Effects of simulation on novices' understanding of the concept of inheritance in conceptual modeling

Gayane Sedrakyan, Monique Snoeck

Katholieke Universiteit Leuven,
Management Information Systems,
Naamsestraat 69, 3000 Leuven

E-mail:{gayane.sedrakyan, monique.snoeck}@kuleuven.be

Effects of simulation on novices' understanding of the concept of inheritance in conceptual modeling

Gayane Sedrakyan, Monique Snoeck

Katholieke Universiteit Leuven,
Management Information Systems,
Naamsestraat 69, 3000 Leuven

E-mail:{gayane.sedrakyan, monique.snoeck}@kuleuven.be

Effects of simulation on novices' understanding of the concept of inheritance in conceptual modeling

Gayane Sedrakyan, Monique Snoeck

Katholieke Universiteit Leuven,
Management Information Systems,
Naamsestraat 69, 3000 Leuven

E-mail:{gayane.sedrakyan, monique.snoeck}@kuleuven.be

Abstract. In this paper we present our experience in the experimental development and use of simulation instrument for learning object-oriented conceptual modeling in a master level course on analysis and design of information systems. The focus of our research is on the teaching of one particular topic in object-oriented conceptual modeling - *inheritance*. The results from the pilot experimental study (with a student sample N = 32), demonstrate a positive effect of simulation-based learning method on the understanding by novice business analysts of the concept of inheritance when applied in a conceptual model.

Keywords: teaching conceptual modeling, object-oriented analysis, inheritance, simulation-based learning, automated feedback

1 Introduction

Modern software engineering builds largely on object-oriented (OO) paradigm [1, 2] that aims to incorporate the advantages of modularity and reusability. In the OO approach requirements are organized around cooperating objects that belong to hierarchically constructed classes which encapsulate both structure and behavior [3, 4].

The possibility of software reuse during the development lifecycle being not just a matter of reusing the code of a subroutine, but also encompassing the reuse of any commonality expressed in class hierarchies [5], was among the important reasons promoting the rapid growth of this paradigm during the last decades. Consequently object-oriented analysis (OOA) and design (OOD) have emerged to support the use of object-oriented paradigm throughout the entire software engineering lifecycle [2]. This has been supported by the introduction of a unified notation and OO modeling (OOM) language (the Unified Modeling Language (UML)) being currently heavily used in OOA and OOD activities.

One major advantage introduced by the object oriented paradigm is the conceptual continuity across all phases of the software development lifecycle, i.e. the conceptual structure of the software system remains the same, from system analysis down through implementation [5]. Therefore when the object-oriented paradigm is used, the design phase is linked more closely to the system analysis and the implementation phases because designers have to deal with similar abstract concepts (such as classes and objects) throughout software development phases [5]. Conceptual structures are represented through a conceptual model – the first artifact produced in OO analysis. Clearly, the quality of the conceptual model is the foundation of consistency between the requirements and the final software. At the same time continuous efforts are made in the area of OO conceptual modeling, in order to provide reliable and productive software production environments [6].

Teaching requirements formalization through conceptual modeling however has been proven to be challenging [7, 8]. Amongst the factors affecting learning outcomes of novice requirements engineers and business analysts are 1. the complexity of industry tools being "noisy" with various constructs which can result in misusing concepts and creation of unintended models, 2. the lack of domain experience as a result of absence of trial and error rehearsals, 3. the lack of validation techniques and tool support for testing/validating models. Additionally, several researchers correlated novices learning achievements in system's analysis with 4. the lack of technical insights considering the absence of technical components (such as computer-assisted learning) from education as a major contributing factor to the lack of preparedness of their skills [9].

