use case diagrams had to be consulted, as the question was about functionality. The information required for answering question # 6, on the other hand, is in the class diagrams, as this question deals with structural aspects. This separation was made possible by the choice of diagram types for the experiment which represent different aspects of the modeled product line.

Before the experiment took place the ethics committee of the department examined the experiment design and gave its approval. In addition, since we had different versions of the exam, we normalized all grades according to the average grades of the different groups. Lastly, we compared the grades achieved by the students in this experiment to the exam grades of the same course in previous years. We found that the grades in the experiment were on the average higher, while their distribution was similar to the distributions of the exam grades in previous years. Thus, we claim that the subjects' grades were not negatively affected by the experiment.

## 5 Experiment Results

The subjects' questionnaires were checked by a grader who was not involved in this research and who was given a predefined solution in order to grade the subjects' answers. The comprehension level, which as noted is a dependent variable in this empirical study, was measured as the number of correct answers (true/false) plus the number of fully or partially correct explanations (arguments): each correct true/false answer was 1 point, while an explanation could be 0, 0.5, or 1 point for representing an incorrect, partial or full argumentation, respectively. Examining the Pearson correlations between the answers, the explanations, and the overall comprehension levels (Witte and Witte 2009; pp. 133–137), we found that the students' answers and explanations are highly correlated (correlations ranging from 0.637 to 0.992 with a significant level of 0.01). Thus, in the rest of the paper we present and refer only to the comprehension levels which sum the answer and explanation scores.

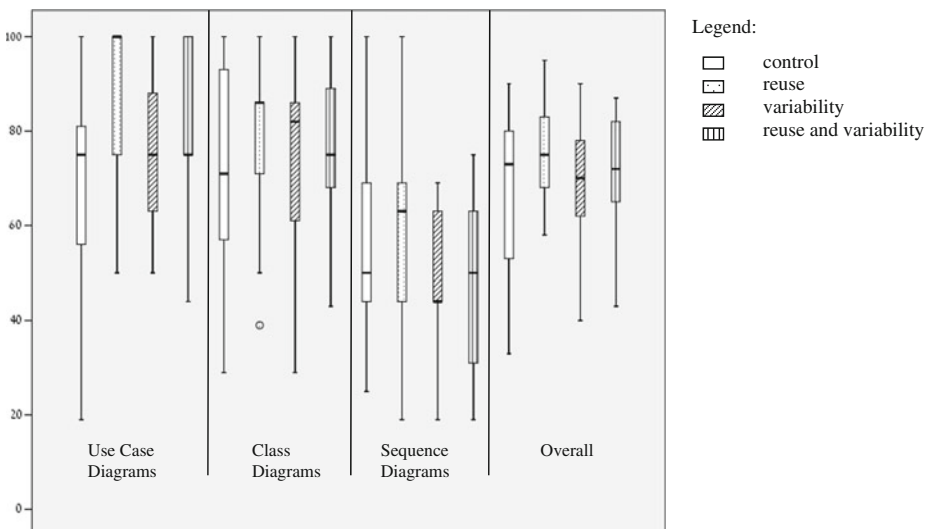
Table 2 summarizes the descriptive statistics of the results, whereas Fig. 8 visually presents this information in boxplots. The thick horizontal line in each boxplot is the median of the overall results (in percentages); the box stands for the 2nd and 3rd quartiles and the whiskers represent the 1st and 4th quartiles. Outliers, which represent measures that fall beyond the whiskers, are denoted by circles (for small deviations) or stars (for larger deviations). Figure 8 further elaborates on the achievements of the subjects according to the employed diagram types and overall.

<sup>1</sup> The experimental material is accessible at: <http://mis.hevra.haifa.ac.il/~iris/research/SPLEeval/ComADOM.htm#reusability>.

<sup>2</sup> The questionnaire was given to the participants in their mother tongue. Thus, Appendix C is actually the translation of the questions into English.

**Table 2** Overall comprehensibility of the domain models

Group	Measure	Use case diagram	Class diagram	Sequence diagram	Overall
Control	Mean	70.69	70.83	56.24	66.72
	Median	75.00	71.00	50.00	73.00
	Min	19.00	29.00	25.00	33.00
	Max	100.00	100.00	100.00	90.00
	StdDv.	21.60	24.78	19.21	16.35
Reuse	Mean	86.66	81.00	58.07	76.34
	Median	100.00	86.00	63.00	75.00
	Min	50.00	39.00	19.00	58.00
	Max	100.00	100.00	100.00	95.00
	StdDv.	16.93	14.39	20.20	10.37
Variability	Mean	74.83	75.97	49.86	68.52
	Median	75.00	82.00	44.00	70.00
	Min	50.00	29.00	19.00	40.00
	Max	100.00	100.00	69.00	90.00
	StdDv.	17.34	17.44	15.14	12.72
Reuse and variability	Mean	81.72	78.03	48.31	71.17
	Median	75.00	75.00	50.00	72.00
	Min	44.00	43.00	19.00	43.00
	Max	100.00	100.00	75.00	87.00
	StdDv.	18.11	16.75	17.94	12.91
$\chi^2$		11.01	2.28	4.51	6.41
$p$		0.01	0.52	0.21	0.09


**Fig. 8** Boxplots representing the distribution of the domain model comprehensibility

Examining the median scores in that figure, it is clear that in all cases the best results were achieved by the subjects in the ‘reuse’ group whose models were explicitly equipped with reuse guidance. Furthermore, referring to the group equipped with only commonality specification aids as the ‘control’ group,<sup>3</sup> we can observe that the results regarding the comprehensibility of models equipped with variability specification were mixed; the addition of variability specification increased the comprehensibility of class diagrams with respect to the ‘control’ group but did not affect the comprehensibility of use case diagrams and decreased the comprehensibility of the sequence diagrams. Another observation that can be made about Fig. 8 is that for use case and sequence diagrams, the comprehension level of models equipped with both variability specification and reuse guidance was between the comprehension levels of models equipped with only one of these specification aids. However, in these cases, the comprehension level of the ‘reuse and variability’ group was similar to that of the ‘control’ group. We will discuss these differences between the diagram types further in the next section.

To examine the data statistically, we chose the non-parametric Kruskal-Wallis test, as the distribution of the grades in the four experiment groups did not meet the parametric test assumptions. Applying this test, we found that overall there may be a difference in the comprehension level in at least one of the groups, but this difference is not statistically significant ( $\chi^2=8.47$ ,  $p=0.09$ ). With use case diagrams, however, there is a statistically significant difference ( $\chi^2=11.01$ ,  $p=0.01$ ). Thus, we applied the Mann–Whitney with Bonferroni correction<sup>4</sup> test to pairs of groups in this case. We found that there are statistically significant differences in the use case diagram’s comprehensibility between the ‘control’ and the ‘reuse’ groups ( $U=239$ ,  $p<0.01$ ,  $r=0.38$ ) and between the ‘reuse’ and the ‘variability’ groups ( $U=263$ ,  $p=0.01$ ,  $r=0.33$ ),<sup>5</sup> in favor of the ‘reuse’ group in both cases.

In order to form deeper insights into the effects of the various specification aids, the different comprehension questions were divided according to the specification aids they mainly examined. Table 3 presents the descriptive statistics for this analysis. The comprehensibility of commonality-related aspects was similar in all groups, but this is not surprising as the models in all groups were equipped with the same commonality specification aids. Moreover, the results show that complicating the notations by introducing additional (variability- and/or reuse-related) stereotypes does not negatively affect the comprehensibility of commonality-related aspects in use case and class diagrams. In sequence diagrams, on the other hand, these additions decreased comprehensibility, however the differences are not statistically significant ( $\chi^2=0.86$ ,  $p=0.84$ ).

