



Understanding Quality in Conceptual Modeling

ODD IVAR LINDLAND, GUTTORM SINDRE, and
ARNE SØLVBERG, University of Trondheim, NTH

◆ *With the increasing focus on early development as a major factor in determining overall quality, many researchers are trying to define what makes a good conceptual model. However, existing frameworks often do little more than list desirable properties. The authors examine attempts to define quality as it relates to conceptual models and propose their own framework, which includes a systematic approach to identifying quality-improvement goals and the means to achieve them.*

Most developers are recognizing that the quality of the end product depends greatly on the accuracy of the requirements specification and are concentrating more on improving the early stages of development.

A conceptual model — any collection of specification statements relevant to some problem — is an important part of early development. Within the scope of conceptual modeling is the requirements specification, as well as any additional knowledge not in the form of requirements.

Much has been written about the quality of conceptual models and how to achieve it through the use of frameworks that classify quality goals. Unfortunately, these frameworks for the most part merely list properties, without giving a systematic structure for evaluating them. We

propose a framework that not only identifies major quality goals, but gives the means for achieving them.

Our framework has two unique features:

- ◆ It distinguishes between goals and means by separating *what* you are trying to achieve in conceptual modeling from *how* to achieve it. We have made the goals more realistic by introducing the notion of feasibility.

- ◆ It is closely linked to linguistic concepts because we recognize that modeling is essentially making statements in some language.

EXISTING FRAMEWORKS

The box at right gives some useful model properties we compiled from frameworks by David Kung,¹ Gruia-Catalin Roman,² Raymond Yeh,³ and Alan

Davis,⁴ as well as from a glossary by Richard and Mildred Thayer.⁵ We chose these sources because we believe they represent mainstream requirements-engineering terminology.

In compiling this list of properties, we observed several significant trends:

- ♦ *Many definitions are vague, complicated, or in some cases, even lacking.* Conceptually clean, for example, covers simplicity, clarity, and ease of understanding, but these subproperties are not explained. The definition of complete may be practical but it also covers four aspects of the specification: functionality, input data, documentation, and content. It would have been simpler had there been a definition for each aspect. For example, a functionally complete specification is one in which all functional requirements have been included.

Sometimes terms are not defined at all, such as incompleteness tolerance. The term's intended meaning—prob-

ably that the specification can have a low quality temporarily (can contain syntax errors, be inconsistent, and so on) so as not to compromise the inherent creativity of specification building—is left to the framework user to interpret.

- ♦ *The list is unstructured and the properties are partly overlapping.* Some features, such as consistency and unambiguity have a well-defined meaning and address the degree to which the specification gives a correct picture of the customer's needs. Other features, such as verifiability and modifiability are on a different level—relating to the work to be done with the specification itself, rather than the relationship between the specification and the customer's needs. Furthermore, features like precision, formality, and testability are highly interdependent; whereas, appropriate, correct, and minimal in fact almost have the same meaning. They all require that the specification not address design and

implementation issues.

- ♦ *Specification properties are mixed with language and method properties.* Expressiveness is something you expect from the language used to build the specification, not from the specification itself. A constructable specification depends on a method for formulating the requirements; you can't derive such a method from the specification itself.

- ♦ *Some properties presuppose the existence of a design and even an implementation.* For example, to check if a specification is correct, you must have a design. To check if a specification is precise and testable, you must have an executable implementation.

- ♦ *Some goals are unrealistic, even impossible to reach.* According to the first definition of complete, for example, a specification should include everything that the software is supposed to do. Because customers generally do not know exactly what they need, you cannot

HOW OTHERS VIEW QUALITY PROPERTIES

Annotated. Prioritize requirements, for example, relative necessity and relative stability.

Appropriate. Specification's ability to capture, in a straightforward manner that is free of implementation considerations, the concepts germane to the system's role in the environment for which it is intended (business data processing, process control, communication hardware, and so on).

Complete. Holds true if

- ♦ Everything that the software is supposed to do is included in the specification.

- ♦ The system responds to all classes of input data.

- ♦ All pages are numbered; all figures and tables are numbered, named, and referenced; all terms and units of measure are provided; and all reference mat-

erials are present.

