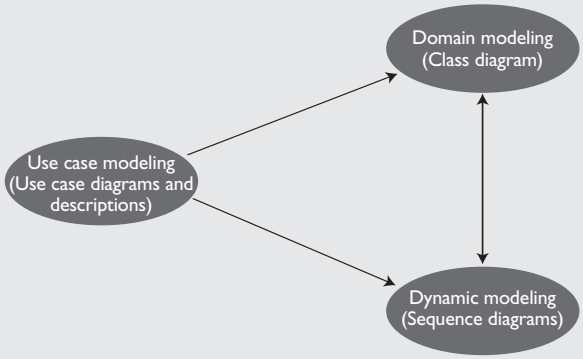
Assisting Novice Analysts in DEVELOPING QUALITY CONC

Knowing the kinds of modeling errors they are most likely to produce helps prepare novice analysts for developing quality conceptual models.

By Narasimha Bolloju and Felix S.K. Leung

During the analysis phase of information systems development, systems analysts capture and represent systems requirements using conceptual models (such as entity-relationship diagrams, class diagrams, and use case diagrams). Considering the fact that the reported failures of a significant percentage of developed systems are linked to faulty requirements, it is extremely important for these analysts and critical to the system's ultimate success to ensure the quality of the conceptual models they develop in the early phases of systems development.



Modeling activities and artifacts.

However, developing good-quality conceptual models is a challenge for many analysts. The models must support communications among end users and developers in defining and documenting systems requirements as faithfully as possible. The models' effectiveness is influenced by the complex interactions among modeling constructs, task requirements, analysts' own modeling experience and cognitive abilities, and interpreters' experience with conceptual models [11]. Novice analysts developing conceptual models have more difficulty compared to experienced analysts in terms of domain-specific knowledge, problem structuring, and cognitive processes [9]. In addition, for novice analysts a lack of established validation procedures [10] makes conceptual modeling that much more difficult.

Many systems analysts develop conceptual models by following the object-oriented approach in the modeling techniques of the Unified Modeling Language (UML) [6]. For example, UML provides 12 different types of diagrams for documenting a system from a variety of perspectives, and a typical systems analyst is expected to be familiar with many of them. Though UML is widely used, the UML diagrams are not highly rated by analysts in terms of usability [2]. Several practitioners offer recommendations and guidelines (such as [3, 4]), suggesting analysts employ commonly used patterns (such as [1, 5]) when using UML modeling techniques. However, typical novice analysts fail to derive maximum benefit from such assistance due to the cognitive overload involved in the recommendations and guidelines.

Here, we present the results of an empirical study we conducted aimed at identifying the most typical set of errors frequently committed by novice systems analysts in four commonly used UML artifacts—use

EPTUAL MODELS WITH UML

case diagrams, use case descriptions, class diagrams, and sequence diagrams—and discuss how they affect the quality of artifacts developed. Ensuring that artifacts are free of such errors helps novice analysts develop better-quality UML artifacts. Our findings are relevant to instructors of systems analysis courses, software quality-assurance teams, CASE tool developers, and researchers in the field of conceptual modeling, as well as to the analysts themselves.

Use case-driven modeling is a popular approach employed in systems development using the object-oriented method. First to be developed are use case models comprising use case diagrams and use case descriptions; the models then guide subsequent modeling activities. The figure here outlines major activities and artifacts (in parentheses) developed through these activities. Use case models are used in the analysis phase to capture and represent high-level system requirements. These models include two types of components:

Diagrams. To depict use cases corresponding to elementary business processes, associations among actors and use cases, and relationships (such as includes, extends, and generalization) among use cases; and

Descriptions. To provide requirements described in terms of main scenarios with a sequence of steps and alternate scenarios described as extensions to steps in the main scenarios.

Domain models, represented by analysts through class diagrams, include classes from the problem domain and a variety of relationships (such as generalization hierarchies, associations, and aggregations) among classes. Each class is described by the analyst through a set of attributes and a set of operations. Although classes, attributes of classes, and relationships among classes are identified mostly through descriptions in use case models, most operations of classes are derived from interaction diagrams (such as sequence and collaboration).

Dynamic models (such as sequence diagrams and collaboration diagrams) are used by analysts to capture system behavior through a sequence of message flows among classes and objects. They help identify and depict responsibilities (expressed as operations) of various classes and objects in fulfilling the systems requirements previously identified in use case descriptions.

