On the Syntactic, Semantic, and Pragmatic Quality of Students' Conceptual Models

Justin MacCreery, Bastian Tenbergen*
State University of New York at Oswego
justinmaccreery@gmail.com | bastian.tenbergen@oswego.edu*
* corresponding author

Abstract

Visual notations and conceptual models, such as ER diagrams or UML diagrams aid in aligning stakeholder needs, defining and prioritizing processes and goals for the system under development, serve as a reference for requirements elicitation, negotiation, and enable validation as well as verification of artifacts. With such a ubiquitous presence and paramount importance, conceptual models have therefore been introduced in software engineering curricula far and wide. However, it is exceedingly difficult to teach and learn conceptual modeling. Not only does it require educators to instruct notation and syntax of the visual language, but also semantic intricacies. Similarly, students struggle with what differentiates a "good" conceptual model from an inadequate one, how to use conceptual models of different types in conjunction with one another in a meaningful way, or simply how to avoid ambiguity and vagueness. In this paper, we discuss the syntactic, semantic, and pragmatic quality of conceptual models in four courses from an undergraduate software engineering program. It is not our aim to present empirically rigorous results, but to contribute to the body of knowledge on the quality of typical novices' conceptual models. We seek to foster discussion in the community and present observations and results for comparison.

1. Introduction

It has been over 20 years since the need to standardize visual languages has been identified in software engineering [3]. Albeit natural language remains the most widely used form of documenting software engineering artifacts [14, 18, 19], conceptual models and visual notations are gaining importance for academia [4] and industry [19, 22]. In fact, conceptual models play a significant role in aligning stakeholder needs for a system under development, guide requirements elicitation, negotiation, and validation, help in conceiving system architectures, enable code generation, and allow for formal verification [22].

It is therefore not surprising that there have been increasingly strong arguments to incorporate conceptual modeling into undergraduate software engineering curricula at the university level [7, 13, 16]. Yet, teaching conceptual modeling to students and, of course, learning the intricacies of conceptual models as a student, are daunting tasks [21]: students must learn the notation, syntax, the meaning of notational elements, as well as the meaning of the diagram. Moreover, instructors must deal with vagueness and uncertainty in student models, offer and discuss appropriate modeling alternatives, and find ways to enable students to select the right level of abstraction.

Yet, often, conceptual modeling instruction is done as a by-product of courses on software engineering processes, architecture, or requirements engineering (see, e.g., [5]). This means that students must become familiar with the inception process while they are learning about conceptual modeling. While this may be successful in some cases, there is the danger that either conceptual modeling, software engineering processes, or both are learned only superficially, or not at all. Albeit some reports on the experiences with and avenues to improve the quality of students' conceptual models in software engineering education are available (e.g., [5, 11]), to the best of our knowledge, very little quantifiable data on model *content* is available.

Therefore, the purpose of this paper is to contribute to the body of knowledge on students' conceptual model quality, typical syntactic and semantic errors, and instructional experiences. We hope that this paper sparks discussions in the community and helps fellow educators tailor their instructional approaches to instill good modeling practices in software engineering students.

The following Section 2 discusses the related work with regard to model quality and reviews some instructional approaches and experience with conceptual modeling. Section 3 introduces the study design. Section 4 reports on qualitative and quantitative findings in student model quality as well as our experiences in instructing conceptual modeling. Section 5 draws conclusions and Section 6 discusses threats to validity. Section 7 concludes this paper.

URI: https://hdl.handle.net/10125/60209
ISBN: 978-0-9981331-2-6

2. Related Work

In the following, we review the central notion of model quality (Section 2.1), and report on experiences and approaches in instructing conceptual modeling (Section 2.2).

2.1. Conceptual Model Quality

Alongside the unification of largely divergent conceptual model types into UML, work was undertaken to answer the question what differentiates a "good" from a "poor" conceptual model. One of the earliest works to investigate this question was done by Lindland et al [17], who stipulate that the quality of a conceptual model consists of three aspects:

- **Syntactic quality** describes the correspondence of a diagram with the notational rules of the modeling language. For example, syntactic quality is impaired when sharp-cornered rectangles (like in UML classes) are used to depict UML actions and activities (instead of rounded-cornered rectangles).
- **Semantic quality** describes the correspondence of a diagram with the semantic domain that is being depicted. For example, semantic quality is impaired when UML class diagrams contain one or more associations labeled "has" or "is a," instead of aggregations, compositions, or generalizations.
- **Pragmatic quality** describes the correspondence of a diagram for its intended purpose. For example, pragmatic quality is impaired if a modeler uses a UML class diagram instead of a sequence diagram to document interactions between components.

These types of quality depend on one another to a large degree. For example, if the wrong notation is used (syntactic), the diagram may become ambiguous (semantic), which in turn might mean that the diagram is no longer useful during development (pragmatic). Specifically, the question of usefulness for development has been a concern for quite some time. To this end, several frameworks and approaches have been proposed, less to assess quality, but more to prescribe beneficial use of diagrams. Examples include Kruchten's renowned 4+1 View Model [12], the SPES Modeling framework [4], but also approaches suggested in software engineering textbooks that focus on the use of UML and other conceptual modeling languages (see, [15], for just one example). Yet, these approaches and frameworks focus on the consistency between diagrams as well as their use for a development project, and less on the quality of the diagrams themselves.