Computer-based simulation has been proven to be an excellent technique assisting juniors in understanding complex systems by allowing them to "learn by experiencing" [10-12]. Simulated environments are also known to promote successful transfer of the skills learned in classroom to real-world environments by allowing to simulate real-life situations where learners improve their technical and problem-solving skills. In the domain of conceptual modeling the use of simulation-based teaching is hampered by at least two shortcomings introduced by the existing standards for simulation technologies. The major disadvantages include being too complex and time consuming to be achieved by novice modelers whose technical expertise is limited [13]. Another important disadvantage is connected with the difficulty of interpreting the simulation results. Our previous work presents significant positive effects on learning achievements of novices for conceptual modeling when using a simulation that is 1. adapted to limited technical expertise of novices using easy and fast ("single-click")

approach to achieve simulation 2. adapted to conceptual modeling goals in which constructs irrelevant for conceptual modeling goals are filtered away, and 3. is enhanced with feedback that links simulation results to their causes in a model design [14-16].

The work presented in this paper builds on our previous research on simulation-based teaching/learning of conceptual models by extending it with OO concepts. While the OO development approach is defined by one of its founders as a *"hierarchy of reusable classes united via inheritance relationships"* [2, 3], this perspective is largely neglected in literature on simulation-based teaching of OO system analysis and modeling. More specifically, in this work we target at a simulation technique that allows novices to master the concept of *reusability that can be exploited by means of class inheritance* – a key OO concept, the semantics of which is among the most challenging to be mastered by novices [17, 18], while inheritance being also often avoided [19] and/or a misapplied modeling construct [20, 21]. The effectiveness of proposed method is evaluated with respect to comprehension by novices of the concept of *inheritance when applied in a conceptual model*. The results of the experimental study show a positive impact of the proposed technique on learning outcomes of novices.

The remainder of the paper is structured as follows. The second section describes the educational context and assumptions used within this paper. Section 3 gives a brief overview of the simulation environment subsequently highlighting the learning benefits of the proposed method. Section 4 describes the experimental study targeting to measure the effects of the proposed simulation technique on the learning outcomes of novice modelers, followed by the data analysis and subsequently reports on the results. Finally, section 5 concludes the work proposing some future research directions.

2 Educational context

The proposed simulation method has been developed and validated within the course "Architecture and Modeling of Management Information Systems"[1] over a 5-years period of teaching, with participation and constant feedback from 500 students overall. The course targets at master level students with heterogeneous backgrounds from the Management Information Systems program. The goal of the course is to familiarize the students with modern methods and techniques of Object-Oriented Analysis and Design for Enterprise Information Systems, to let them understand the relation between an information system and the organizational aspects of an enterprise, and to let them acquire sufficient skills of developing an enterprise model as basis of an enterprise information system. During the course students have to formalize business requirements into conceptual domain models using an adapted for conceptual model-

[1] The course page can be found on http://onderwijsaanbod.kuleuven.be/syllabi/e/D0I71AE.htm

ing and simulation environment JMermaid[2]. The methodology uses the UML as modelling language, but underneath it relies on the concepts of MERODE[3], an Enterprise Information Systems engineering methodology developed at the university of Leuven, which follows the Model-Driven Architecture and Engineering approach. In MERODE a conceptual model integrates both structural and behavioral views of a system to be engineered using a restricted class diagrams with regular binary associations, multiple interacting state charts and an interaction model. Throughout a modeling process self-regulated activities, such as testing and validation of models, are promoted through model simulation using MERODE's semantic prototyper [16] which allows simulating model solutions using a "one-click" approach. In this paper we will refer to simulation of a conceptual model as a process of generating prototype applications using a conceptual model as an input. We will therefore use the terms "simulated model" and "prototype" interchangeably. The simulation effects on the learning outcomes of novice modelers are measured with respect to *understanding of model semantics* as defined in the Conceptual Model Quality Framework [22].

3 Extending simulation model with the concept of inheritance

MERODE's model simulation environment has been extended to support the key concepts of object-oriented approach. 1. classes are created in hierarchies, and *inheritance* allows the *structural features (attributes)* and *behavioural features (methods)* to be passed down the hierarchy 2; this is realized through the concepts of *concrete classes* (classes which can be instantiated) and *abstract classes* (classes that have no instances but are used for creating other classes via *inheritance*).