- ♦ No sections are marked "to be determined."

Conceptually clean. Covers notions such as simplicity, clarity, and ease of understanding.

Consistent. No set of requirements is in conflict with any other set.

Constructable. Existence of a systematic approach to formulating requirements.

Correct. Every requirement represents something required of the system to be built and no error will affect design.

Executable. The extent to which you can construct functional simulations of the component from the specification before starting design or implementation.

Expressive. Able to express everything that needs to be mod-

eled without much effort from the modeler.

Expressive economy. Not defined. We assume it is the fewest number of statements with which the specification can be built without losing semantic content.

Formal. A specification that is built in a formal language, which permits a mechanic analysis of the specification.

Incompleteness tolerant. Not defined.

Minimal. The specification should not overconstrain system design.

Modifiable. The structure and style are such that you can make any necessary changes to the specification easily, completely, and consistently.

Precise. You can develop a procedure to determine if some

realization does or does not meet some particular requirement.

Testable. There are cost-effective procedures that let you verify if the design and/or realization of some component satisfies its functional and nonfunctional requirements.

Traceable. The origin of each requirement is clear and it is easy to reference each statement in future documentation.

Unambiguous. Every requirement has only one interpretation.

Understandable. The specification is understood by non-computer specialists without too much effort.

Verifiable. There is a cost-effective process that lets a person or machine check that the actual as-built software meets the requirements.

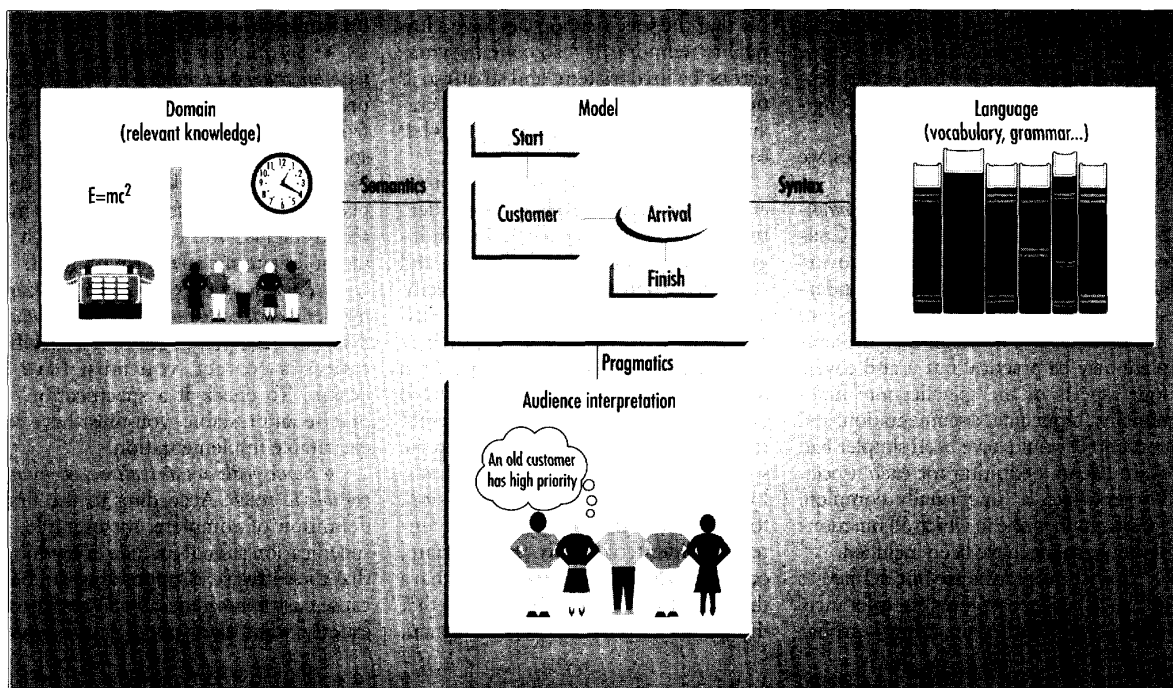
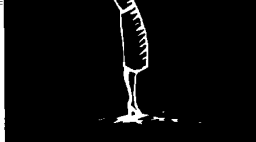


Figure 1. Four cornerstones of the framework (boxes) and the three connecting linguistic aspects (edges).

hope to build a complete specification.