2.2 Experiences with and Approaches for Conceptual Model Instruction

Improving the way of instructing software engineering methods, skills, processes, and conceptual modeling at the undergraduate level is a core endeavor of many educators. A plethora of reports exist which reflect on the experiences made therein. However, most of these reports mainly reflect on either teaching software engineering processes (e.g., [5, 7]), formal methods (e.g., [9]), or how to use conceptual modeling to improve the instruction of basic Software Engineering skills, such as programming (e.g., [1, 16]).

Others reports on the qualitative and quantitative quality of student models. For example, in [11] novice modelers were asked to interpret and create class diagrams. Their performance was rated against an ideal solution. Results show approximately 30% error rate in model interpretation. During creation, an error rate of between 14% and 38.6% was reported, mainly due to attribute related errors.

In [23], a report on the modeling process itself is given. Using an online experiment, novice modelers' way of creating visual diagrams was assessed. Results show that both a "depth first" or a "breadth first" modeling approach is feasible. However, one key aspect that fosters understandability within the model is the layout of the model. Layout correlates with the grades received on models by approximately 32%. Yet, concrete numbers on the content of the model elements used by students are not given.

A comparison of model understandability between student and expert modelers is reported in [10]. Both expert and student modelers were asked to self-evaluate and peer-evaluate diagrams. Results show that peer-evaluated student diagrams are significantly similar to expert judgments. Moreover, when assessing model quality, understandability, layout, and completeness are rated as the most significant.

It is to note that [10, 11, 23] predominantly consider class diagrams. In fact, UML class diagrams appear to be the most often studied conceptual models in regards to novice modeler's model quality. A notable exception is [20], where a sample solution-based tutoring system was used to critique students' UML activity diagrams, albeit the study in [20] considers UML class diagrams as well.

While reports on students' ability to solve a modeling task or acquire expertise are valuable, quantitative measurements about the specific *content* of student models in free-modeling scenarios are largely absent. In other words, it is largely unknown how students perform, when the task is to produce a conceptual model for a hypothetical system under development, when no ideal solution available.

3. Study Design

Considering that for modeling tasks in industrial settings, instructor-provided ideal solutions are seldom available, we seek to understand how students perform when tasked with freely inventing system properties. Therefore, with this paper, we investigate the content of students' conceptual models, specifically UML diagrams, syntactic errors, and diagram interrelation. We gathered data about student models in four different undergraduate software engineering courses over two years of instruction. In this section, we present our study approach and research question (Section 3.1), introduce the courses in which we recorded data (Section 3.2), and explain our analysis procedure (Section 3.3). Section 4 reports on results.

3.1. Approach and Research Questions

To investigate the content, syntactic errors and subjective quality of students' conceptual models, we applied Glaserian Grounded Theory (GGT, see [8]) and followed the guidelines for applying grounded theory in software engineering research, as outlined in [24]. As mentioned in Section 2.2, students' modeling performance is usually evaluated against some ideal solution. In this research, we were interested in how students perform in free-modeling tasks, i.e. when asked to conceive the requirements for some system under development, from scratch. Our underlying theoretic starting point is the idea that the quality of conceptual models will have an effect on the entire development process. This means that not only the finished product, but also intermediate artifacts are influenced by the quality of other artifacts [2]. However, what a "poor," "better," or "good" conceptual model is cannot easily be established. We built our grounded theory with regard to the qualities from [17], by investigating three research questions:

- **RQ1: What quantifiable properties and errors do student models present?** The purpose of this RQ is to allow gauging the average model size and complexity of student's models, wrt. the typically used language. As size and complexity can change with diagram type, this RQ informs interpretations of the following RQ2 and RQ3.
- **RQ2: What is the semantic quality of the student models?** The purpose of this RQ is to gauge how the perceived (subjective, but quantified) quality of the models are, despite errors, but in light of the number and nature of contained modeling elements.
- **RQ3: What is the pragmatic quality of the student models?** The purpose of this RQ is to gauge continuity between models, i.e. how elements from one diagram appear in other diagrams and thereby make up the specification of one system under development. We theorize that only diagrams that "fit together" are useful for implementation.

3.2. Data Collection & Course Descriptions

Data was collected in four undergraduate software engineering courses at the State University of New York at Oswego (SUNY Oswego). Each course focused on the use of diagrammatic representations and conceptual models to specify a hypothetical system under development in a semester-long case study project. The following subsections describe the courses, properties of the students, and how conceptual models were used. Unless otherwise specified, all courses meet three times a week for 55 minutes over the course of 15 weeks. Each course can be taken for elective credit in software engineering (SE), computer science (CS), and information science (IS) baccalaureate degree programs at the institution. In each course, the SPES Requirements Viewpoint [6] and the diagram types discussed therein were taken as the example pattern to follow during specification. The study design was approved by the institution's ethics review board and students supplied informed consent.