[2] http://merode.econ.kuleuven.ac.be/mermaid.aspx

[3] MERODE is an Object Oriented Enterprise Modeling method. Its name is the abbreviation of Model driven, Existence dependency Relation, Object oriented DEvelopment. Cfr. http://merode.econ.kuleuven.be

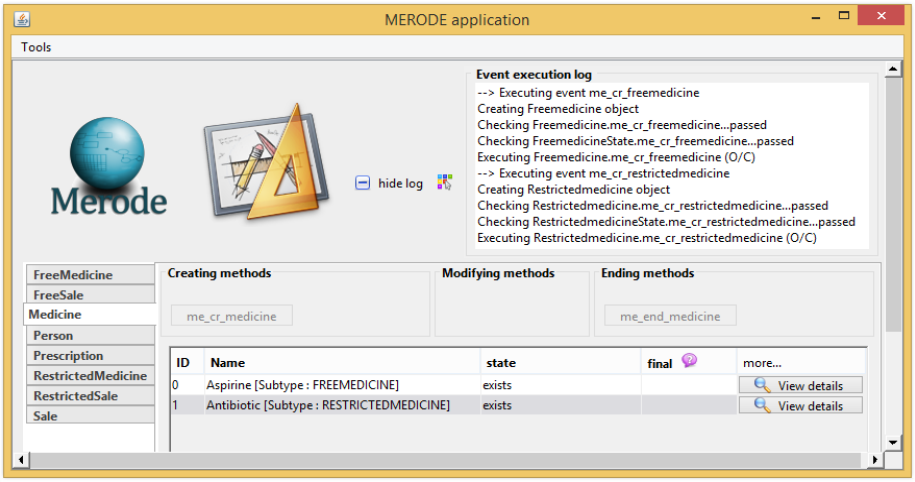

Fig. 1. Interface showing a tab for an abstract superclass

Understanding the interface (input and output models) of the generated prototype is quite intuitive. The graphical interface of a prototype application includes a main window and a set of input/output popup windows. Business entities are presented across tabbed views in the main window each containing corresponding properties of object instances (such as attributes and associated objects) presented in a tabular format. Each view of a tab panel also contains buttons corresponding to the business events that can be triggered for a particular class. MERODE prototypes offer basic functionality like triggering the creating and ending of objects, and triggering other business events that returns the output to a user in a passed/failed format. The tabs representing abstract classes are in disabled mode except for viewing buttons. An attempt to trigger any business event for this class is followed by an explanation message about the concept of 'abstract class'. The list of instances has an indication of the subclass name for each subclass instance. **Fig. 1** shows the main interface of the prototype: the tab for abstract class "Medicine" with disabled functionality includes instances "Aspirine" that belongs to the subtype "FreeMedicine" and "Antibiotic" that belongs to the subtype "RestrictedMedicine".

Inherited (from a supertype) and specialized (further extended by a subtype) methods are defined by a modeler in JMermaid environment. Inherited methods are those methods that have been inherited from a supertype without changing their signature, whereas specialized methods are new methods that "extend" the subclass with additional features. Tabs for subtype classes show buttons both for the inherited and specialized methods (see **Fig. 2**).