Thus, while existing frameworks have made a first step toward defining the desired properties of a requirements specification, they fall short of a deep analysis. A framework that offers a more in-depth treatment of quality should

- ◆ Separate direct specification properties from language and method properties.

- ◆ Separate quality goals and the means to reach them. The goals should also be realistic. Total completeness, for example, is *not* a realistic goal. We have made goals more realistic by introducing the notion of feasibility.

- ◆ Have a mathematical foundation so that you can precisely interpret the properties.

- ◆ Have properties that relate directly to specification building and not depend on later project deliverables.

FRAMEWORK FOUNDATIONS

Our framework addresses quality in

the conceptual modeling *process* as well as in the model itself. In our opinion, the terms “specification” and “requirement” are too limiting to describe this process, even though mainstream terminology embraces them. Requirements imply something prescriptive. In early modeling, most of what is done is to analyze the problem area, so many pieces of information are more statements than requirements.

For these reasons, we believe “model” and “model statement” are more accurate and our framework is based on these interpretations and terms (which we use for the rest of the article).

Figure 1 shows the main points of our framework, which borrows from three important linguistic concepts and applies them to four aspects of modeling: Language, domain, model, and audience participation.

The linguistic concepts we borrowed are

- ◆ *Syntax*. Relates the model to the

modeling language by describing relations among language constructs without considering their meaning.

- ◆ *Semantics*. Relates the model to the domain by considering not only syntax, but also relations among statements and their meaning.

- ◆ *Pragmatics*. Relates the model to audience participation by considering not only syntax and semantics, but also how the audience (anyone involved in modeling) will interpret them.

Language. The modeling language, *L*, consists of all the statements that can be made according to the syntax, which for most languages is an infinite number. *L*’s syntax is formed by an alphabet and a grammar. The alphabet contains a set of modeling constructs, each of which has a unique notation, which are the building blocks for the conceptual model. The grammar contains rules that define how to legally combine modeling constructs. *L* also has semantics, which defines the

meaning of L 's constructs and thus the statements' implications. Semantics may contain deduction rules explaining how new statements can be derived from statements already made.

Domain. The domain, D , consists of all possible statements that would be correct and relevant for solving the problem. Hence, every unique problem has a unique, usually evolving, domain.

This use of domain differs from that widely found in literature on information-systems engineering, in which domain is rather loosely equated with problem area. Domain analysis has emerged as the discipline of analyzing the functional requirements of a problem area like banking or insurance, rather than requirements for a specific customer. But here, domain is synonymous with the ideal knowledge about a particular problem. It is similar to the mathematical domain, which is the set of values allowed as arguments to a function.

Model. The model, M , is the set of statements actually made. Its components are the explicit model, M_e , which is the set of statements explicitly made, and the implicit model, M_i , which is the set of statements that can be derived from M_e according to L 's deduction rules.

For example, assume that L is propositional logic and that M_e contains the statement $A \wedge B$. M_i will contain the derived statements A and B , as well as an infinite number of other variations that can be derived from the explicit statement $A \wedge A$, $A \wedge A \wedge A$, or $A \wedge B \wedge A$. Hence, although M_e must contain a finite number of statements, M_i (and thus M itself) usually contains an infinite number. This infinite set will generally be highly redundant.

Audience interpretation. Audience interpretation, A , is the set of statements that the audience thinks the model contains. By audience, we mean all who need to understand the model, including all stakeholders in the development process. Potential users fall in this category as well as customers, domain experts, analysts, and designers. An audience might even be a computer, which must understand the

model to automatically manipulate it.

We prefer audience to user because the latter term might be mistaken for users of the system being built rather than users of the model.

Quality concerns. The focus of our article is the quality of a model in and of itself. However, many factors affect a model's quality. Three are worth mentioning:

- ♦ *Language-domain appropriateness.* This is the measure of how the language fits the domain, the extent to which the language makes the kind of statements needed in the domain.