The conceptual model quality framework discussed in [7] provides a systematic way to analyze the quality of UML artifacts from syntactic, semantic, and pragmatic quality perspectives (see the sidebar "Quality Categories for Conceptual Models"). Different types of errors in artifacts help produce different types of quality. For example, semantic errors (such as wrong cardinality specification and missing attributes in domain models) affect the validity and completeness aspects of semantic quality, respectively. Table 1 lists examples of errors belonging to these three quality categories. Here, we consider that the relationship between the overall quality of artifacts and the numbers of errors that can be identified is negatively correlated. That is, fewer errors indicate better quality.

UML Artifacts Analyzed

Using the framework in [7], we analyzed the quality of the UML artifacts in 15 team-project reports submitted by final-year full-time undergraduate students taking a course in object-oriented analysis and design in the Department of Information Systems at the City University of Hong Kong. All had previously taken a structured systems analysis and design course in their second year of the program. They worked in teams of three or four students each on semester-long team projects. Each project required a final submission consisting of four parts: a use case diagram; a set of use case descriptions; a class diagram; and a set of sequence diagrams corresponding to the use case descriptions. All teams used Microsoft Visio, a diagramming program with rudimentary CASE support, for drawing UML diagrams and Microsoft Word for writing use case descriptions with a provided document template.

The teams worked on projects involving a variety of business applications, including banking, hotel reservations, movie ticketing, and airline reservations. These projects involved comparable complexity in terms of the modeling skills and effort that would be required of the typical novice analyst. On average, the use case diagrams included six actors and 16 use cases with three or four important use cases described in detail. The class diagrams included an average of 14 classes and up to 50 attributes and 23 operations across all classes. Each sequence diagram, corresponding to a use case description, included an average of six objects and 14 messages.

	Syntactic Quality	Semantic Quality	Pragmatic Quality
Use Case Models	Inappropriate use case names (such as not beginning with an action verb) Invalid notation in use case diagram	Invalid relationship (include, extend, or generalization) between use cases Incomplete scenario description	Poor layout of use case diagram Presence of implementation details (such as user interface) in use case description
Domain Models	Missing cardinality details for associations Inappropriate naming of classes and associations	Incorrect cardinality specification Used aggregation in place of association	Redundant attributes and associations Specialization with little distinction among subclasses
Dynamic Models	Improper positioning of classes and/or objects along the timeline in sequence diagram	Incomplete specification of message parameters	Inappropriate delegation of responsibilities

Table 1. Example errors in various quality categories.

To prepare a coding scheme, we identified and compiled a list of errors from each artifact of each project included in the study, then separated the errors into the three categories of quality—syntactic, semantic, pragmatic—according to the framework in [7]. The final coding scheme included 13, 14, 35, and 23 errors for use case diagrams, use case descriptions, class diagrams, and sequence diagrams, respectively.

As part of the study, we tested the coding scheme with one of the project reports that had been excluded from the rest of the study, then separately examined each artifact of 14 remaining projects (on separate

Quality Categories for Conceptual Models

Many researchers today are trying to describe and define the various aspects of the quality of conceptual models in UML. For example, [7] focused on the need for a framework addressing both process and product in quality, proposing a framework that borrows three linguistic concepts—syntax, semantics, and pragmatics—as suitable categories for defining the quality of conceptual models:

Syntactic quality. The syntactic correctness of a model implies that all statements in it depend on the syntax of the language, capturing how a given model adheres to the language rules or to the syntax. Therefore, fewer errors and deviations from the rules indicate better syntactic quality.

Semantic quality. This category captures the quality of a model in terms of what the model lacks that is present in the domain, as well as what the model includes that is not present in the domain. Semantic quality is described in terms of validity and completeness goals. The validity goal specifies that all statements in the model are correct and relevant to the problem domain. The completeness goal specifies that the model contain all statements about the problem domain that are correct and relevant. However, it may be that these two goals cannot be achieved, unlike syntactic correctness, which can be achieved.

Pragmatic quality. This category addresses the comprehension aspect of the model from the stakeholders' perspective. Pragmatic quality captures how the model has selected "from among the many ways to express a single meaning" and essentially deals with making the model easy to understand. The comprehension goal specifies that all audience members (or interpreters) completely understand the statements in the model that are relevant to them.