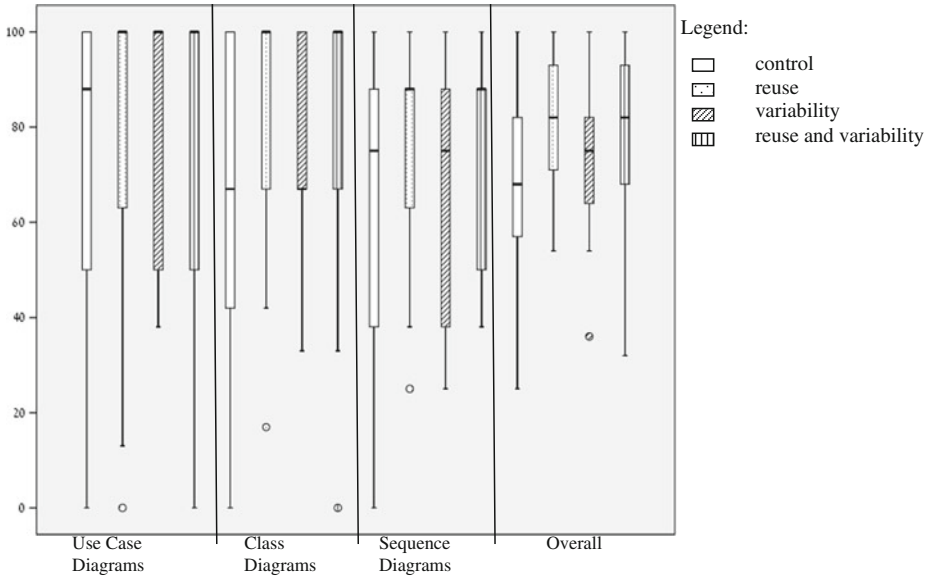
Figures 9 and 10 present in boxplots the distribution of the comprehensibility of variability- and reuse-related aspects, respectively. Examining the median scores in Fig. 9, the best comprehensibility of variability-related aspects was achieved by the subjects in the ‘reuse’ and ‘reuse and variability’ groups. These results were always better than those of the ‘control’ group, while the results of the ‘variability’ group were the same or a little bit higher than those of the ‘control’ group for the same category of issues. Still, the comprehensibility of variability-related issues in models not equipped with reuse guidance was lower than the comprehensibility of these issues in models equipped with reuse guidance. With respect to variability-related

<sup>3</sup> As noted, we referred to this group as ‘control’ since all UML-based SPLE methods support the specification of commonality with similar aids (see Appendix A and Section 2.3). Therefore all the models in our experiment were equipped with the same commonality specification aids.

<sup>4</sup> We adopted the more conservative correction approach of Bonferroni for performing multiple comparisons. As we had 6 comparisons, we looked at a significance level lower than 0.0083 (i.e., 0.05/6).

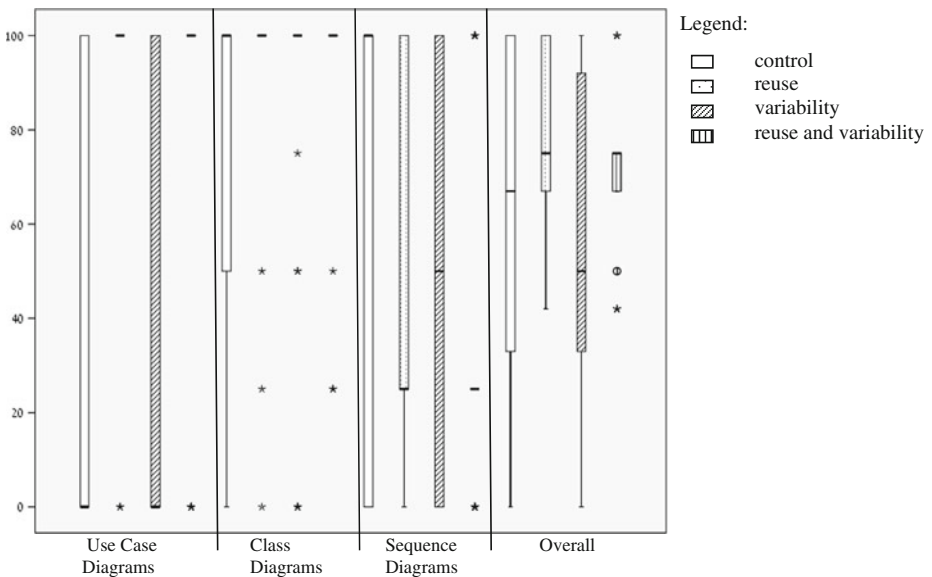
<sup>5</sup>  $r$  is the size effect.





**Fig. 9** Boxplots representing the distribution of the comprehensibility of variability-related issues

issues, Fig. 10 shows that the addition of reuse guidance in the ‘reuse’ and the ‘reuse and variability’ groups improved the use case diagram’s comprehensibility, did not affect the class diagram’s comprehensibility, and decreased the sequence diagram’s comprehensibility. The comprehension in the ‘variability’ group was similar to that of the ‘control’ group for use case and class diagrams. However,



**Fig. 10** Boxplots representing the distribution of the comprehensibility of reuse-related issues

for sequence diagrams, any addition of variability specification and/or reuse guidance decreased comprehensibility.

We once again applied the Mann–Whitney with Bonferroni correction test to pairs of groups in cases where statistically significant differences were found, namely, the overall comprehensibility of variability-related aspects and use case diagrams and the overall comprehensibility of reuse-related aspects. Table 4 presents this statistical analysis, where the bold numbers and grey cells indicate statistically significant results. The results show that providing reuse guidance improved not only the comprehensibility of reuse-related aspects, but also the comprehensibility of variability-related issues. In particular, the inclusion of reuse guidance without variability specification significantly improved the comprehensibility of variability-related issues, the inclusion of reuse guidance with or without variability specification improved the comprehensibility of reuse-related aspects in use case diagrams, and the inclusion of reuse guidance without variability specification improved the overall comprehensibility of reuse-related aspects.

Based on these results, none of the null hypotheses can be rejected with respect to the overall domain model comprehensibility. However, when referring to variability-related aspects, only  $H^0_{none-reuse}$  can be rejected and thus  $H^1_{none-reuse}$  can be accepted. This indicates that only when adding explicit reuse guidance would the comprehensibility of the variability-related aspects of the domain model improve. When referring to reuse-related aspects, both  $H^0_{none-reuse}$  and  $H^0_{reuse-var}$  can be rejected and thus both  $H^1_{none-reuse}$  and  $H^1_{reuse-var}$  can be accepted. This implies that adding explicit reuse guidance improves the comprehensibility of reuse-related aspects. This is extremely remarkable when dealing with use case diagrams; in this case  $H^0_{none-reuse}$ ,  $H^0_{none-both}$ ,  $H^0_{var-both}$  and  $H^0_{reuse-var}$  can be rejected, i.e.,  $H^1_{none-reuse}$ ,  $H^1_{none-both}$ ,  $H^1_{var-both}$ , and  $H^1_{reuse-var}$  can be accepted.