- ♦ *Language-audience appropriateness.* This is the extent to which the audience is able to learn, understand, and use the language.

- ♦ *Audience-domain appropriateness.* This is the extent to which the audience is already familiar with or is able to get familiar with the domain.

All three concerns might also work in reverse. In other words, if you know that you have to address a certain domain, you will try to choose a language that fits. If you know that your analysts are fluent in one particular modeling language, you will seek assignments in domains where this language is appropriate.

For the rest of the article, we focus on

issues that directly involve the model, assuming for the most part that an acceptable level of appropriateness has been achieved.

Figure 2 is an overview of our framework. In it, we assume there are three types of model quality. *Syntactic* quality is how well the model corresponds to the language. *Semantic* quality is how well the model corresponds to the domain. *Pragmatic* quality is how well the model corresponds to its audience interpretation.

The means section of the figure illustrates how our framework distinguishes activities to improve quality (modeling activities) from the properties of the model that enable these activities (model properties). The figure also shows means grouped according to the type of quality they address *directly*. For example, means whose primary effect is to increase model comprehension are classified as pragmatic.

This does not mean that a certain property or activity contributes exclusively to that type. Improvements in pragmatic quality can improve semantic quality by making it easier to detect invalidity and incompleteness, for example. Likewise, improvements in syntactic quality can improve pragmatic quality because the model is easier to understand.

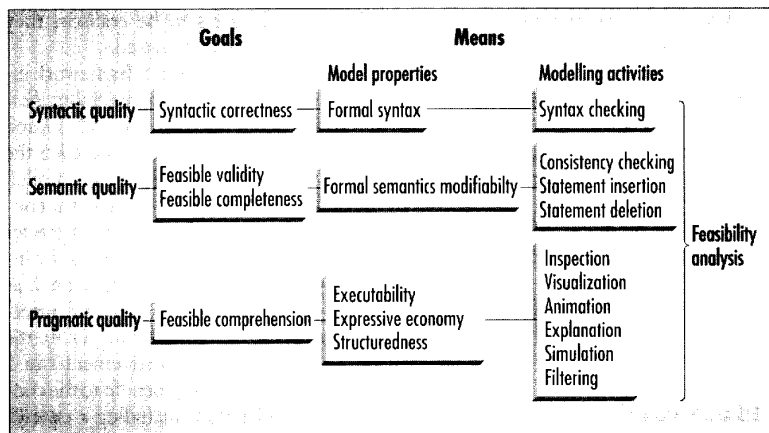
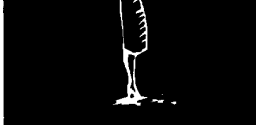


Figure 2. Proposed framework for distinguishing quality-improvement goals and the means to achieve them.



SYNTACTIC QUALITY

The more closely the model adheres to the language rules, the higher the syntactic quality. It follows then that the more errors and deviations from the rules, the lower the syntactic quality.

Goals. There is only one syntactic goal, syntactic correctness, which means that all statements in the model are according to the syntax. That is, $MV = \emptyset$.

The corresponding error, syntax error, occurs when this equation does not hold. Syntax errors are caused by one of two errors: Morphological errors, in which the model expresses statements with symbols that are not defined in the language's alphabet, and syntactic incompleteness, in which the model lacks constructs or information to obey the language's grammar.

Errors, of course, must be made in the explicit model, M_E . Because you derive M_I using the language's derivation rules, it cannot contain syntactically incorrect statements, as long as the assumption is that nothing can be derived from a syntactically incorrect statement.

Means. There are three basic syntactic means: error prevention, error detection, and error correction. Prevention and detection are easier with a formal syntax like Backus-Naur form. Prevention is rejecting the insertion of erroneous statements into the model. Detection is finding errors after statements have been inserted. Prevention is most useful for morphological errors; detection, for syntactical completeness. You can easily automate both these activities. Error correction — to replace a detected error with a correct statement — is more difficult to automate.

SEMANTIC QUALITY

The model can lack something that

the domain contains, or it can include something the domain doesn't have. The more similar the model and domain, the better the semantic quality; the more different, the worse the quality.