3.2.1 Software Engineering (CSC380)

This course is an introductory course on software engineering processes, the development life cycle, and conceptual modeling. Half the course is dedicated to lecturing, giving detailed introductions into notation and beneficial use of UML for the development process. The other half is dedicated to developing and implementing the specification of a project, with weekly meetings being dedicated to presentation, discussion, implementation, and revision of artifacts.

The course is a core requirement for SE undergraduate students at the institution and may be taken for elective for CS and IS degree programs. Moreover, this course is a prerequisite for all other courses and takes place every semester. Data was gathered in this course in fall 2017, where 40 students (second year and higher) enrolled, separated into eight groups of five to eight students.

3.2.2 Software & Safety Requirements Engineering (CSC436)

This course is an advanced course on development of safety-critical embedded software, with particular focus on requirements engineering, safety assessment, and safety argumentation in early stages of development. During a semester-long industrially realistic project, students produce and implement a

requirements specification consisting of the aforementioned diagram types from [6], enhanced with natural language requirements. Class meetings are dynamically allocated to presentation, discussion, and improvement of diagram artifacts as well as lecturing. Content delivery in lectures is aligned with project milestones and student group progress.

The course takes place every spring semester. Data was gathered in spring 2017, where 24 students (third year and higher) enrolled, separated into four groups of six students.

3.2.3 Software Design (CSC480)

This course is an advanced course, in which the entire software development process of a start-up company is simulated. During a semester-long project, all students in the course work together to produce and implement a requirements specification consisting of the aforementioned diagram types, enhanced with natural language requirements. Students are responsible for quality assurance (i.e., unit testing, usability testing, etc.). Class meetings are exclusively dedicated to daily SCRUM meetings, progress planning, discussion of artifacts, and conflict resolution (when necessary). There are no classic lectures.

The course is co-listed as a core requirement for graduate students in the institution's Human Computer Interaction degree program. Like CSC380, it is also a core requirement for SE undergraduate students and can be taken for elective credit in CS and IS. The course takes place every spring semester. Data was gathered in spring 2016, where 30 students (15 graduates and 15 third year and higher) enrolled. Students are separated into requirements, quality assurance, usability, engine, database, and user interface groups, consisting of three to six students.

3.2.4 Software Engineering Capstone (CSC495)

This course is an advanced course, in which the contracted software development process is simulated. Like in CSC480, a requirements specification is produced, implemented, and quality assured in a semester-long project. However, unlike CSC480, students work by themselves, or in small groups. Class meetings are dedicated to progress reporting, planning, discussion of artifacts, and demos of the current state of the project. There are no classic lectures.

Like CSC380, the course is a core requirement for SE undergraduate students and can be taken for elective credit in CS and IS. The course takes place every semester. Data was gathered in fall 2016, where eight students (third year and higher) enrolled, separated four teams of one to three students.

3.3 Analysis Procedure

The object of study were the diagrams produced in each course. The diagram types discussed and created in all courses comprised UML sequence and class diagrams. More activity and state machine diagrams were discussed in CSC380, CSC436, and CSC480, but were excluded from CSC495 due to time constraints. The final version for each diagram in each course was collected after the respective course had concluded. Subsequently, each diagram for each course was subjected to the data analysis procedure is shown in Fig. 1. The steps are explained in the following.

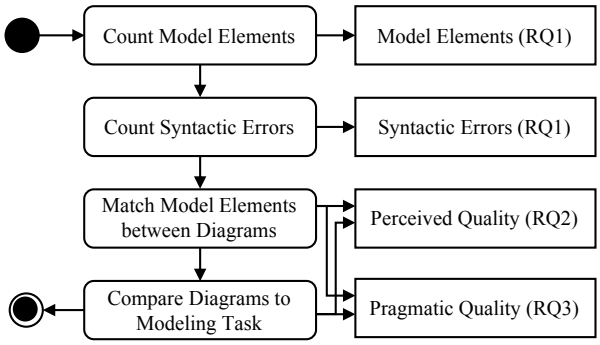

Fig. 1 Data Analysis Procedure and RQ Dependencies

Count Model Elements. The first step consisted of counting the modeling elements for each diagram by each group in each course. This entailed, for example, counting the number of classes, attributes, operations, associations, etc. in class diagrams (and equivalently in other diagram types). This informed RQ1.

Count Syntactic Errors. This step involved looking for explicit syntax errors within each diagram by each group in each course. For example, it was counted as a syntactic error, if rounded-edges (which in UML is reserved for actions and states in dynamic diagrams) are used for classes in class diagrams (which are static). For another example, it was counted as a syntactic error, if a decision node in activity diagrams contained only one outgoing edge, was missing guards on the outgoing edges, etc. This also informed RQ1.

Match Model Elements between Diagrams. Diagrams produced by each group (in each course) were then compared to the other diagrams by the same group (in the same course). The goal was to ascertain how the diagrams fit with each other. This was done by checking for name identity of modeling elements (e.g., names of lifelines in sequence diagrams, class names, operations in class diagrams, or activity names). For example, a sequence diagram describing the interaction between the system with its context is expected to contain lifelines, which (depending on the system under development) would also be present in the implementation or in a class diagram describing the

internal structure and external interfaces of the system. During this step, whether or not model element labels and diagram purpose were adequately applied was also checked. For example, if a sequence diagram was used to describe the internal structure, this was counted as inadequate (since the purpose of sequence diagrams is to highlight component interaction, not structure). For another example, if a UML class was labeled using a verb, this was also considered inadequate. Together with the previous steps, this step allows understanding specific project specification produced by each student group and hence build our theory of conceptual model quality. This step informs RQ2 and RQ3.