## 6 Discussion

The results show that the performance of the ‘reuse’ group was the best in most cases. However, the performance of the ‘reuse and variability’ group, whose models were also equipped with variability specification, were worse than the ‘reuse’ group (see Fig. 8). This may be due to the fact that using two specification aids together, variability specification and reuse guidance, complicated the models and negatively affected comprehensibility. Moody

**Table 4** Statistical analysis of the main differences in domain model’s comprehensibility

Compared groups	Variability-related overall			Reuse-related aspects					
				UC			Overall		
	<i>U</i>	<i>p</i>	<i>r</i>	<i>U</i>	<i>p</i>	<i>r</i>	<i>U</i>	<i>p</i>	<i>r</i>
Control-reuse	247.5	0.007	0.35	190.5	0.000	0.57	249.5	0.006	0.31
Control-variability	348.5	0.260	0.15	419.5	0.986	0.00	408.5	0.849	0.00
Control-reuse and variability	301.0	0.062	0.24	237.0	0.001	0.43	309.0	0.079	0.16
Reuse-variability	306.0	0.074	0.23	190.5	0.000	0.57	209.5	0.001	0.37
Reuse-reuse and variability	383.5	0.563	0.08	377.0	0.231	0.16	298.5	0.050	0.26
Variability-reuse and variability	359.0	0.337	0.13	237.0	0.001	0.43	270.5	0.018	0.21

(2009) refers to this issue through the graphic economy principle which suggests using a cognitively manageable number of different graphical symbols in visual notations. Most UML diagram types exceed this cognitively manageable number, which is around six categories, thus adding new concepts via the stereotypes mechanism may only worsen the situation. This principle also suggests some kind of explanation as to why the performance of the ‘reuse’ group (which add only one stereotype) was better than that of the ‘variability’ group (with the addition of two stereotypes) and the ‘reuse and variability’ group (with the addition of three stereotypes), but does not offer an explanation as to the differences in the performance between the ‘variability’ and ‘reuse and variability’ groups in favor of the more “expensive” notation (the ‘reuse and variability’ group) in the sequence diagrams. The reason for these differences may be that the additional information in the ‘reuse and variability’ group, i.e., the reuse guidance, improved the comprehensibility even though the notation was more complicated.

Although we do not investigate the usage of the different UML diagrams for supporting different SPLE activities, we cannot ignore the clear outcome of our experiment, according to which the most comprehensible diagram type is the use case, followed by class, and finally by sequence diagram (see Fig. 8). Since each student answered all the questions which referred to different diagram types, our study resulted in dependent samples. Thus, we applied the Friedman and Wilcoxon statistical tests to analyze the differences in the comprehensibility of the various diagram types. We found out that the differences between the diagram types are statistically significant ( $p < 0.001$ ). We believe that these differences originate from the language complexity; use case diagrams are less conceptually complex than class diagrams (Siau and Cao 2001) and sequence diagrams are one of the most difficult and time-consuming UML diagrams to develop and understand (Song 2001).

As the analysis of the results demonstrates, the superiority of the reuse guidance utilities over the variability specification aids can be justified by two explanations. Firstly, reuse guidance explicitly refers to the usage of domain models in a concrete application artifact, while variability specification refers to the allowed variety in a certain product line. Thus, variability specification can be considered more abstract than reuse guidance and more complicated to understand. Secondly, the reuse concept is modeled in ADOM via a single stereotype which encapsulates all the required guidance via the associated tagged values. The variability concept, on the other hand, is specified via two different stereotypes, «variation point» and «variant», as well as the relationships among them (usually through inheritance relationships). Moreover, the low success rate in comprehending the variability of the sequence diagram, for example, can be partially attributed to the implicit relationships between variation points and variants through tagged values in this type of diagram. The subjects received the specification of the variation point separately from those of the different variants; they had to integrate the information from the different diagrams and come up with single answers that reflect their understanding of the allowed variability. This outcome can be explained or justified by the claim of Kim et al. (2000) that understanding a model represented by multiple diagrams involves perceptual and conceptual integration processes and thus is a source of cognitive difficulty. Our conjecture is that reuse guidance focuses on single elements and is more concrete, whereas variability specification refers to multiple elements (variation points and different variants) in a higher abstraction level.

An interesting observation regarding the comprehensibility of variability-related aspects in use case diagrams is that the different aids improve comprehensibility, yet no differences exist among them (see Fig. 9). This result may be attributed to the relative simplicity of the UML use case diagram. Moreover, the statistically significant difference in the

comprehensibility of reuse-related aspects in use case diagrams may indicate that explicit guidelines are superior to implicit guidelines (see Fig. 10). This general statement is also confirmed by Nugroho (2009) who states that “the amount of information in a model can be increased/reduced by being more explicit/implicit in portraying modeling constructs using UML modeling notations”.

## 7 Threats to Validity

The above results need to be considered in view of several threats to validity categorized by Wohlin et al. (2000) as construct, internal, conclusion, and external validity.

**Construct validity** threats, which concern the relationships between theory and observation, are mainly due to the method used to assess the outcomes of the tasks. In this experiment we examined a specific method only, ADOM, and thus the results may be influenced by the selected method. However, we based our choice on a comprehensive comparison we made, whose results are presented in Appendix A. The chosen method, ADOM, is similar in many aspects to other UML-based SPLE methods, but it explicitly refers to different variability specification and reuse guidance aspects (as elaborated in Section 2.3). Nevertheless, the way ADOM facilitates the various specification aids and its richness might affect the results. For example, the use of multiple stereotypes on a single element and the association of tagged values to stereotypes may cause problems in model readability and negatively affect comprehensibility. Thus, repeating the experiment with other UML-based methods and different sets of UML diagram types is needed.

**Internal validity** threats, which concern external factors that might affect the dependent variables, may be due to individual factors, such as familiarity with the domain, the degree of commitment by the subjects, and the training level the subjects underwent. These effects are mitigated by the experiment design that we chose. That is, we used one factor experiment design with four treatments and random assignments that should eliminate these threats. As the experiment was part of an exam, the motivation and commitment of the students were high in all groups. Furthermore, applying the Kruskal-Wallis test on the students’ average grades, we found no statistically significant differences between the four groups.

**Conclusion validity** concerns the relationship between the treatment (the used sets of explicit specification aids) and the outcome. We followed the various assumptions of the statistical tests when analyzing the results. For example, when data normality could not be assumed, we performed a statistical analysis using mainly non-parametric tests (i.e., the Kruskal-Wallis). In addition, we used a pre-defined solution for grading the students’ answers and thus only a limited amount of human judgment was required (for grading the arguments of the explanations and deciding whether they provided full or partial answers).

Lastly, **external validity** concerns the generalizability of the results. The main threat in this area stems from the choice of subjects and from using simple tasks in the experiment. The subjects were undergraduate students with little experience in software engineering in general and in SPLE in particular, but they represented a population of students specifically trained in domain modeling. The students were further trained and had practiced the method used in the experiment, ADOM, for 12 academic hours. The average of the grades the students achieved in the experiment, which ranged from 67 to 78, indicate that they basically understood the profile under test. Furthermore, the subjects were at an advanced stage in their studies and close to becoming software engineers and developers. Thus, they

approximate the population intended to use domain modeling methods and ADOM. More generally, Kitchenham et al. (2002) argue that using students as subjects instead of software engineers is not a major issue as long as the research questions are not specifically focused on experts, as is the case in our empirical study.

The tasks used in this experiment were limited in their size and complexity, as they were used in an exam with a limited amount of time and under stress conditions. However, both models and tasks refer to important aspects in SPLE in general and domain modeling in particular. The questions were designed to be simple enough without being too obvious. In addition, the models used in the experiment were checked by three experts in order to guarantee correctness and similar expressiveness. Furthermore, during the exam the students could get extra time in order to complete the task in the best way. Only further studies may confirm or disconfirm whether our results can be generalized to more experienced subjects (e.g., professional software product line engineers) and more complicated tasks.

## 8 Summary and Future Work

In this paper we presented the design and results of an experiment whose aim was to evaluate the comprehensibility of different specification aids used in UML-based domain models. These aids include: (1) *specification of commonality* by means of mandatory and optional elements and dependencies between elements, (2) *specification of variability* by means of variation points, possible variants, and rules for selection of variants in certain variation points, and (3) specification of reuse guidance by referring to specific reuse mechanisms which can be used in order to create concrete application artifacts. Particularly, we examined how the various modeling aids explicitly incorporated within a domain model affect the comprehensibility of the model by non-experts. To the best of our knowledge, this is the first attempt to examine the comprehensibility of domain artifacts in such a wide perspective. The experiment results show that providing additional specification and guidance aids helps comprehend domain models. In particular, we found that providing explicit reuse guidance has the highest influence. Our conjecture is that this utility is usually attached to specific domain elements, thus making its guidance clear and more concrete. The variability specification also improved comprehensibility, but to a limited extent when compared to the reuse guidance. We believe that the reason for this difference is the fact that variability specification consist of multiple elements (variation points and variants) and these are specified in a higher level of abstraction in view of the whole software product line. Following the results, it appears that it is important to pay careful attention to the way reuse guidance is incorporated into domain models and not just to the associated development processes, as most UML-based methods for specifying product lines do. It is also important to further investigate how to improve the comprehensibility of variability specification utilities in ADOM and in SPLE methods in general.