Goals. There are two semantic goals: validity and completeness. Validity means that all statements made by the model are

correct and relevant to the problem. That is, $MVD = \emptyset$. The corresponding error class is invalidity, meaning that the equation does not hold.

Completeness means that the model contains all the statements about the domain that are correct and relevant. That is, $D \setminus M = \emptyset$. The corresponding error class is incompleteness, meaning that the equation does not hold.

For anything but extremely simple problems, you cannot achieve total validity and completeness. Attempts to do so would require spending unlimited amounts of time and money, which is unacceptable. The time to terminate a modeling activity is thus not when the model is perfect (which will never happen) but when it has reached a state where further modeling is less beneficial than applying the model in its current state. Hence, for the semantic goals to be realistic, they have to be somewhat relaxed.

To accomplish this, our framework incorporates the concept of feasibility. *Feasible validity* is defined as $MVD = R \neq \emptyset$, and there is no statement r in R (where R is the set of invalid statements in the model) such that the additional benefit to the conceptual model from removing r is greater than the drawbacks of eliminating the invalidity. *Feasible completeness* is defined as $D \setminus M = S \neq \emptyset$, where S is the set of correct and relevant statements not yet in the model, and there is no statement s in S such that the additional benefit to the conceptual model from including s exceeds the drawbacks of including it.

Feasibility thus introduces a trade-off

between the benefits and drawbacks for achieving a given model quality, both of which are part of the domain because they are an integral part of the problem to be solved. We use the term drawbacks instead of the more familiar term cost in an effort to distinguish between purely economic issues and economic issues plus factors like the user preferences and ethics.

Relaxing validity and completeness is in line with the observation that there is no one right solution to a wicked problem.⁶ The choice of solution should depend on who is trying to solve the problem. The more competent the model builders (analysts, end-users, domain experts) the more feasible it will be to achieve a high level of validity and completeness because they will work smarter (spend fewer resources) to achieve that level.

Compared with the number of semantic properties listed in the box on p. 43, having only feasible validity and completeness as our semantic goals seems an oversimplification. However, a closer look at several properties reveals that it is not. (In the following comparison, we drop the consideration of feasibility because none of the definitions addresses it.)

♦ *Correct.* This corresponds to valid. The definition of correct in the box — "Every requirement represents something required of the system to be built and no error will affect design." — is the same as saying all statements must be relevant and correct. This is the same definition we use for valid. Thus, if some statement is not relevant or leads to a design error, it is invalid.

♦ *Minimal.* This is subsumed by valid because if M contains a model statement that overconstrains system design, the statement is simply invalid.

♦ *Annotated and traceable.* Both of these are subsumed by completeness because of how we define domain. We assume that to annotate something means to associate priorities to requirements. Such priorities are themselves an important part of the relevant domain knowledge. Hence a model that lacks important statements about priorities is certainly incomplete. Traceabil-

ADDING THE NOTION OF FEASIBILITY TO GOALS LETS YOU TRADE OFF BETWEEN BENEFITS AND DRAWBACKS.

ity means that the origin of model statements should be clear. Because the model itself becomes a part of the domain as soon as modeling starts, the lack of important knowledge about the model is a case of incompleteness.

♦ **Consistency.** This is one of the most common semantic goals, yet it too is subsumed by validity and completeness. The reason is that any case of inconsistency must be caused by at least one invalid statement or the lack of a statement (incompleteness). The definition of inconsistency is that M contains contradictory statements.

To illustrate, consider the case in which you must model an organization's business rules, many of which can be contradictory.

Suppose the model must account for the rule "If a company has been our customer for more than 10 years, the customer status should be high priority." and the rule "If a customer has been late with payments for more than three times, the customer status should be low priority." (To detect inconsistencies, such rules should be in a formal language, but we present them in English for the sake of readability.)

But what if a company who has been the customer for 12 years is late with payments more than three times? You could decide to change the first rule to rate the customer as "medium priority" or the second to "more than four times." Either of those two actions would mean that the original rules were invalid.