Compare Diagrams to Modeling Task. Finally, the diagrams produced by each group in each course were compared to the respective milestone and/or assignment, under consideration of UML's notational rules and check rules between diagrams from [6]. Unlike the previous step, it was ascertained to what degree the diagrams satisfy the requirements outlined in the individual course. This was done by assigning a numeric value between -1 and 1 to a diagram, if a certain requirement from the modeling task was met. For example, a standing requirement in all courses was for class diagrams to "define all relevant details for implementing and testing the system." If 75% or more of the details were found in the implementation, the conceptual model scored a 1; if less then 75%, but more than 25% of the conceptual model details were relevant to implementing and testing, a score of 0.5 was assigned. If 75% or more of the conceptual model details were not relevant to implementing or testing, the score of -1 was assigned. Otherwise, 0 was assigned. In addition, in this step, consistency between the diagrams of the same group was recorded. This entailed the number of modeling elements from RQ1, which were used in two or more *types* of diagrams within a project as well as their representation in the final implementation. This step informs RQ2 and RQ3.

4. Results

In the following, we present results from our research questions from Section 3.1.

4.1 RQ1: Quantifiable Model Properties

As we have illustrated in Section 3.2, UML sequence, class, activity, and state machine diagrams were used in the courses. The tables show the average and standard deviation of modeling elements in all courses. If a standard deviation cannot be computed, "n/a" is noted. Empty cells indicate that the measurement is not applicable; missing fractions indicate precise number (and not an average). If a language feature was not used, the corresponding row is missing. For example, in Table 2, the fact that a row for "property-strings" for UML class attributes is missing indicates that this language feature was not used in any course. Due to the fact that for many courses, standard deviations cannot be computed (since only one group used some language feature), statistical hypothesis testing was inapplicable.

Table 1 shows the modeling elements and syntactic errors for **sequence diagrams**. Albeit all courses instructed students to use the entire feature set of sequence diagrams, students focused almost exclusively on lifelines and interactions between them. Partitions, i.e. "par," "break," "loop," "alt," "opt," or "ref" were not employed by most groups. A notable exception is one group in CSC380, who used two loops in their diagrams. Moreover, two groups made frequent use of "alt" and "loop" in CSC495. Only two syntactic errors were found in the same group in CSC380. This can likely be explained with the fact that this group chose to use only a single sequence diagram, of a particularly large size. It featured a total of 49 modeling elements (messages, lifelines, and partitions). By comparison, in CSC480, a total of nine sequence diagrams with between seven and fifteen modeling elements were created. The lowest number of modeling elements was found in one sequence diagram in CSC495, where only three modeling elements (two lifelines and one message) were employed.

Table 1 Modeling Elements in Sequence Diagrams

	CSC380	CSC436	CSC480	CSC495
Lifelines	11.75 (10.42)	3 (n/a)	3.75 (0.96)	6.79 (4.89)
Messages	24.50 (16.03)	8 (n/a)	3.63 (1.85)	2.79 (1.40)
Partitions (e.g., loop, break)	4 (n/a)	1 (n/a)		1.63 (1.06)
Errors	2 (n/a)			

Table 2 shows the modeling elements and syntactic errors for **class diagrams**. It can be seen that in CSC480, the student team decided not to use class diagrams. Albeit in that semester, a database was developed and students were encouraged to make use of a class diagram to show the internal structure of the program, students decided not to follow this suggestion. In terms of used language features, it can be seen that students in CSC380 and CSC495 employed the largest variety. In fact, only students in CSC380 used association classes, multiplicities, and generalizations. Aggregations and composition were used by students in all remaining courses, however only relatively few such modeling elements were included. It is also noteworthy that not only did students in CSC380 use the widest variety of language features, the diagrams themselves were also larger in

size, as can be seen by the higher means across the board. In general, the minimum number of classes in the diagrams was five and the maximum was 22 with the average being approximately nine. The number of attributes ranged between six and 110 with (avg. ca. 27). Similarly, the range in the number of associations was between two and 218 (avg. ca. 9).

