Zeitschrift: Studies in Communication Sciences : journal of the Swiss Association of Communication and Media Research

Herausgeber: Swiss Association of Communication and Media Research; Università della Svizzera italiana, Faculty of Communication Sciences

Band: 4 (2004)

Heft: 2

Artikel: A framework for addressing the quality of UML artifacts

Autor: Kamthan, Pankaj

DOI: https://doi.org/10.5169/seals-790976

Studies in Communication Sciences 4/2 (2004) 85-114

Pankaj Kamthan*

A FRAMEWORK FOR ADDRESSING THE QUALITY OF UML ARTIFACTS

The Unified Modeling Language (UML) is a standard language for modeling the structure and behavior of object-oriented systems. In recent years, there has been a rapid increase in the use of UML artifacts in a variety of application areas. As the use of UML becomes pervasive, the quality of UML artifacts as effective means of communication arises. The aim of this paper is to contribute to a systematic assessment and assurance of the quality of UML artifacts. A quality framework for UML artifacts using notions from semiotics, human-computer interaction, and technical aspects of software diagramming, is proposed. The goals for quality and mechanisms to address them are identified. The mechanisms are themselves analyzed with respect to their usefulness in achieving the quality goals of the artifact. Examples of UML artifacts that compromise quality, and techniques for improvement, are given.

Keywords: communicability, feasibility, graph drawing, modeling quality, representation, semiotics.

*Department of Computer Science and Software Engineering, Concordia University, Montreal, kamthan@cse.concordia.ca

1. Introduction

Modeling has become an integral activity in software development. After embracing a variety of notations and languages over the years, the software industry has now adopted the Unified Modeling Language (UML) (Booch et al. 2005; Fowler 2003). UML is a semi-formal language for structural and behavioral modeling that provides diagram types applicable to a wide variety of domains. The UML has been used for modeling activities in a software process as well as representing snapshots of product under development (Lethbridge, Laganière 2001), modeling embedded systems consisting of software and hardware components (Green, Edwards 2004), modeling real-time systems (Douglass 2004), modeling Internet applications (Carlson 2001; Conallen 2003), and modeling business activities (Eriksson, Penker 2000), to name a selected few.

According to the ISO 8402 Standard (ISO 1994), quality is defined as the totality of features and characteristics of a product that bear on its ability to satisfy stated or implied needs. An investigation into of the quality of UML artifacts is necessary for various reasons. UML has begun to play an increasingly central role in activities and deliverables of adaptive software process environments such as Extreme Programming (XP) (Beck 1999) and the Rational Unified Process (RUP) (Kruchten 2004). These model-driven approaches create UML artifacts during requirements and design, and therefore addressing the issue of quality early is crucial (Lindland et al. 1994; Krogstie 1998; Shanks et al. 2003) from the point of view of control and prevention of problems that can propagate into later stages. If left unattended, these artifacts may, for example, fail to communicate their purpose (Arlow et al. 1998), could be misleading, or be virtually non-modifiable. This would undermine the basic philosophy of UML to unify multiple notations that were potentially threatening interoperability among tools and communicability among engineers, and can adversely affect further acceptance and growth of UML.

There has been a surge of activity in the last decade in understanding the notion of quality in modeling. Efforts to approach it in a systematic manner have led to introduction of frameworks for information quality (Eppler 2001) in conceptual modeling (Lindland et al. 1994; Krogstie 1998; Shanks et al. 2003) and data modeling (Reingruber and Gregory 1994; Shanks 1999; Moody and Shanks 2003).

In this paper, we address the issue of quality in UML artifacts based on the ideas from semiotics, human-computer interaction, and graph drawing. In doing so, we propose a framework as a first step towards understanding, assessing, and ensuring the quality of UML artifacts. The outline of the paper is as follows. Section 2 provides the infrastructure of the framework. Section 3 considers the application of the framework with respect to the semiotic levels. Finally, Section 4 concludes with some recommendations and remarks towards future research.