In the future we plan to perform additional experiments with different UML-based methods and methods in various categories, e.g., UML-based and feature-oriented methods. We also intend to extend the investigated tasks to application engineering tasks that support the creation of application artifacts based on domain models. This kind of task increases the required comprehension, as it requires not only understanding an existing domain model but also understanding how to utilize it for creating a concrete application artifact. We also plan to further understand the role of the various specification aids in both domain and application engineering tasks by applying qualitative research methods.

## Appendix A: Comparison of UML-based SPLE Methods

**Table 5** Comparison of UML-based SPLE methods

Method name	Commonality specification	Variability specification	Reuse guidance	Supported diagrams
SPLIT - Coriat et al. 2000	Explicit specification via the existence attribute	Explicit specification of VPs and variants; general relationships connect VPs and variants	Reference to variability implementation (insertion, extension, and parameterization)	Class
Morisio et al. 2000 (Lazilha et al. 2004)	Implicit specification of mandatory elements and explicit specification of optional and alternative elements	Explicit specification of variable elements		Class
Riebisch et al. 2000	Implicit specification of mandatory elements	Explicit specification of variable elements		Activity, component
Clauß 2001	Implicit specification of mandatory elements and explicit specification of optional elements and dependencies between elements	Explicit specification of both VPs and variants; enabling the specification of binding time and selection conditions of variants	Reference to variability implementation via standard UML concepts, such as generalization and parameterization	Class, (Component, Package, Collaboration)
Robak et al. 2002	Implicit specification of common elements	Explicit specification of variants; support for VP specification via branches or components		Activity, component
Maßen and Lichter 2002	Implicit specification of common elements	Implicit specification via dependencies		Use Case
PuLSE-John and Muthig 2002	Support for common elements specification as in a single-system context	Explicit specification of variants	Support for guidance of application artifacts creation by a separate decision model	Use Case
Halmans and Pohl 2003 (Bühne et al. 2003)	Explicit specification of mandatory and optional VPs; support for dependencies specification	Support for VP specification and their connections to variants; enabling the specification of selection conditions		Use Case
Ripon et al. 2003	Implicit specification of common elements	Use of a variability model, which consists of a variant model and a decision table	Use of the variability model, and especially its decision table, for guiding reuse	Activity
		Explicit specification of variants		

**Table 5** (continued)

Method name	Commonality specification	Variability specification	Reuse guidance	Supported diagrams
PLUS – Gomaa <a href="#">2004</a> (PLUSSE - Gomaa and Shin <a href="#">2002</a> )	Explicit specification of mandatory and optional elements			Use Case, class, communication, state
VPM – Webber and Gomaa <a href="#">2004</a>	Explicit specification of mandatory and optional VPs	Explicit specification of variants; indication of optional and mandatory VPs	Categorization of VPs into four types: parameterization, information hiding, inheritance, and callback	Class, component, sequence, package
Ziadi et al. <a href="#">2004</a> (Ziadi and Jézéquel <a href="#">2006</a> )	Possibility to classify optional elements	Explicit specification of both VPs and variants; variants inherit VPs; support for Virtual parts specification in sequence diagrams	Support for product derivation by using a decision model and a three step process: variant classes selection, model specialization, and model optimization	Class, sequence, interaction
Bachmann et al. <a href="#">2004</a>	Implicit specification of common elements	Explicit specification of both VPs and variants		Orthogonally to class
Kim et al. <a href="#">2004</a>	Explicit specification of mandatory, optional, and alternative elements			Use Case
Oliveira Junior et al. <a href="#">2005</a> (later version SMartyProfile <a href="#">2010</a> )	Explicit specification of mandatory and optional variants, as well as dependencies	Explicit specification of VPs and variants; explicit specification of variant selection conditions	Support for reuse in the process level through generalization, interface realization, aggregation association, and composite aggregation	Use Case, class, component
DREAM - Moon et al. <a href="#">2005</a> .	Explicit specification of mandatory and optional elements; enabling the usage of commonality ratios in a primitive requirements matrix			Use Case
Bragança and Machado <a href="#">2006</a>	Implicit specification of common elements	Specification of Location, Rejoin, extension fragment, and inclusion point		Use Case
Korherr and List <a href="#">2007</a>	Explicit specification of mandatory, optional, alternative choice variants, and dependencies between variants and/or VPs	Explicit specification of VPs and variants		Class

**Table 5** (continued)

Method name	Commonality specification	Variability specification	Reuse guidance	Supported diagrams
COVAMOF – Sun et al. 2010.	Implicit specification of mandatory elements	Explicit specification of VPs and variants; relationships between VPs and variants are separately specified in a VP interaction diagram	Enabling the specification of rules that determine variant selection conditions by realization relations	Class, Activity, Sequence, Deployment
ADOM –Reinhartz-Berger and Tsoury (2011a, b) (based on Reinhartz-Berger and Sturm 2008, 2009)	Explicit specification of mandatory and optional elements as well as dependencies among elements	Explicit specification of VPs and variants, as well as variant selection rules; variants inherit VPs when possible	Support for different reuse mechanisms specification	Use Case, Class, Sequence, State, (All)

## Appendix B: The Formalism of the ADOM Method

Next we formally define the ADOM method.

The elements of UML are classified into three categories: dependent, relational, and first order elements.

*Definition 1* *Dependent elements* are elements that depend on other elements in the model such that the omission of the dependees from the model implies the omission of the dependent elements.

Examples of dependent elements are attributes and operations in UML class diagrams that depend on their owning classes and sub-packages that depend on their owning packages.

*Definition 2* *Relational elements* are explicit binary (directional) relationships between pairs of elements.

Examples of relational elements are associations in UML class diagrams and messages in UML sequence diagrams. Note that all kinds of relationships in UML are binary or can be mapped to binary relations.

*Definition 3* *First order elements* are elements which are not relational nor dependent in the model.

All kinds of elements can be associated with the «multiplicity» stereotype which is defined next.

*Definition 4* The «multiplicity min= $n$  max= $m$ » stereotype, which can be associated to any element in the domain model, specifies the range of elements in the application artifact that can be classified as the same domain element. The two tagged values, min and max, are used for defining the lowest and upper-most boundaries of that range.

The meanings of the «multiplicity» stereotype are slightly different according to the element types: first order, dependent or relational elements. The multiplicity stereotype of a dependent element specifies the range of times this domain element can be used *in any dependee* of a software product in that line. Specifying, for example, the multiplicity of an attribute A of class C as «multiplicity min=1 max=» implies that between 1 to  $\infty$  attributes of type A have to be included *in each class* of type C in a certain product in the line.

The multiplicity stereotype of a relational element specifies the range of times this domain element can appear, connecting the relevant source and target in any product of the line. Specifying, for example, the multiplicity of an association r between class A and class B as «multiplicity min=1 max=» implies that in any product in the line in which both A and B appear, each appearance of A is connected to at least one appearance of B via associations of type r, and vice versa.

Lastly, the multiplicity stereotype of first order elements specifies the range of times the relevant domain elements can appear in any product of that line.

Note that any combination of the multiplicity stereotypes is feasible. For example, a mandatory relational element can connect two optional elements, implying that if the two elements appear in a certain product, then the relational element *must* connect them. Similarly, a mandatory dependent element may reside within an optional element, implying that if the dependee appears in a certain product, then the dependent element *must* appear too.

Optional elements can be connected via «requires» and «excludes» dependencies in order to constrain larger configurations.

**Definition 5** Let A and B be two optional elements in a domain model. A «requires» B implies that if A appears in a particular product, then B should appear too. A «excludes» B implies that if A is included in a particular product, then B should not appear. Furthermore, in order to avoid redundancy or non-feasible models, the following constraints must hold:

C1. If A requires B, then A.multiplicity.min=0 and B.multiplicity.min=0.