Table 2 Modeling Elements in Class Diagrams

	CSC380	CSC436	CSC480	CSC495
Classes	11.75 (2.22)	8.57 (2.37)	no such diagram created by students	4.75 (3.50)
Attributes	17.50 (14.90)			6.00 (1.73)
Operations	28.00 (12)			9 (n/a)
Associations	11.28 (4.96)	8.29 (1.89)		2.50 (0.71)
Assoc. Classes	3 (n/a)			
Assoc. Names	8.80 (8.11)			
Multiplicities	17.00 (9.84)			
Aggregations & Compositions	5.00 (1.41)	3.67 (3.06)		2.00 (1.41)
Generalizations	2.50 (0.71)			
Errors	2 (n/a)			2.50 (0.71)

Table 3 shows the modeling elements and syntactic errors for **activity diagrams**. As outlined above, in CSC495, were excluded from course proceedings due to time constraints. Instead, students were encouraged to use sequence diagrams and focus on the interaction between components rather than control flow. In CSC380, however, with the exception of one group (who produced one activity diagram with six actions and control flows in four swimlanes), all groups attempted to use activity diagrams, the content and nature of model elements was largely akin to that of state machine diagrams. A possible explanation for this issue is that in a prerequisite course for CSC380, strong emphasis is placed on automata theory. Hence, the idea of statefulness in systems together with the largely overlapping notation in UML between activity- and state machine diagrams could have contributed to students not understanding the difference. In general, there appears to be a conceptual burden to understand the purpose of UML activity diagrams. In both CSC436 and CSC480, actions (which typically denote things the system *does* and not what it *has* or *is*) were frequently described using nouns (e.g., an action called "card reader," instead of "read card," avg. 3.50, std. dev. 0.707). It is also noteworthy that in CSC480, only a single activity diagram was created. Yet, like with class diagrams in CSC380, the activity diagram in CSC480 was rather monolithic and large, consisting of 16 actions and 26 control flows. Interestingly, language features like object flows, parameter pins, forks/joins, or interruptible regions, were omitted by all groups.

Table 3 Modeling Elements in Activity Diagrams

	CSC380	CSC436	CSC480	CSC495
Opaque Actions & Activities	6 (n/a)	8.00 (3.70)	16 (n/a)	diagram type excluded from course
Control Flows	6 (n/a)	10.28 (4.75)	26 (n/a)	
Start / End Nodes	2 (n/a)	1.00 (0.00)	0 (n/a)	
Decision & Merge Nodes		1 (n/a)	4 (n/a)	
Swimlanes	4 (n/a)			
Errors			2 (n/a)	

Table 4 shows the modeling elements and syntactic errors for **state machine diagrams**. Similar to activity diagrams, state diagrams were excluded from CSC495. Instead, students were encouraged to consider the interaction between components to describe the system behavior. Moreover, like class diagrams, students in CSC480 decided not to model state machine diagrams. By contrast, state machine diagrams were used rather excessively in both CSC380 and CSC436. In fact, many groups created precisely one diagram for other diagram types, hence satisfying the course requirement of "creating *at least* one diagram". Yet, all groups in CSC380 and CSC436 created several diagrams. The number of elements in state machine diagrams ranged between three states and five transitions in one group in CSC436 and 228 states with 233 transitions in one group in CSC380. This explains the high standard deviation in Table 4. It is furthermore to note that compared to other diagram types, there was a considerable number of mistakes. This was in part due to the aforementioned problems students had with activity diagrams. The majority of mistakes in both courses were either due to missing start/end nodes or due to erroneously used decision/merge nodes (which are syntactic features of activity diagrams only).

Table 4 Modeling Elements in State Machine Diagrams

	CSC380	CSC436	CSC480	CSC495
States	68.33 (74.35)	7.00 (2.58)	no such diagram created by students	diagram type excluded from course
Transitions	80.33 (73.17)	13.43 (10.11)		
Hierarchical Substates	6.50 (0.50)	2.33 (1.53)		
Concurrent Substates	3 (n/a)			
Entry/Exist Points	15.67 (12.27)	1.50 (1.17)		
Conditionals (Guards)	2.86 (2.85)			
Errors	13 (n/a)	7.50 (7.98)		

4.2 RQ2: Semantic Model Quality

As outlined in Section 3.3 and Fig. 1, after matching of modeling elements concluded, we

compared the quality of the models against the specific project milestones, modeling tasks, and project scope of each group. Our aim was to ascertain how the conceptual models and visual diagrams were able to express system properties and features. For each diagram type, five to eight criteria were adopted, based on the notational rules of the UML and check-rules of the Requirements Viewpoint ([6]). As stated in Section 3.3, for each criterion, values were assigned to models being evaluated which correspond to whether or not the model satisfied the criteria. For example, a criterion for state diagrams is "state transitions describe change of state", to which models could be given a score according to this measurement: "1 if 75% of state transitions describe change in state; -1 if 75% of state transitions do not describe change in state; 0.5 if 25% < of state transitions describe change in state < 75%; 0 otherwise". A full list of measurement criteria and their scoring, broken down by model type, can be found in the Appendix. In the following, we provide an overview of the perceived model quality for each course. Missing standard deviations indicate exact results, not averages.

In **CSC380**, sequence diagrams all scored similarly, as can be seen in Table 5, as the standard deviation for the quality score is relatively low. However, the relative quality of activity diagrams was relatively poor in that on average, the diagrams scored less than one-thirds of the possible score. Sequence and state machine diagrams were of a similar quality, albeit with lower variance in sequence diagrams. Activity diagrams had the poorest quality, albeit interestingly, this was the most used diagram type. Class diagrams scored the highest quality with comparable variance between them.