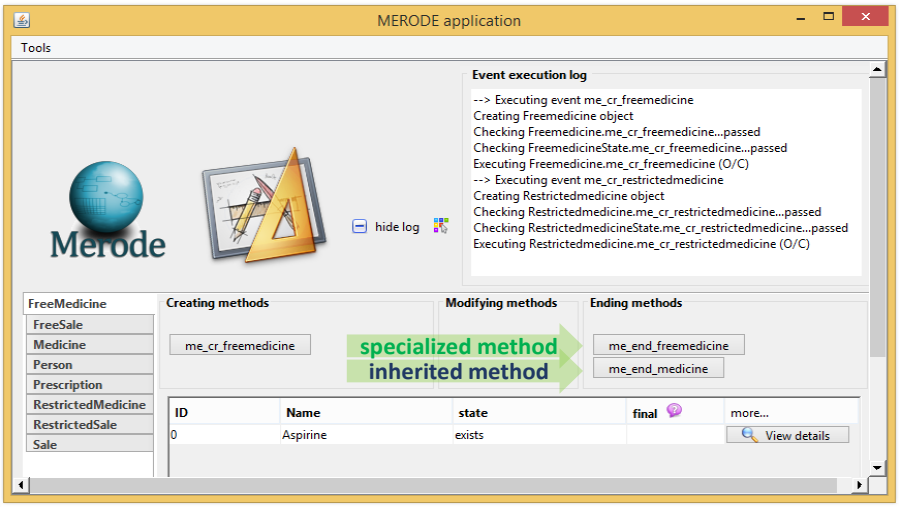

Fig. 2. Interface showing a tab for an subclass with inherited and specialized methods

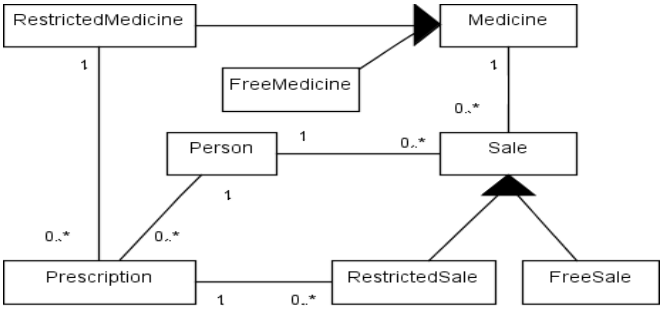

Fig. 3. Sample erroneous model

The entire interaction process is guided by user-friendly messages in case of an invalid input or a failure of an event execution (e.g. creating, ending or modifying object instances) thus ensuring maximum transparency between a prototype and its design model. A sample erroneous model and validation scenario is described in **Fig. 3**. Notice that in the diagrams abstract classes have been graphically represented by a black filled triangle rather than by the textual keyword {abstract} as in UML.

An example of a modeling task would be to validate a given model solution for the requirement "*To buy a restricted medicine a customer needs to have a prescription. However a prescription is not required for buying a free medicine. Registering a buyer's identity is not required for selling a free medicine. Both free and restricted sale can have a shared behavior, e.g. undergo a promotion,… . Likewise free and restricted medicine can both be out of stock, removed from sale, have a delivery request, expire,…*".