C2. If A excludes B, then A.multiplicity.min=0 and B.multiplicity.min=0.

The following definitions refer to the specification of variability in ADOM.

**Definition 6** The «variation\_point open=true/false card=m..n» stereotype indicates places in which variability can be introduced. The variation point can be *open* (open=true), meaning that product-specific variants not specified in the domain models can be added at this point or *closed* (open=false), i.e., addition of product-specific variants that are not specified in the domain models is not allowed at this point. The *card* tagged value indicates the minimal (m) and maximal (n) numbers of variants the can be selected for this variation point. The following constraints must hold with respect to variation points:

C3. If  $m=0$ , then open=true.

C4. If open=false, then  $m \geq 1$ .

**Definition 7** The «variant vp=name» stereotype is used for indicating a possible way to realize variability in a certain variation point. *vp* indicates the name of the variation point in case an explicit inheritance relationship between the variant and variation point cannot be specified (in the case of non-classifiers). The following constraints must hold with respect to variants and their associated variation points:

C5. The multiplicity of a variant (v) can be the same or more restricted than the multiplicity of the relevant variation point (vp). Formally expressed as:  $vp.multiplicity.min \leq v.multiplicity.min \leq v.multiplicity.max \leq vp.multiplicity.max$ .

Note that no restriction exists between the cardinality values and the multiplicity values of both variants and variation points, as cardinality refers to the selection of variants in a *single occurrence* of the variation point in a software product, while multiplicity refers to the general number of occurrences of the variation point or the variant in the *whole* product.

Finally, the reuse guidance is specified in ADOM using the «reuse» stereotype, defined below.

**Definition 8** The «reuse mechanism=name» stereotype is used to guide the usage of a domain element in various products in the line. The *mechanism* tagged value specifies the applicable reuse mechanism. In the current work we refer only to the following values: *s*—usage (a representative of the selection reuse mechanisms), *e*—extension (a representative of the substitution reuse mechanisms), and *se*—not constrained (i.e., either usage or extension). The following constraints must hold with respect to the reuse guidance of variation points and the associated variants:

C6. The reuse mechanisms of a variant (v) can be the same or more restricted than the reuse mechanisms of the relevant variation point (vp), namely:

1. If  $vp.reuse.mechanism = 'se'$ , then  $v.reuse.mechanism \in \{'s', 'e', 'se'\}$ .
2. If  $vp.reuse.mechanism = 'e'$ , then  $v.reuse.mechanism = 'e'$ .

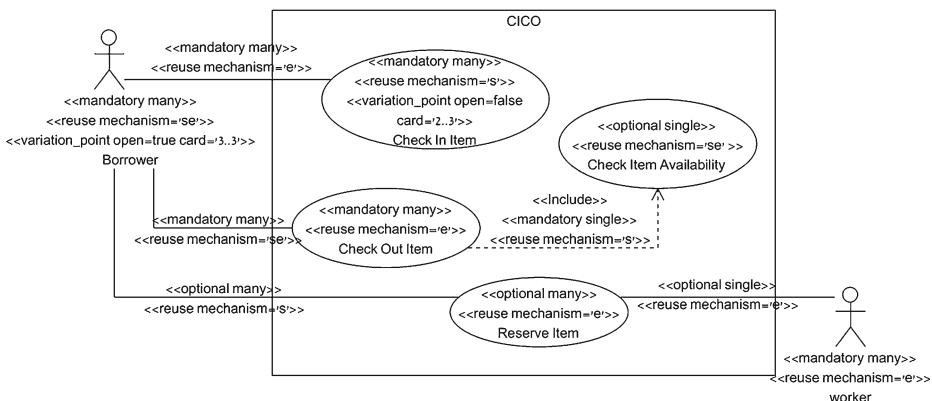
3. If  $vp.reuse.mechanism = 's'$ , then  $v.reuse.mechanism = 's'$ .

Note that there are no constraints on the reuse mechanisms of dependees and dependent elements or on the reuse mechanisms of relational elements and their associated source and target elements, as these are separate elements whose reuse may differ.

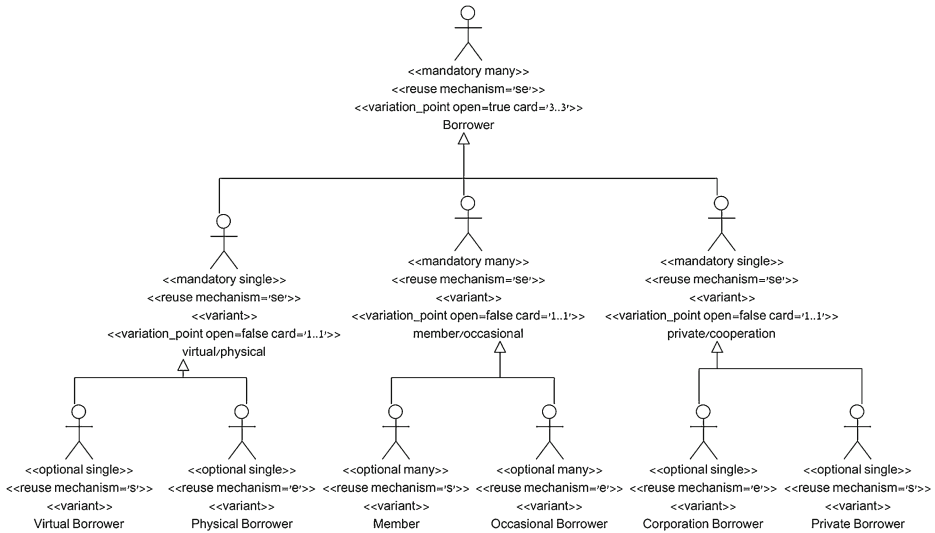
## Appendix C: The Experiment Questionnaire

Below are the models and translation of the 15 comprehension questions given to the ‘reuse and variability’ group. The questions of the other three groups were identical to the questions provided here, while the different models can be found with the experimental material at <http://mis.hevra.haifa.ac.il/~iris/research/SPLEeval/ComADOM.htm#reusability>.

1. A product in the CICO line may include an item reservation. In this case, only borrowers and workers can perform this activity that cannot undergo any refinement or extension.
2. Each product in the CICO line must interact with either private borrowers or cooperation borrowers.
3. In case a product in the CICO line has the functionality of “Check-In with Fee Calculation”, then it must include the functionality of “Calculate Fee” too.
4. In the CICO line there might be products which will not include explicit reference to fee calculation (through use cases, such as “Check-In with Fee Calculation”, “Check-In with possible Fee Calculation” or “Check-In without Fee Calculation”).
5. In the CICO line there might be products that support both Lending Policy and Waiting Lists.
6. Products within the CICO line may include associations of type “handle” between Borrower and Lending that are not of type “manage” nor of type “perform”.
7. Products in the CICO line may include Items with no loan fee information nor location details.
8. An Item in a CICO product may have both loan fee information and location details.