Table 5 Model Quality in CSC380

	Sequence	Class	Activity	State
# Groups (# Students)	8 (40)			
# Diagrams	21	7	40	8
Quality Score	62.50%	89.29%	39.43%	65.56%
Standard Deviation	7.07%	15.75%	26.81%	21.28%

In **CSC436**, variance between diagrams was quite high, ranging from ca. 32% to ca. 47%. Moreover, quality of activity diagrams was very low as indicated by the negative percentage (please recall the grading scheme in the interval [-1;1], negative results indicate that more than half the criteria were satisfied to less than 75%). This was mainly due to the aforementioned confusion between activity and state diagrams as outlined in Section 4.1. Average scores for all models within a team ranged from -0.5 to 3.75. Table 6 shows these results.

Table 6 Model Quality in CSC436

	Sequence	Class	Activity	State
# Groups (# Students)	7 (24)			
# Diagrams	1	6	7	7
Quality Score	40.00%	26.19%	-7.14%	89.29%
Standard Deviation		37.09%	31.67%	47.01%

In **CSC480**, as we have outlined in Section 4.1, no class and state machine diagram were created. Moreover, only one activity diagram was created, which satisfied only few criteria fully. On the other hand, the system described in the class project was a web-based system, which was described rather successfully using a series of nine sequence diagrams. Albeit quality was on average 58%, standard deviation in quality between diagrams was quite low, as can be seen in Table 7.

Table 7 Model Quality in CSC480

	Sequence	Class	Activity	State
# Groups (# Students)	1 (30)			
# Diagrams	9	0	1	0
Quality Score	58.00%		7.14%	
Standard Deviation	6.32%			

As mentioned above **CSC495**, only sequence- and class diagrams were created. Like in CSC480, the system to be developed was described using a series of sequence diagrams, mostly. The case example systems were mainly from the safety-critical embedded systems domain and hence were heavily dependent on their interaction with the operational context (i.e. external users and systems).

Table 8 Model Quality in CSC495

	Sequence	Class
# Groups (# Students)	4 (8)	
# Diagrams	52	5
Quality Score	47.76%	63.33%
Standard Deviation	34.11%	7.45%

Yet, many of the sequence diagrams contained but a single lifeline and (see Section 4.1) and many reflexive messages (i.e. from the object to *itself*), which indicates that students focused on the process- or state-oriented aspects of the system. For this, activity or state machine diagrams could have yielded higher quality and lower variance than the 47.76% (34.11% std. dev) shown in Table 8. By comparison, class diagrams depicted the entire system and were of relatively high quality (63%), with low variance.

4.3 RQ3: Pragmatic Quality

As outlined in Section 3.3, RQ3 is answered based on matching of the name identity of modeling elements to check for continuity between diagrams, i.e. whether there was correspondence of modeling elements between diagrams of the same and different types and between the conceptual model and the implementation. Modeling elements in each group's models were noted for how many different diagram types the modeling element appeared in. CSC380, CSC435, and CSC495 continuity was considered between diagrams and implementation. For CSC480, continuity was considered between sequence diagrams and implementation (see Section 4.2). In the following, we report on the qualitative pragmatic quality we found between diagrams.

In **CSC380**, results show that continuity between models was relatively high. Most modeling elements had corresponding modeling elements in other diagrams and could be found in the implementation. However, one group had no modeling elements appear in more than two model types. Seemingly, this group failed to understand that different diagrams pertain to the same conceptual model of the system and hence considered each individual diagram a chore to be completed. This is further evidenced by the fact that this group frequently questioned the use of conceptual models in class and produced only one diagram of each type, with relatively poor quality.

In **CSC436**, unlike CSC380, only two groups had at least twice the number of modeling elements in only one model type compared to the sum of modeling elements that were mentioned in two or more diagram types. These two groups had very similar projects, where the project of one group dependent on the results of the other group. Both groups hence worked together. Interestingly, albeit both groups had relatively poor continuity and correspondence between their own diagrams, a high level of correspondence to diagrams of the respective other group was achieved. Other groups in this course had a relatively high pragmatic quality, where the vast majority of modeling elements (e.g., interfaces, data types, or components) were found in several diagrams and the implementation.

In **CSC480**, a particular effort was made to by the project's requirements team to specifically demonstrate the interaction between humans and the system. This is part of the reason for the relatively high number of sequence diagrams. Moreover, correspondence between components (in the implementation) and lifelines (in sequence diagrams) as well as between sequence diagrams was exceptionally high. Yet, due to the fact that the activity diagram was created by a different group, there was little correspondence between the sequence diagrams and the activity diagram. In fact, one sequence diagram was taken as the template, after which the processes were modeled.

In **CSC495**, only one of the four student groups had at least twice the number of modeling elements in only one model type compared to the sum of modeling elements that were mentioned in two or more model types or the implementation. This indicates that some modeling elements were included in one diagram, however, never treated any further in corresponding diagrams. Similarly, these modeling elements were also absent from the implementation. These modeling elements could have been remnants from previous diagram iterations which in the later revisions, where simply forgotten to be removed or renamed. Other groups included at least one of same object in all model types and the implementation.

5. Conclusions

The conclusions that can be drawn from the data presented in Section 4 are limited due to the diversity in diagrams and courses. Nevertheless, we summarize some intriguing findings with regard to the research questions from Section 3.1 and share some additional subjective experiences.