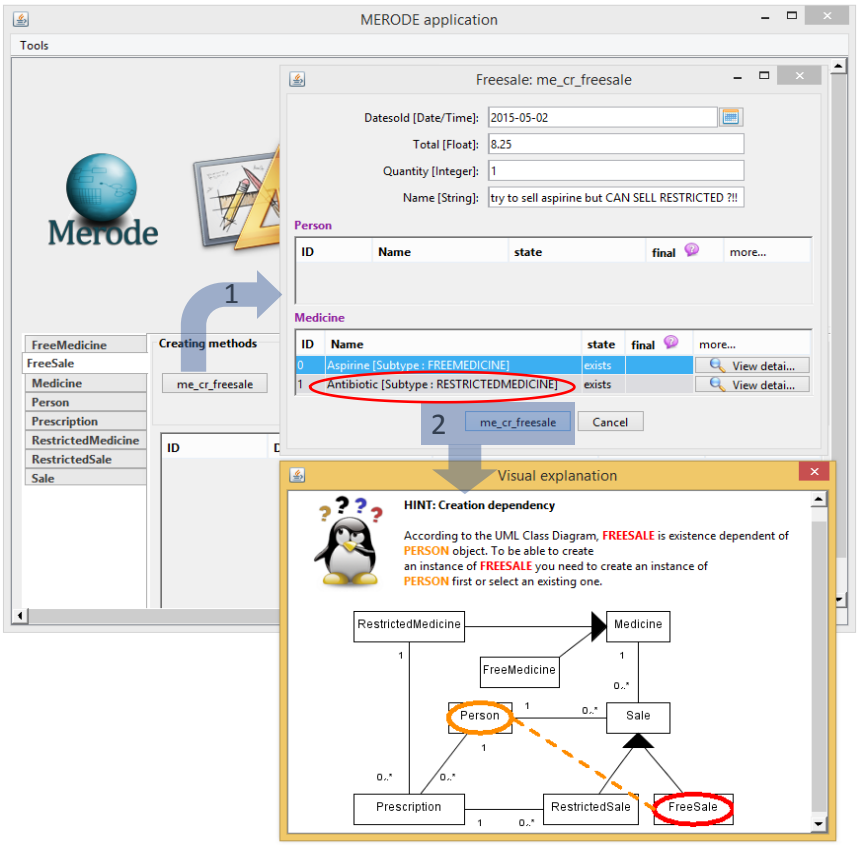

Fig. 4. Validation through a simulated model with a sample feedback on mandatory one rule violation for an association linked to the supertype class

When testing a model's prototype a student will be confronted with the following scenario: trying to register a free sale for “Aspirine” (according to a designed model this will be by means of triggering a creating event of a ‘FreeSale’) a popup window will request an input for the attributes specified in the model as well as to choose instances of the associated mandatory objects “Medicine” and “Person”. As a result, a first problem that the prototype will allow to discover is that the selling of a free medicine requires a person to be associated with it as a result of the mandatory relationship between its superclass “Sale” and the class “Person” (cardinality of [1..1]). This is clearly in contradiction with the requirement that registering a buyer's identity is not required for selling a free medicine. A second problem that the student will discover, is that -as a consequence of the Liskov principle of substitutability- while choosing the medicine to associate with the newly created instance of the “FreeSale” object also instances of the subtype “RestrictedMedicine” will be available as potential substitutes for the supertype “Medicine”, thus enabling an unrestricted sale for a restricted medicine. Through such testing, a student's analytical problem-solving ability is

stimulated to pursue a correct design solution in order to fix the detected error in the model. While iteratively improving a model solution and testing with a simulated model a student is involved in a self-regulative learning process through what-if scenarios and trial and error rehearsals that allows him/her to not only achieve better but also allowing to gain knowledge that emerges from own practice. The generated feedback facilitates the process of achieving the intended behavior by explaining the reasons each time the execution of intended behavior is refused as illustrated in **Fig. 4**.

4 Evaluation method

Our research goal was to assess the effectiveness of such feedback-enabled simulation in improving the novices understanding of the semantics of inheritance in a conceptual model. We opted for an experimental study with a pre/post-test control group experimental design.

Procedure: The experiment was conducted in two parts. In the first part students had to answer the set of TRUE/FALSE questions without the use of the model's prototype, so only by means of manual inspection of a given model solution. The goal of this part was to establish a baseline model validation capability level to measure the simulation effects in the second cycle. Then, in the second part of the experiment, analogous questions about a similar model had to be answered again, this time with the use of the generated prototype. The answers had to be recorded on an answer sheet. We will further refer to cycles without simulation (a paper exercise by means of manual inspection of a given model to answer the test's questions) as "withoutPT", and cycles with the use of simulation (students were required to use laptops to run a simulated model of a given model solution to answer the test's questions) as "withPT". The effectiveness of the proposed simulation method was measured by means of comparison of the test results of students between experimental cycles (without and with a use of a simulation).

Observable dimensions: Assessing the semantic correctness of a model requires a combination of model understanding and comparing model statements with requirements. Model-reading knowledge specifically for inheritance can be assessed at different levels of understanding. In this pilot study we target at understanding of the semantics of a single level inheritance, hidden dependencies through chain of associations and parallel paths via subclass and superclass:

Level 1: Understanding the difference between the concepts of abstract and concrete classes.