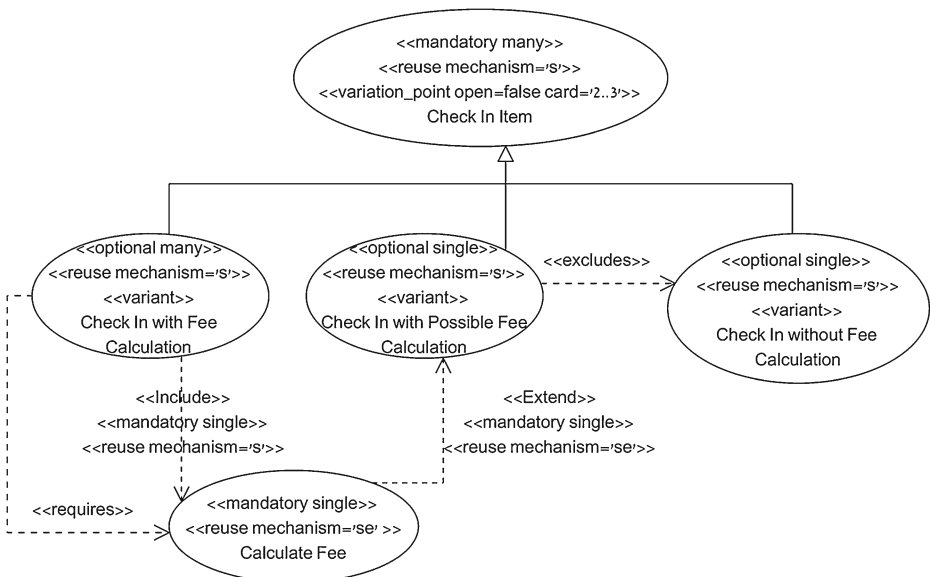


**Fig. 11** CICO model: the top level use case diagram



**Fig. 12** CICO model: the Borrower variation point

9. A product in the CICO line may include a virtual item that exhibits a method for calculating reservation fees which receives the actual lending period as a parameter.
10. In each product in the CICO line, a class of type Lending should be related to exactly one class of type Borrower.
11. In each product in the CICO line, if a class of type Borrower is associated to a class of type Lending with an association of type “manage”, then there should be an association class of type “managing” which documents the changes.