RQ1: Syntactic Quality. Results show that student diagrams rarely make full use of the diagram type's feature set and are limited to relatively simple features. Moreover, class diagrams are most likely to be used successfully, yielding large (i.e., non-trivial) diagrams and activity diagrams are least likely to be used successfully. This may be due to the fact that activity diagrams overlap with state machine diagrams in terms of notation and with sequence diagram in terms of content. In general, we observe that students struggle with adopting the appropriate notational elements of the various diagram type. Albeit the diagrams submitted at the end of the courses were typically of high syntactic quality (i.e. only few syntactic errors remained), we observed that the modeling process itself was riddled with difficulty for most students. Detailed introductions and "cheat sheets" detailing the notation of diagram types were given in each course, yet students struggled with even simple concepts (e.g., sharp-edged corners for classes *only*; rounded-edge corners for activities and states *only*; state machine diagrams *must* contain at least one start node; reading directions on class associations are denoted with association label decorations; etc.). In early modeling phases, these were the most frequently noted syntactic errors.

RQ2: Semantic Quality. Recurring semantic errors include missing labels on associations, control/data flows, or transitions; ambiguity in labelling of classes, activities, states, or associations, confusion between states and events, confusion between states and activities; and general appropriateness of diagrams. In

this case, the term "appropriateness" is to be understood with regard to the system properties expressed in the diagram. For example, students often used decisions in activity diagrams to check, e.g., the value of a variable. Students would specify conditions that check if the value is above or below some threshold, but would forget to specify the *exact threshold* value. Such and similar specification gaps were quite common and is the main reason why the reported quality in Section 4.2 is between +30% and +50% for most diagrams.

RQ3: Pragmatic Quality. Despite semantic flaws in diagrams and syntactic struggle during the modeling process, pragmatic quality was surprisingly high in most groups. In part, conceptual models contain considerable syntactic and semantics flaws, which may impair interpretability by external stakeholders. Yet, the process of conceptual modeling was useful and highly effective for members of the same development team. In nearly all cases, the team members knew how to interpret the information contained in the conceptual models and how to move on with development, despite the flaws. We consider this one of our key findings.

However, students' motivation seems to be an important confounding factor. It appeared to us that while some students truly appreciate the systematic process of conceiving the system before implementing it, others despised it. For example, after work on some diagram was concluded, students shared the sentiment that they have a better understanding of how and what to do next. For a counter-example, several students, especially more technically inclined ones, often claimed that they failed to see the point in modeling and "would rather just code." In fact, a testimony given by one student was "I'm a code monkey, why should I care about pictures." Ironically, at one point during the semester, the same student spontaneously resorted to drawing a UML class diagram in pencil while clarifying an idea for the system behavior with a team mate.

6. Threats to Validity

Some threats to validity may have impaired our work. These and their mitigations are as follows.

Internal Validity. One threat that may impair the generalizability of the conclusions presented in Section 5 is related to sampling. We have taken diagrams from courses at our home institution, to which we had convenient access. Moreover, these courses had educational objectives that were not aligned with our study and therefore contain some differences, which limit comparability. Lastly, some courses were instructed by one of the authors such that researcher bias may be have been a factor. However, it is not our aim to present empirically rigorous results, but to contribute to the body of knowledge on how the quality of typical novice modelers' conceptual models may look like. We seek to foster discussion in the community and present results for comparison. For this reason, we have adopted a grounded-theory approach, based on the quality framework in [17] and make available our raw data in the Appendix, so the reader may compare our work to their own courses easily.

Construct Validity. Especially with regard to the semantic quality for RQ2, our mode of measurement may have impacted our results. Measuring the semantic quality of a diagram is inherently hard, as it largely depends on the specific system under development. For this purpose, we have not only resorted to UML's notational rules and the check rules for model-based specifications outlined in [6], we are also making available our scoring criteria in the Appendix to allow others to adopt and/or compare our results.

Conclusion Validity. As mentioned above, researcher bias may have been a factor in drawing our conclusions. Of course, as the instructor of the courses, the second author has inherent interest in their students' success. To remedy this issue, we provided quantifiable results to the farthest possible degree and supplemented conclusions with our in-class experiences, thereby clearly differentiating objective and subjective findings.

External Validity. A considerable issue with any study, especially regarding education, is the use of student participants. We concede that given a different choice of classes and students, the results outlined above may be different. Moreover, as mentioned above, comparability between courses is limited, due to the educational objectives. For this reason, we have provided a detailed description of similarities and differences between courses and provided sufficient details to foster comparability to other courses and other institutions. We have drawn conclusions that pertain to students' conceptual models at large, pointing out individual factoids and experiences, when appropriate. In fact, we are confident that the results and experiences reported herein are typical and similar for other samples and encourage other educators to share, compare, and discuss their results.