Level 2: Understanding the basic concept of inheritance: that attributes and methods are inherited by the subtype from the supertype. This includes understanding the direction of the inheritance relationship.

Level 3: Understanding a single inheritance hierarchy (inheritance of associations): through inheritance and the Liskov principle of substitution, objects of the subclass can participate in associations defined at the level of the superclass (e.g. in **Fig. 4**, a subclass "FreeSale" inherits the association to "Person" from supertype "Sale").

Level 4: Understanding complex models with more than one inheritance hierarchies connected with a single (chain of) association.

Level 5: Understanding complex models with more than one inheritance hierarchies connected with multiple parallel (chains of) associations (e.g. in **Fig. 4**, the hierarchy of "Sale" and the hierarchy of "Medicine" are directly connected via the association between "Sale" and "Medicine" classes, but also through "Person" and "Prescription").

Minimum 2 questions per each dimensions have been included in the tests.

Gender	*Male*	59 %
	Female	41 %
Age distributions	*Min age*	22 y.
	Max age	42 y.
	Mean age	24.6 y.
Previous knowledge of data modeling	*No knowledge*	19 %
	Little knowledge	28 %
	Moderate knowledge	38 %
	Extensive knowledge	15 %

Table 1. Summary of sample demographics

Participants: Students who participated in the experiments were final year master students from Management Information Systems programs at KU Leuven. Overall 32 students participated in the experiment. Analysis of the pre-experimental test for basic data modeling skills and the context information from the post-study questionnaire resulted in the demographics presented in **Table 1**. The normality of distributions of pre-experimental knowledge as well as the experimental test scores were confirmed.

Data Analysis and findings: Data has been analyzed by means of statistical comparison of mean scores among the experimental cycles. The effectiveness was assessed based on the relative average advantage (positive correction). The results of the paired T-Test comparing mean scores of withoutPT and withPT experimental cycles ($\overline{X}_{withoutPT} = 6.04$, $\overline{X}_{with} = 8.37$, $\overline{X}_{difference} = 2.33$, p-value = 0.000) provide evidence that the prototyping method was effective in producing positive correction with regard to students' understanding of the concept of inheritance applied in a conceptual model. At individual test level the corrections in scores varied from 0 up to 9 (out of total 12). Despite a tool's usefulness, insufficient user acceptance can however be another factor affecting learner performance (Venkatesh, et al., 2003) such ease of use (EU) referring to the required effort to interact with a system and perceived usefulness (PU). The ratings collected in a post-study questionnaire used to assess students perceptions on a 6-point Likert scale reflect positive perceptions (Mean EOU = 4.7/6, Mean PU = 5.15/6) on the simulation environment. Students also reported a high perceived utility on the inclusion of feedback to execution failures in the prototype (5.75/6).

5 Conclusion and future work

The work reported on experimental extension of a feedback-enabled simulation of conceptual models with the concept of OO inheritance. The results of our pilot study described in this work show a positive effect of the proposed learning method on novices understanding of the concept of inheritance when applied in a conceptual model. While the simulation environment presented in this work uses regular binary associations in a class diagram, expanding the simulation environment with other concepts such as associative classes, composition/aggregation as well as broader coverage for inheritance with more advanced concepts such as multi-level inheritance are in the domain of future research. Among possible further directions of this research a replication experiment in the context of other university study programs with an improved experimental design (e.g. factoral 4 group experiment) is considered. In addition extending feedback framework specifically for inheritance can be another direction for this research. Yet further research must be conducted towards methodologies allowing better consistency between different abstraction levels used in a conceptual model and more detailed design models used for lower level specifications and code implementation.