**Fig. 13** CICO model: the Check In Item variation point

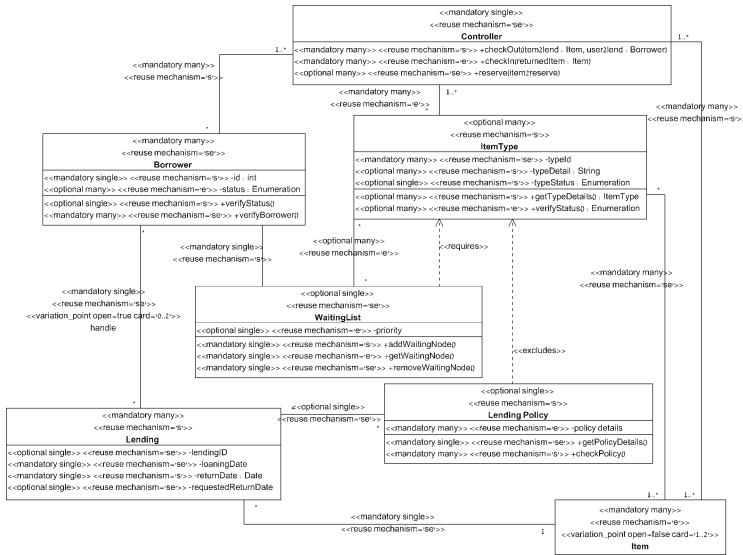


Fig. 14 CICO model: the top level class diagram

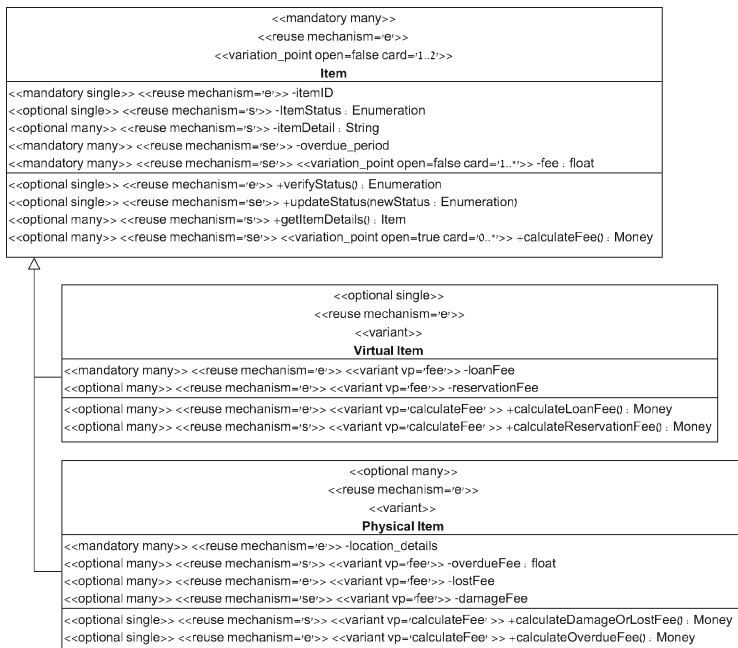


Fig. 15 CICO model: the Item variation point

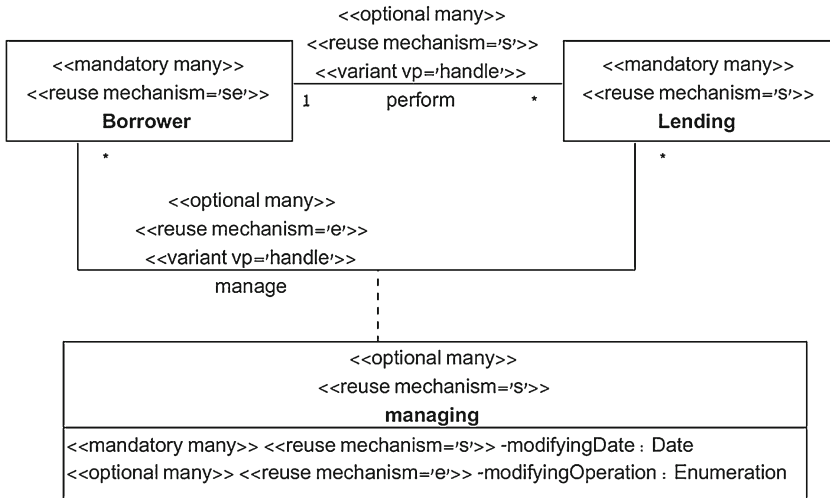


Fig. 16 CICO model: the Handle variation point

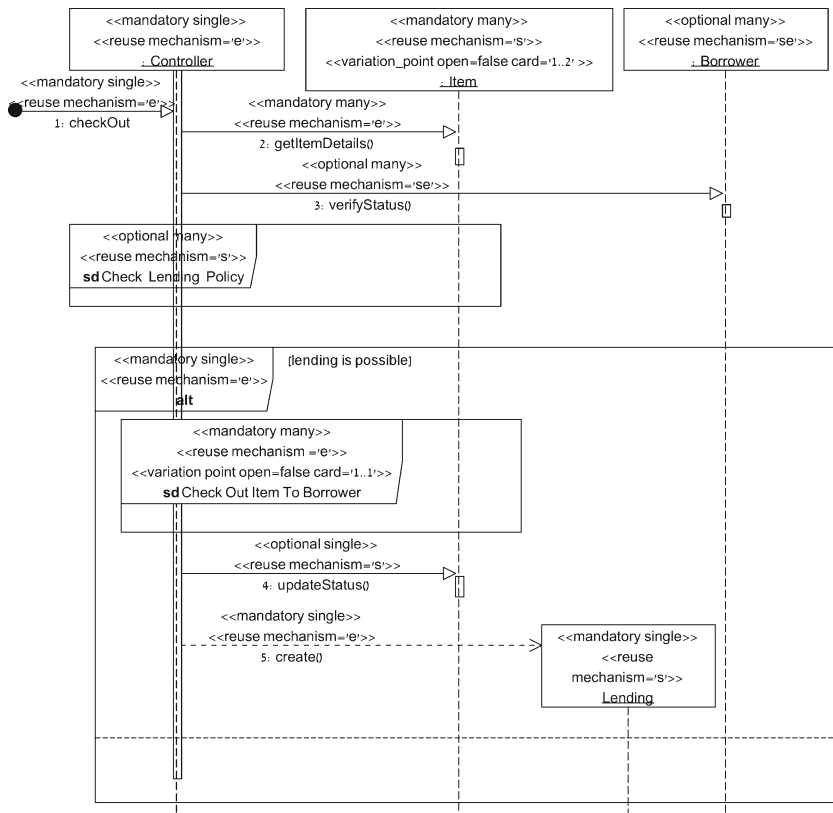
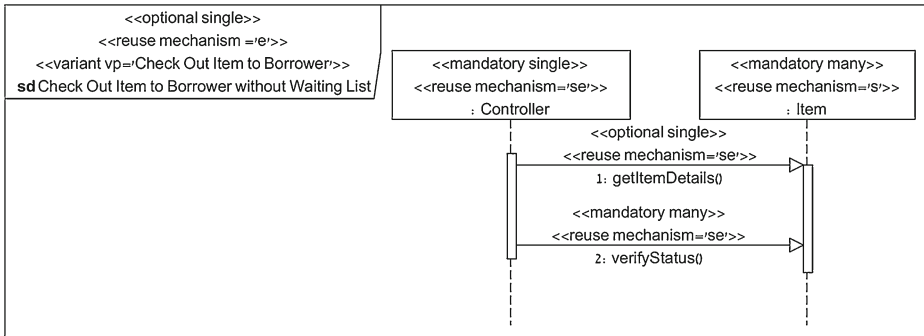
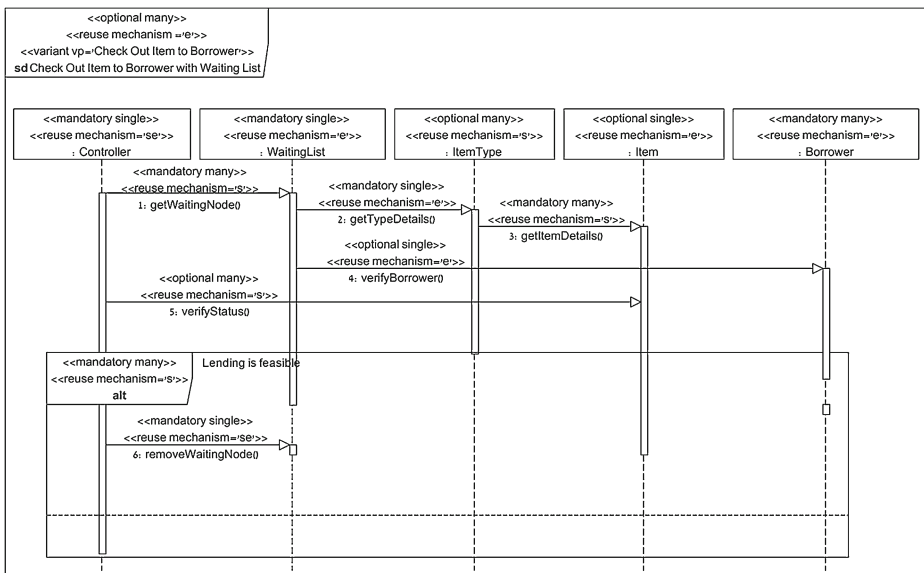


Fig. 17 CICO model: the check out scenario



**Fig. 18** CICO model: check out without waiting list

12. There might be a product in the CICO line in which a controller object will not be involved in a scenario of Check-Out Item to Borrower.
13. In a Check-Out scenario of a CICO product, one can borrow both virtual and physical items.
14. In a CICO product, a controller object can send messages other than getItemDetails and verifyStatus to an Item object while executing the scenario named “Check-Out Item to Borrower without Waiting List”.
15. It is possible that in a CICO product that does not define a waiting list, the combined fragment named “Check Out-Item to Borrower with Waiting List” will be executed while performing Check-Out.



**Fig. 19** CICO model: check out with waiting list

## References

- Ahmed F, Capretz LF (2008) The software product line architecture: an empirical investigation of key process activities. *Inf Softw Technol* 50:1098–1113
- Anastasopoulos M, Gracek C (2001) Implementing product line variabilities. *ACM SIGSOFT Softw Eng Notes* 26(3):109–117
- Bachmann F, Clements PC (2005) Variability in software product lines. Technical Report CMU/SEI-2005-TR-012, available online at <http://www.sei.cmu.edu/library/abstracts/reports/05tr012.cfm>, Accessed 9 September 2012
- Bachmann F, Goedicke M, Leite J, Nord R, Pohl K, Ramesh B, Vilbig A (2004) A meta-model for representing variability in product family development. In: van der Linden F (ed): PFE'2003, LNCS 3014, pp. 66–80.
- Bagheri E, Dasevic G (2011) Assessing the maintainability of software product line feature models using structural metrics. *Software Quality Journal*, Springer, doi:10.1007/s11219-010-9127-2
- Becker M (2003) Towards a general model of variability in product families. In: Proceedings of the Software Variability Management Workshop, University of Groningen, The Netherlands
- Bragança A, Machado RJ (2006) Extending UML 2.0 metamodel for complementary usages of the «extend» relationship within use case variability specification. In: Proceedings of the 10th International Software Product Line Conference (SPLC), pp 123–130
- Bühne S, Halmans G, Pohl K (2003) Modeling dependencies between variation points in use case diagrams. In: Proceedings of the 9th international workshop on requirements engineering – Foundation for Software Quality (REFSQ'03), pp 59–70
- Burton-Jones A, Wand Y, Weber R (2009) Guidelines for empirical evaluations of conceptual modeling grammars. *J Assoc Inf Syst* 10:495–532
- Chen L, Babar MA (2011) A systematic review of evaluation of variability management approaches in software product lines. *Inf Softw Technol* 53:344–362
- Clauß M (2001) Generic Modeling using UML extensions for variability. In: Proceedings of OOPSLA Workshop on Domain-specific Visual Languages, pp 11–18
- Clements P, Northrop L (2001) Software product lines: practices and patterns. Addison-Wesley Professional. Part of the SEI Series in Software Engineering series
- Clotet R, Dhungana D, Franch X, Grunbacher P, Lopez L, Marco J, Seyff N (2008) Dealing with changes in service-oriented computing through integrated goal and variability modelling. In: Proceeding of the second International Workshop on Variability Modelling of Software-intensive Systems (VaMoS'2008), pp 43–52
- Coplien J, Hoffman D, Weiss D (1998) Commonality and variability in software engineering. *IEEE Softw* 15 (6):37–45
- Coriat M, Jourdan J, Fabien B (2000) The SPLIT method: building product lines for software-intensive systems. In: Proceedings of the first conference on Software Product Lines: experience and research directions: experience and research directions (SPLC'2000), pp 147–166
- Czarniecki K, Kim CHP (2005) Cardinality-based feature modeling and constraints: a progress report. OOPSLA Workshop on Software Factories
- Denger C, Kolb R (2006) Testing and inspecting reusable product line components: first empirical results. In: Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering, ACM, pp 184–193
- Djebbi O, Salinesi C (2006) Criteria for comparing requirements variability modeling notations for product lines. The Fourth International Workshop on Comparative Evaluation in Requirements Engineering (CERE'06), in conjunction with RE'06
- Gomaa H (2004) Designing software product lines with UML: from use cases to pattern-based software architectures, Addison-Wesley Professional
- Gomaa H, Shin ME (2002) Multiple-view meta-modeling of software product lines. In: Proceedings of the 8th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'02), pp 238–246
- Halmans G, Pohl K (2003) Communicating the variability of a software-product family to customers. *Softw Syst Model* 2(1):15–36
- Halmans G, Pohl K, Sikora E (2008) Documenting application-specific adaptations in software product line engineering. In: Proceedings of the 20th International Conference on Advanced Information Systems Engineering (CAiSE'2008), LNCS 5074, pp 109–123
- Haugen Ø, Møller-Pedersen B, Oldevik J (2005) Comparison of system family modeling approaches. In: Proceeding of Software Product Lines Conference (SPLC'2005), LNCS 3714, pp 102–112