7. Summary & Outlook

In this paper, we have reported on the syntactic (RQ1), semantic (RQ2), and pragmatic (RQ3) quality of students' conceptual models in software development. We have reported quantifiable measurements on the usage and error frequency of language features in UML sequence, class, activity, and state machine diagrams in four software engineering courses at the baccalaureate level. Moreover, we have quantified the semantic quality based on criteria derived from UML's notational rules and the correspondence rules of the SPES Requirements Viewpoint [6]. Lastly, we have reported the pragmatic quality of the student

produced models, commenting on how well different diagrams fit together and lead towards implementation. Results show that albeit student modelers only use a small feature set of the UML language, and semantic errors are frequent, the benefit of conceptual models for the own development team is high.

Future work is concerned with ongoing investigation of students' model quality and shall lead towards improvement of our course curricula to maximize learning benefit, especially with regard to diagram semantics. In the interest of full disclosure, we make scoring scheme and raw data available (see Appendix). We seek collaboration in this regard.

References

[1] Alperowitz, L., J. Johanssen, D. Dzvonyar, B. Bruegge, "Modeling in Agile Project Courses", Proc. Educators' Symp at MoDELS, 2017.

[2] Boehm, B., "Software Engineering Economics," Prentice-Hall, 1981.

[3] Booch, G., J. Rumbaugh, I. Jacobson, "Unified Modeling Language," 2nd Ed. Addison-Wesley, 2005.

[4] Broy, M., W. Damm, S. Henkler, K. Pohl, A. Vogelsang, T. Weyer, "Introduction to the SPES Modeling Framework," In Model-Based Eng of Embedded Systems. Springer, 2012.

[5] Daun, M., A. Salmon, T. Weyer, K. Pohl, B. Tenbergen, "Project-Based Learning with Examples from Industry in University Courses," Proc. IEEE 29th Intl. Conf. SE, 2016.

[6] Daun, M., B. Tenbergen, T. Weyer, "Requirements Viewpoint," In Model-Based Eng of Embedded Systems. Springer, 2012.

[7] Engels, G., J. Hausmann, M. Lohmann, S. Sauer, "Teaching UML Is Teaching Software Engineering Is Teaching Abstraction," In Proc. MoDELS Sat. Events, 2005.

[8] Glaser, B., A. Strauss, "The Discovery of Grounded Theory," Transaction Publishers, 1967.

[9] Gorp, P., H. Schippers, S. Demeyer, "Students can get excited about Formal Methods," Proc. Educators' Symp at MoDELS, 2007.

[10] Karasneh, B., D. Stikkolorum, E. Larios, M. Chaudron, "Quality Assessment of UML Class Diagrams," Proc. Educators' Symp at MoDELS, 2015.

[11] Kayama, M., S. Ogata, D.K. Asano, M. Hashimoto, "Quantitative Conceptual Model Analysis for Evaluating Simple Class Diagrams made by Novices,"Joint Proc. EduSymp 2016 and OSS4MDE 2016, 2016.

[12] Kruchten, P., "The 4+1 View Model of architecture," IEEE Softw 12(6), 1995.

[13] Kuzniarz, L., M. Staron, "Best Practices for Teaching UML Based Software Development," Proc. EduSymp at MoDELS, 2005.

[14] Laplante, P., C. Neill, C. Jacobs, "Software requirements practices: some real data," Proc. 27th Annual NASA Goddard/IEEE Soft Eng WS, 2002.

[15] Lethbridge, T., R. Laganière, "Object-oriented Software Engineering: Practical Software Development Using UML and Java," McGraw-Hill, 2005.

[16] Lethbridge, T., G. Mussbacher, A. Forward, O. Badreddin, "Teaching UML using Umple: Applying model-oriented programming in the classroom," Proc. 24th IEEE Conf. Soft. Eng. Education & Training, 2011.

[17] Lindland, O., G. Sindre, A. Solvberg, "Understanding quality in conceptual modeling," IEEE Softw 11(2), 1994.

[18] Lubars, M., C. Potts, C. Richter, "A review of the state of the practice in requirements modeling," Proc. IEEE Intl. Symp. RE, 1993.

[19] Neill, C., P. Laplante, "Requirements engineering: the state of the practice," IEEE Softw 20(6), 2003.

[20] Schramm, J., S. Strickroth, N.-T. Le, N. Pinkwart, "Teaching UML Skills to Novice Programmers Using a Sample Solution Based Intelligent Tutoring System," Proc. Intl. Conf. Florida Art. Intelligence Research Society, 2012.

[21] Siau, K., P.-P. Loo, "Identifying Difficulties in Learning UML," Info Sys Mgmt 23(3), 2006.

[22] Sikora, E., B. Tenbergen, K. Pohl, "Industry needs and research directions in requirements engineering for embedded systems", Requirements Engineering 17(1), 2012.

[23] Stikkolorum, D., T. Ho-Quang, B. Karashneh, M. Chaudron, "Uncovering Students' Common Difficulties and Strategies During a Class Diagram Design Process: An Online Experiment," Proc. EduSymp at MoDELS, 2015.

[24] Stol, K., P. Ralph, B. Fitzgerald, "Grounded Theory in Software Engineering Research: A Critical Review and Guidelines", Proc. 38th IEEE/ACM Intl Conf. SE, 2016.

Appendix

The scoring schema for RQ2 can be found at https://bit.ly/2Joumvz . The raw data can be found at https://bit.ly/2Nfr4vX .