On the other hand, you might decide that both rules are valid, and add another rule, "If there are contradictory rules about customer status, the sales manager should resolve the issue," which requires the system simply to notify someone. Adding a rule to resolve a contradiction would mean that the original model was incomplete because it had no such rule.

♦ **Unambiguity.** Like consistency, this is subsumed by validity and completeness. In our framework, M is the union of interpretations that are either contradictory (which leads to the same

argument given for inconsistency) or they are not. If M is consistent and valid, nothing is wrong with having ambiguity, except that you should state explicitly that all alternative interpretations are intended. Without this explicit statement, there is incompleteness.

Means. To avoid entering into a discussion about the development life cycle, modeling approach, or requirements-acquisition technique, we look at modeling only in terms of the actions that affect it directly. These include introducing statements to make the model more complete or deleting them to make the model more valid. Correcting a statement can increase both completeness and validity. Conceptually, you can view correction as a deletion followed by an insertion.

Most quality improvement requires that you manually inspect the model. For this reason, we classify many means traditionally associated with semantic goals as pragmatic because they directly support a clearer understanding of the model. However, consistency checking can often be performed automatically, without really understanding the model, which makes it a suitable semantic means. We consider consistency checking as a semantic means even though we do not consider it a semantic goal. The reason has to do with tool support. It is much easier to detect inconsistency than invalidity or incompleteness.

To do automatic consistency checking, the model must be in a formal language. To make it easy to insert and delete statements, the model should be modifiable. This includes properties such as structure and locality of change (performing semantic changes by changing only a few closely related statements).

PRAGMATIC QUALITY

While semantic quality affects choices

between models with alternative meanings, pragmatics affects how to choose from among the many ways to express a single meaning.

Goals. There is one pragmatic goal: comprehension. Not even the most brilliant solution to a problem would be of any use if no one could understand it. An important point in making the model understandable is that you must ensure that all concerned parties understand it, not just select groups. For this reason, we have addressed comprehension by introducing several model projections — $M_1, \dots, M_n$ in M — and a corresponding set of audience projections — $A_1, \dots, A_n$ in A . The projections represent different group's interpretations. With these modifications, we can define comprehension in a way that is similar to what we did for validity and completeness. Comprehension means that all model projections

have been understood by their relevant audience. That is $(\forall i) (M_i = A_i)$, meaning that all audience members completely understand the statements in the model that are relevant to them.

The corresponding error class is incomprehension, meaning that the formula does not hold. For a large model, you cannot realistically assume that each audience will be able to com-

prehend all the statements relevant to them. Thus, our definition of comprehension is an ideal goal, just as validity and completeness were unachievable goals in the face of reality. Again we introduce the notion of feasibility: *Feasible comprehension* means that, although all audience parts may not have correctly understood the model — which is stated as

$(\exists t) (A_i \setminus M_i) \cup (M_i \setminus A_i) = T_i \neq \emptyset$ — there is no statement t in T_i such that the benefit of rooting out the misunderstanding corresponding to t exceeds the drawbacks of taking that effort. T_i is the set of statements in the model rele-

**THE GOAL IS
NOT TO MAKE
THE MODEL
EASY TO
UNDERSTAND
BUT TO ENSURE
THAT IT IS
UNDERSTOOD.**



vant to a particular audience group that that group has misunderstood or is not aware of *plus* the set of statements the audience believes is in the model but is not. The index i stands for an arbitrary audience group. Hence, the equation $T_i \neq \emptyset$ means that the audience group has not completely grasped the part of the model that is relevant to them.

Note that we have stated the pragmatic goal as comprehension (the model has been understood), not as comprehensibility (the model can be understood). We did this for several reasons. First, the ultimate goal is that the model be understood, not that it be understandable. Second, it is difficult to quantify a model's comprehensibility because so much depends on how the model was developed, the way the analyst communicates with the end-user, and the degree of tool support.

Means. Pragmatic means are whatever makes the model easier to understand. Looking at the modeling activities first, inspection is the most primitive — obviously you must read the model to begin to understand it. For large models, however, this rapidly becomes less feasible. You most likely will need tool support like a browser to navigate through the model quickly.