- Haugen Ø, Møller-Pedersen B, Oldevik J, Olsen GK, Svendsen A (2008) Adding standardized variability to domain specific languages. In: *Proceeding of the 12th International Software Product Line Conference (SPLC)*, IEEE Computer Society, pp 139–148
- Jacobson I, Griss M, Jonsson P (1997) *Software reuse: architecture, process and organization for business success*. ACM Press, New York
- John I, Muthig D (2002) Tailoring use cases for product line modeling. In: *Proceedings of the International Workshop on Requirements Engineering for Product Lines (REPL'02)*, pp 26–32
- Kim J, Hahn J, Hahn H (2000) How do we understand a system with (so) many diagrams? cognitive integration processes in diagrammatic reasoning. *Inf Syst Res* 11(3):284–303
- Kim J, Kim M, Yang H, Park S (2004) A method and tool support for variant requirements analysis: goal and scenario based approach. In: *Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC'04)*, pp 168–175
- Kitchenham BA, Lawrence S, Lesley P, Pickard M, Jones PW, Hoaglin DC, Emam KE (2002) Preliminary guidelines for empirical research. *IEEE Trans Softw Eng* 28(8):721–734
- Korherr B, List B (2007) A UML 2 profile for variability models and their dependency to business processes. In: *Proceedings of the 18th International Conference on Database and Expert Systems Applications*, pp 829–834
- Lazilha FR, Barroca L, Alves de Oliveira E, de Souza Gimenes IM (2004) A component-based product line for workflow management systems. *The IEEE Conference on Information Reuse and Integration*, pp 112–119
- Maßen T, Lichter H (2002) Modeling variability by UML use case diagrams. In: *Proceedings of the International Workshop on Requirements Engineering for Product Lines (REPL'02)*, pp 19–25
- Matinlasi M (2004) Comparison of software product line architecture design methods: comparison of software product line architecture design methods: COPA, FAST, FORM, Kobra and QADA. In: *Proceedings of the 26th International Conference on Software Engineering (ICSE'04)*, pp 127–136
- Moody D (2009) The physics of notations: toward a scientific basis for constructing visual notations in software engineering. *IEEE Trans Softw Eng* 35(6):756–779
- Moon M, Yeom K, Seok Chae H (2005) An approach to developing domain requirements as a core asset based on commonality and variability analysis in a product line. *IEEE Trans Softw Eng* 31(7):551–569
- Morisio M, Travassos G, Stark M (2000) Extending UML to support domain analysis. In: *Proceedings of the 15th IEEE International Conference on Automated Software Engineering (ASE'00)*, pp 321–324
- Nugroho A (2009) Level of detail in UML models and its impact on model comprehension: a controlled experiment. *Information and Software Technology – special issue on Quality of UML Models*, 51(12):1670–1685
- Oliveira Junior EA, Gimenes IMS, Huzita EHM, Maldonado JC (2005) A variability management process for software products lines. In: *Proceedings of the 2005 conference of the Centre for Advanced Studies on Collaborative research*, Toronto, Ontario, Canada, pp 225–241
- Oliveira Junior EA, Gimenes IMS, Maldonado JC (2010) Systematic management of variability in UML-based software product lines. *J Univ Comput Sci* 16(17):2374–2393
- Pohl K, Böckle G, van der Linden F (2005) *Software product line engineering: foundations, principles, and techniques*. Springer, Berlin
- Ramesh V, Topi H (2002) Human factors research on data modeling: a review of prior research, an extended framework and future research directions. *J Database Manag* 13(2):3–19
- Reinhartz-Berger I, Sturm A (2008) Enhancing UML models: a domain analysis approach. *J Database Manag* 19(1):74–94
- Reinhartz-Berger I, Sturm A (2009) Utilizing domain models for application design and validation. *Inf Softw Technol* 51(8):1275–1289
- Reinhartz-Berger I, Tsoury A (2011) Experimenting with the comprehension of feature-oriented and UML-based core assets. In: Halpin T et al. (eds): *BPMDS 2011 and EMMSAD 2011*, LNBP 81, pp 468–482
- Reinhartz-Berger I, Tsoury A (2011) Specification and utilization of core assets: feature-oriented vs. UML-based methods. In: De Troyer O et al. (eds): *ER 2011 Workshops*, LNCS 6999, pp 302–311
- Riebisch M, Böllert K, Streitferdt D, Franczyk B, Ilmenau TG (2000) Extending the UML to model system families. In: *Proceedings of the 5th International Conference on Integrated Design and Process Technology (IDPT)*
- Ripon SH, Talukder KH, Molla KI (2003) Modelling variability for system families. *Malays J Comput Sci* 16(1):37–46

- Robak S, Franczyk B, Politowicz K (2002) Extending the UML for modeling variability for system families. *Int J Appl Math Comput Sci* 12(2):285–298
- Salicki S, Farcet N (2002) Expression and usage of the variability in the software product lines. In: van der Linden F (ed): PFE-4 2001, LNCS 2290, pp 304–318
- Schmid K, Rabiser R, Grünbacher P (2011) A comparison of decision modeling approaches in product lines. In: *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems (VaMoS'2011)*, pp 119–126
- Shanks G, Nuredini J, Tobin D, Moody D, Weber R (2003) Representing things and properties in conceptual modeling: an empirical evaluation. In: *Proceedings of the 11th European Conference on Information Systems*, pp 1775–1785
- Siau K (2004) Informational and computational equivalence in comparing information modeling methods. *J Database Manag* 15(1):73–86
- Siau K, Cao Q (2001) Unified Modeling Language (UML) – a complexity analysis. *J Database Manag* 12(1):26–34
- Sinnema M, Deelstra S (2008) Industrial validation of COVAMOF. *J Syst Softw* 81(4):584–600
- Sinnema M, Deelstra S (2007) Classifying variability modeling techniques. *Inf Softw Technol* 49(7):717–739
- Song IY (2001) Developing sequence diagrams in UML. In: *Proceedings of the International Conference on Conceptual Modeling (ER'2001)*, LNCS 2224, pp 368–382
- Sun C, Rossing R, Sinnema M, Bulanov P, Aiello M (2010) Modeling and managing the variability of web service-based systems. *J Syst Softw* 83(3):502–516
- Svahnberg M, Van Gorp J, Bosch J (2005) A taxonomy of variability realization techniques. *Software-Practice and Experience* 35 (8): 705–754.
- Webber D, Gomaa H (2004) Modeling variability in software product lines with variation point model. *Sci Comput Program* 53:305–331
- Weiler T (2003) Modelling architectural variability for software product lines. In: *Proceedings of the Software Variability Management workshop*, van Gorp, Bosch (eds), pp 53–61
- Witte RS, Witte JS (2009) *Statistics*. Wiley
- Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslen A (2000) *Experimentation in software engineering – an introduction*. Kluwer Academic Publishers, Boston
- Ziadi T, Jézéquel JM (2006) Software product line engineering with the UML: deriving products. In: Käkölä T, Dueñas JC (eds), *Software Product Lines—Research Issues in Engineering and Management*, Springer, pp 557–588
- Ziadi T, Hérouët L, Jézéquel JM (2004) Towards a UML profile for software product lines. In: *Proceeding of Software Product-Family Engineering (PFE'2004)*, LNCS 3014, pp 129–139



**Iris Reinhartz-Berger** is a faculty member in the Department of Information Systems, University of Haifa, Israel. She received her M.Sc. and PhD in information management engineering from the Technion, Israel Institute of Technology and B.Sc. in applied mathematics and computer science from the Technion. Her research interests include conceptual modeling, analysis and design of information systems, software product line engineering, and method engineering. She has chaired a series of Domain Engineering workshops and presented tutorials on domain engineering and variability management.



**Arnon Sturm** is an Assistant Professor in the Department of Electrical Engineering and Computer Science at Wichita State University in Kansas, USA. He received the Ph.D. and M.S. in Computer Science from Kent State University, USA and the B.E. in Computer Engineering from Birla Vishwakarma Mahavidyalaya, India. His research interests are in Software engineering: software evolution/maintenance, mining software repositories, empirical software engineering, source code analysis, and software visualization.