6 References

1. Vazquez, G., J.A.D. Pace, and M. Campo, *Reusing design experiences to materialize software architectures into object-oriented designs.* Information Sciences, 2014. **259**: p. 396-411.
2. Booch, G., *Object Oriented Analysis & Design with Application*. 2006: Pearson Education India.
3. Booch, G., *Object-oriented development.* Software Engineering, IEEE Transactions on, 1986(2): p. 211-221.
4. Northrop, L.M., *Object-Oriented Development.* Encyclopedia of Software Engineering, 1994.
5. Capretz, L.F., *A brief history of the object-oriented approach.* ACM SIGSOFT Software Engineering Notes, 2003. **28**(2): p. 6.
6. Pastor, O., et al. *Linking object-oriented conceptual modeling with object-oriented implementation in Java*. in *Database and Expert Systems Applications*. 1997. Springer.
7. Siau, K. and P.-P. Loo, *Identifying Difficulties in Learning Uml.* Information Systems Management, 2006. **23**(3): p. 43-51.
8. Erickson, J. and S. Keng. *Can UML Be Simplified? Practitioner Use of UML in Separate Domains.* in *Proceedings of the 12th Workshop on Exploring Modeling Methods for Systems Analysis and Design (EMMSAD'07), held in conjunctiun with the 19th Conference on Advanced Information Systems (CAiSE'07),Trondheim, Norway*. 2007.

9. Barjis, J., et al., *Innovative Teaching Using Simulation and Virtual Environments.* Interdisciplinary Journal of Information, Knowledge, and Management, 2012. **7**: p. 237-255.
10. Kluge, A., *Experiential learning methods, simulation complexity and their effects on different target groups.* Journal of educational computing research, 2007. **36**(3): p. 323-349.
11. Damassa, D.A. and T. Sitko, *Simulation Technologies in Higher Education: Uses, Trends, and Implications.* EDUCAUSE Center for Analysis and Research (ECAR), Research Bulletins, 2010.
12. EuropeanCommission, *Opening up education: Innovative teaching and learning for all through new technologies and open educational resources. Communication from the commission to the European parliament, the council, the European economic and social committee and the committee of the regions.* 2013.
13. Sedrakyan, G. and M. Snoeck. *A PIM-to-Code requirements engineering framework.* in *Modelsward 2013-1st International Conference on Model-driven Engineering and Software Development-Proceedings*. 2013.
14. Sedrakyan, G. and M. Snoeck, *Lightweight semantic prototyper for conceptual modeling*, in *Advances in Conceptual Modeling*. 2014, Springer. p. 298-302.
15. Sedrakyan, G. and M. Snoeck, *Technology-enhanced support for learning conceptual modeling*, in *Enterprise, Business-Process and Information Systems Modeling*. 2012, Springer. p. 435-449.
16. Sedrakyan, G., M. Snoeck, and S. Poelmans, *Assessing the effectiveness of feedback enabled simulation in teaching conceptual modeling.* Computers & Education, 2014. **78**: p. 367 - 382.
17. Liberman, N., C. Beeri, and Y. Ben-David Kolikant, *Difficulties in learning inheritance and polymorphism.* ACM Transactions on Computing Education (TOCE), 2011. **11**(1): p. 4.
18. Hadar, I. and U. Leron, *How intuitive is object-oriented design?* Communications of the ACM, 2008. **51**(5): p. 41-46.
19. Sedrakyan, G., M. Snoeck, and J. De Weerdt, *Process Mining Analysis of Conceptual Modeling Behavior of Novices - empirical study using JMermaid modeling and experimental logging environment (accepted).* Computers in Human Behavior, 2014. **41**(C): p. 486-503.
20. Rumbaugh, J., *Disinherited-Examples of misuse of inheritance.* Journal of Object-Oriented Programming, 1993. **5**(9): p. 22-24.
21. Deligiannis, I.S., et al., *A review of experimental investigations into object-oriented technology.* Empirical Software Engineering, 2002. **7**(3): p. 193-231.
22. Nelson, H.J., et al., *A conceptual modeling quality framework.* Software Quality Journal, 2012. **20**(1): p. 201-228.