Visualization lets you see the model diagrammatically, rather than textually, which often makes it easier to grasp. Animation goes one step further, using moving pictures to make system dynamics explicit. This might take the form of icons that resemble real-world objects, such as a telephone ringing, a customer arriving at a registration desk, etc. Animation is practically impossible without tool support.

Explanation, on the other hand, can be completely manual, as simple as an analyst explaining the model to an end-user. However, when models grow large, analysts themselves will have problems with comprehension, so tool support again becomes a factor. An explanation generator might answer questions about the model and its behavior.

Simulation uses statistical assumptions about the domain, such as the arrival rate of customers and distribution of process-

ing times, to anticipate how a system built according to the model would behave if implemented. It is not practical without tool support.

Filtering is necessary because you cannot grasp the entire model at once and must concentrate on specific parts or aspects at a time. Filtering may also include aspects of translation. Because a large model may have a rather diverse audience, different languages will be preferred by various groups. End users may want to see business rules in natural language, for example, while analysts may want to see them in logic.

The properties a model must have to make pragmatic modeling activities possible include those for syntactic and semantic quality, as well as executability, expressive economy, and structure. Executability is needed to automate activities like animation and simulation. Expressive economy makes the model easier to grasp because there are fewer statements to analyze. However, it may conflict with structure: A structured model is not always the most concise because the price for the structure may have been additional overhead. Because structure is useful for both modifiability and comprehension, you should not carry conciseness too far.

FEASIBILITY AND TOOL SUPPORT

As Figure 2 shows, feasibility is a consideration for all quality types. It involves asking questions like, has the model reached a satisfactory quality? If not, what quality goal is most worth pursuing at the moment and what means are most likely to be effective in reaching this goal?

While there is not enough room here to give a detailed analysis of feasibility issues, we can say something about considerations with respect to completeness and validity.

When you start modeling, your only problem is that the model is totally in-

complete. There are no problems with syntactic quality or validity because no statements have been made yet. Neither is there a problem with model comprehension — you can hardly misunderstand a model with no statements. Thus, if completeness is the sole problem, the only initial activity is to insert statements, lots of

them, either by writing them from scratch or reusing statements from other models.

So as not to inhibit the creativity needed to expand the model, the tool should allow for some degree of modeling freedom.⁷ This means that the model may temporarily contain syntactic and semantic errors.

When a model grows quickly like this, completeness increases but invalidity and incomprehension also increase. When incomprehension becomes too big, you can no longer expand the model because too many statements will be invalid. At this point, it makes more sense to begin a consolidating phase, with the objective of decreasing incomprehension and invalidity. Now statements are being deleted or corrected, and there is an emphasis on analytic activities like consistency checking and simulation. During this phase, comprehension and validity go up, but completeness is at a near standstill. After a while, comprehension becomes so good that the gain of additional consolidation is marginal. Once again, it makes more sense to return to an expansive phase to reduce incompleteness.

Hence, you can view modeling as a pendulum swinging from expansion to consolidation and back again. This view supports our two-goal framework also, since validity requires consolidation and completeness requires expansion.

State-of-the-art modeling tools support many of the activities we have mentioned, such as syntactic error detection, consistency checking, inspection, filtering, animation, and explanation. In fact, activities like consistency checking and

**FEASIBILITY
ANALYSIS IS
HARD TO
AUTOMATE.
YOU'LL
ALWAYS NEED
SOME GUT
INSTINCT.**

simulation are not feasible without tools.

For feasibility analysis, however, there is little in the way of tool support. Such tools will be difficult to develop for the simple reason that the domain and audience, important elements in determining model feasibility, are beyond automatic manipulation. Hence, you can never know if the model is feasibly complete or understood. There will always be room — and need — for gut instinct.

Moreover, as the process-maturity movement has shown us, tools may be good for automating tasks that we already know how to do, but tools cannot help if we do not understand the development process. Thus, we are not as concerned with supporting the means to achieve quality as we are in reaching an understanding of quality and of how to set realistic quality goals.

Previous attempts to describe the properties of a quality conceptual model have amounted to little more than bread-and-butter collections of useful features. Our framework establishes qualitative relationships between central goals and means, leading to a deeper understanding of quality as it relates to conceptual modeling.