These categories address various aspects of quality that require more and more analyst effort and expertise to achieve. As an overall framework, they can be applied to graphic and text-oriented modeling artifacts, including entity relationship diagrams, data flow diagrams, object models, and use case descriptions.

copies of the reports) for the errors listed in the coding scheme. We noted only the first occurrence of each error in a project, and we ignored any multiple occurrences belonging to the same project. We then exchanged with each other the list of errors we had identified, and independently verified the presence of the errors using our copies of artifacts. We identified a total of 380 errors in the 14 projects, with an overall inter-rater agreement of 75% after the verification. This level of agreement is acceptable, considering the large number of possible error codes (85) in the coding scheme, the complexity of the highly subjective process of finding the errors in artifacts, and the exclusion of errors not identified from the calculation of inter-rater agreement. However, to reach a complete inter-rater agreement we had to discuss and resolve the remaining differences in error occurrences in each project in the study.

	Syntactic	Semantic	Pragmatic
Use Case Diagrams and Descriptions	29	62	56
Class Diagrams	26	32	33
Sequence Diagrams	38	41	33

Table 2. Distribution of various types of errors.

Quality of UML Artifacts

Table 2 outlines the distribution of errors we identified in the study in various quality categories for the four types of artifacts. It indicates the relative difficulty of developing high-quality artifacts. Among the artifacts we considered, we found that developing quality use case diagrams and descriptions was difficult. Fewer errors in class diagrams might be attributed to analysts' prior experience with the entity-relationship modeling technique. Meanwhile, a good number of errors in semantic and pragmatic categories of sequence diagrams might have been preempted due to syntactic errors.

To identify the set of frequently committed errors, we considered only those errors we identified across five or more projects in different categories of quality (see Table 3). Since many of the syntactic errors are easily prevented through CASE tools (such as Rational Rose and Visual Paradigm for UML), we focus here on the errors that affect semantic and pragmatic quality.

	Syntactic	Semantic	Pragmatic
Use Case Diagrams	Improper notation (57%)	Invalid relationship between use cases (64%)	Manual operations listed as use cases (57%)
Use Case Descriptions	Use case name mismatch between diagram and description (64%)	Missing a critical step (64%) Incomplete or ambiguous step description (64%) Invalid extension to a step (50%)	Excessive use of manual steps (71%) Use of implementation details (such as user interface) database (64%) Excessive splitting of steps (57%)
Class Diagrams	Nonimplicit operations from sequence diagram not shown (64%) Implicit operations listed (50%)	Wrong association cardinality range or multiplicity (50%) Wrong location of attributes (36%) Wrong location of operations (36%) Operation cannot be realized using existing attributes and relationships (36%)	Insufficient distinction among subclasses (43%) Presence of derived or redundant attribute(s) (36%)
Sequence Diagrams	Missing initial trigger (71%) Return to an object different from caller (64%) Class/object does not belong to class diagram (50%)	Critical message parameters missing (71%) Message parameters used before their values are available (64%) One or more essential classes/objects left out the sequence diagram (50%)	Responsibility delegated to a wrong object (57%)

Table 3. Frequently observed errors in various quality categories.

Use case diagrams and descriptions. Larger numbers of semantic and pragmatic errors in these artifacts compared to syntactic errors might have been the result of the simple syntax of use case diagrams and the document template provided for use case descriptions. This difference in numbers of errors also highlighted the difficulties in developing good quality use case models, especially in relating use cases in diagrams and in writing steps in use case descriptions. In conducting the study we observed that most of the use case relationship errors were in use cases involving the "extends" type of relationship. Some practitioners even recommend against using this type of relationship between use cases due to its limited utility. Many of the pragmatic errors in use case descriptions (such as used implementation details in step description and manual operations) may be attributed to novice analysts' inability to separate logical and physical specifications and identify the functionality to be provided by the system.

Class diagrams. Participating team members' prior experience with entity-relationship modeling appears to have contributed to overall quality both positively and negatively. We frequently observed errors related to association specification, especially the cardinality details—either wrong range of values or reversed values. Most of the errors we observed in the pragmatic quality category (such as derived or redundant attributes and the use of keys) can be attributed to the ana-

lysts' prior experience with database design and implementation. We also noticed instances where the subclasses in class hierarchies with insufficient distinction among subclasses could have been due to either the urge to use this feature or the lack of depth in requirements specified in use cases.

Sequence diagrams. Most of the errors we saw can be attributed to novice analysts' inexperience in problem-solving skills (such as decomposition) and to their difficulty understanding object-orientation. Many syntactic errors are related to message flow (such as missing initial trigger messages and returning control to objects other than the calling object). Pragmatic errors included improper delegation of responsibility—often to a wrong object/class—and/or making a class/object perform computations that can be delegated to other objects.

One limitation of the study was that we used artifacts from the project work of undergraduate students and might have identified a greater number of errors than if we had looked at only the work of experienced analysts. The students attending the object-oriented analysis and design course had already completed a course on structured systems analysis and design. Their project work in the object-oriented analysis and design course required considerable effort by teams of three or four students over a 13-week semester. As a result, the quality of the artifacts they developed may be considered comparable to the quality of artifacts developed by typical novice systems analysts.

We addressed the pragmatic quality in the study from the perspective of only one type of stakeholder—the instructor or tutor in the role of experienced analyst. This approach can be expected to minimize certain problems associated with employing inexperienced students (such as [8]) in evaluating the quality of the artifacts produced by other students. Although this approach ensured that we identified as many errors as possible from this perspective, it is important to consider other types of stakeholders or interpreters of conceptual models (such as systems designers, programmers, and end users) for identifying quality problems from other perspectives.

Implications

The framework in [7] enabled us to identify a small set of errors typically committed by novice systems analysts. By ensuring that the artifacts being created are free from such errors, novice systems analysts will be able to develop higher-quality conceptual models. Knowing these errors should help practitioners design error-prevention and error-removal mechanisms to enhance the quality of artifacts developed by novice analysts. Since many of the errors pertaining to the syntactic category can be eliminated through CASE tools, we limit our recommendations to the semantic and pragmatic categories. Training programs for novice analysts, based on these errors, aimed at imparting skills and techniques (such as writing proper step descriptions in use cases, defining useful generalization-specialization hierarchies, and delegating responsibilities to objects) would be effective in preventing many types of semantic and pragmatic errors. The errors we identified are also useful for developing checklists and guidelines for quality-assurance teams. Moreover, instructors teaching systems analysis can focus on imparting modeling skills that account for these errors.

Our study also suggests several interesting directions for research on conceptual modeling (such as investigating relationships among different types of quality among artifacts, developing validation procedures, and developing instruments for measuring quality from a variety of perspectives). Developers of CASE tools can incorporate facilities to provide guidance to novice analysts in preventing typical novice errors during the modeling process. C

References

1. Adolph, S. and Bramble, P. *Patterns for Effective Use Cases.* Addison Wesley, Boston, 2003.
2. Agarwal, R. and Sinha, A. Object-oriented modeling with UML: A study of developers' perceptions. *Commun. ACM 46,* 9 (Sept. 2003), 248–256.
3. Ambler, S. *The Elements of UML Style.* Cambridge University Press, New York, 2003.
4. Cockburn, A. *Writing Effective Use Cases.* Addison-Wesley, Boston, 2001.
5. Fowler, M. *Analysis Patterns: Reusable Object Models.* Addison Wesley, Menlo Park, CA, 1997.
6. *Introduction to OMG's Unified Modeling Language (UML)* (Mar. 27, 2004); www.omg.org/gettingstarted/what_is_uml.htm.
7. Lindland, O., Sindre, G., and Sølvberg, A. Understanding quality in conceptual modeling. *IEEE Software 11,* 2 (Mar. 1994), 42–49.
8. Moody, D., Sindre, G., Brasethvik, T., and Sølvberg, A. Evaluating the quality of information models: Empirical testing of a conceptual model quality framework. In *Proceedings of the 25th International Conference on Software Engineering* (Portland, OR, May 3–10). IEEE Computer Society, Washington, D.C., 2003, 295–305.
9. Schenk, K., Vitalari, N., and Shannon Davis, K. Differences between novice and expert systems analysts: What do we know and what do we do? *Journal of Management Information Systems 15,* 1 (Summer 1998), 9–50.
10. Shanks, G., Tansley, E., and Weber, R. Using ontology to validate conceptual models. *Commun. ACM 46,* 10 (Oct. 2003), 85–89.
11. Wand, Y. and Weber, R. Research commentary: Information systems and conceptual modeling: A research agenda. *Information Systems Research 13,* 4 (Dec. 2002), 363–376.

Narasimha Bolloju (narsi.bolloju@cityu.edu.hk) is an associate professor in the Department of Information Systems at the City University of Hong Kong.

Felix S.K. Leung (isfelix@cityu.edu.hk) is a Ph.D. candidate in the Department of Information Systems at the City University of Hong Kong.