By including the notion of feasibility explicitly in the framework we provide realistic quality-assurance goals. We recognize that our contribution is at a rather high level of abstraction. Still it is a good starting point for developing systematic, practical guidelines on how to achieve syntactic, semantic, and pragmatic quality. To make our framework more useful in a practical modeling situation, we must refine it in several ways:

- ◆ *Specialize to a language or method.* The framework is not tied to any particular language or method, which is one reason it is now so abstract. Looking at the three types of quality in the context of a particular language or method would help us provide more concrete guidelines.

- ◆ *Formalize means.* Although we have given goals as a formal definition in set algebra, we have discussed means only informally. The next step is to describe

them in some formal notation as well.

- ◆ *Decompose goals.* To make the framework clear, we proposed only a few goals, subsuming many previously identified goals. For detailed guidelines, a more fine-grained breakdown would be useful. For example, there are several kinds of completeness: functional (all functional requirements have been captured), nonfunctional (all nonfunctional requirements have been captured), descriptive (all descriptive statements — those not in the form of requirements — have been captured), and traceability (all cross references for traceability have been included). We would add to all these the notion of feasibility.

- ◆ *Quantization.* So far, the set-algebra equations state only if a model has met a goal. We could build on that by proposing to-what-extent measures of validity and completeness. We would, of course, want to join such efforts to other work in metrics.

Thus, we are not proposing a new method for conceptual modeling or requirements engineering as much as we are providing a way to think about quality as it relates to these things. We see this as a necessary step toward improving the effectiveness of future modeling efforts. ◆



Lindland received an MSc and a PhD in computer science from the University of Trondheim.



Trondheim.



Solvberg received an MSc in applied physics and a PhD in computer science from the University of Trondheim.

Odd Ivar Lindland is an associate professor of computer science at the University of Trondheim, NTH. His interests are conceptual modeling, CASE tools, validation, and prototyping.

Lindland received an MSc and a PhD in computer science from the University of Trondheim.

Guttorm Sindre is an associate professor of computer science at the University of Trondheim, NTH. His interests are conceptual modeling, visual languages, and software reuse.

Sindre received an MSc and a PhD in computer science from the University of

Arne Solvberg is a professor of computer science at the University of Trondheim, NTH. His interests are information systems design methodologies, database design, information modeling, CASE tools, and software-engineering environments.

Send questions about this article to Lindland at Dept. of Computer Systems and Telematics, Norwegian Institute of Technology, 7034 Trondheim, NTH, Norway; oil@idt.unit.no.

ACKNOWLEDGMENTS

We thank anonymous IEEE Software referees for their insightful and valuable comments. We also thank our colleagues Jon Atle Gulla, John Krogstie, Øystein Nytro, and Geir Willumsen for valuable discussions when we first began the draft for this article.

REFERENCES

1. C. Kung, "An Analysis of Three Conceptual Models with Time Perspective," in *Information Systems Design Methodologies: A Feature Analysis*, T. Olle et al., North-Holland, Amsterdam, 1983, pp. 141-168.
2. G. Roman, "A Taxonomy of Current Issues in Requirements Engineering," *Computer*, Apr. 1985, pp. 14-22.
3. R. Yeh et al., "Software Requirements: New Directions and Perspectives," in *Handbook of Software Engineering*, C. Vick and C. Ramamoorthy, eds., Van Nostrand Reinhold, New York, 1984, pp. 519-543.
4. A. Davis, *Software Requirements: Analysis and Specification*, Prentice-Hall, Englewood Cliff, NJ, 1990.
5. R. Thayer and M. Thayer, "Glossary," in *System and Software Requirements Eng.*, R. Thayer and M. Dorfman, eds., IEEE CS Press, Los Alamitos, Calif., 1990, pp. 605-677.
6. A. Solvberg and D. Kung, *Information Systems Engineering*, Springer-Verlag, Berlin, 1993.
7. M. Feather, "Requirements Engineering — Getting Right from Wrong," *Proc. European Software Eng. Conf.*, A. van Lamsweerde and A. Fugetta, eds., Springer-Verlag, Berlin, 1991, pp. 485-488.