2. Foundations of a Framework for Quality of UML Artifacts

Semiotics is concerned with the use of symbols to convey knowledge and forms the basis of many of the information quality frameworks mentioned previously. According to (Morris 1938/1970), there are three non-mutually exclusive semiotic levels: semantics (the relationship of signs to what they stand for), syntactics (the formal or structural relations between signs), and pragmatics (the relation of signs to interpreters). Further elaborations to other semiotic levels (namely physical, empirical, and social) for analyzing symbols (Stamper 1992) with finer granularity have been discussed in contexts of information quality (Krogstie 2001; Shanks 1999). We do not view any UML-oriented specifics that would require special treatment of these levels and will therefore restrict ourselves to only the three basic semiotic levels as in (Lindland et al. 1994; Gurr 1999).

We adopt a goal-driven, realistic ontological view towards modeling quality: the granularity of entities relevant to a UML artifact is determined by the goal and is necessarily finite. Then, based on the software engineering principle of separation of concerns, we have the following general methodology as the foundation to the quality framework (Figure 1):

1- Given a UML artifact, list the entities directly related to it, and state relevant relationship types including corresponding semiotic levels of interest;
2- Identify the quality goal(s) for each of the semiotic level;
3- If a goal from (2) is at a too high level to be addressed directly, decompose it further into a manageable list of specific criteria;
4- State the mechanism(s) for achieving the criteria. A mechanism can correspond to one or more criteria.

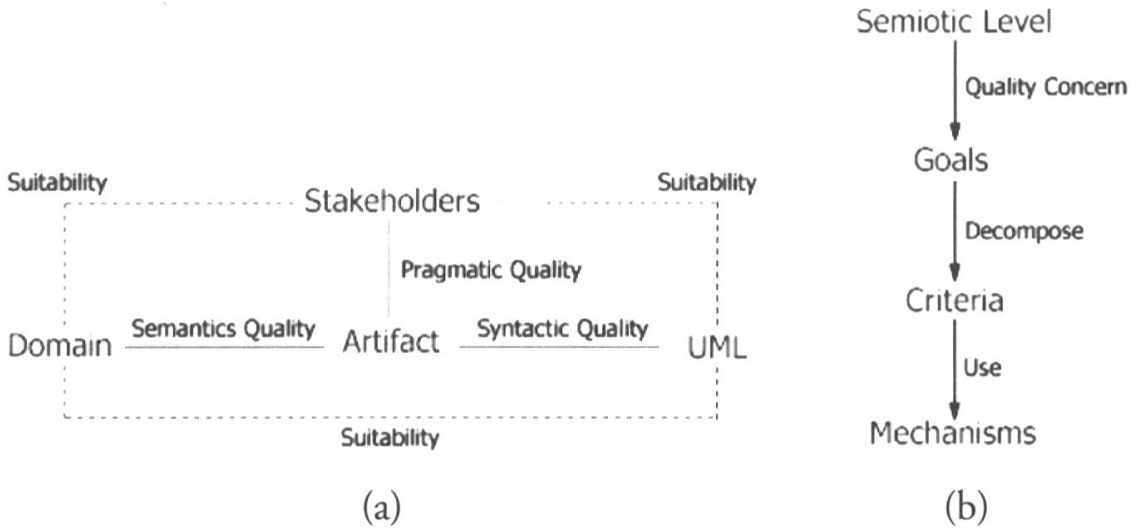

Figure 1: A simplified illustration of the UML artifact quality framework.

From a high-level, as shown in Figure 1(a), the architecture of the quality framework can be viewed as a collection of entities and (non-reflexive) relationship types. At a low-level, illustrated by Figure 1(b), the interest is in the properties of the main relationship types. We now exclusively look at these two aspects for the remainder of the section, starting with the macro-architecture.

2.1. Macro-Architecture of the Quality Framework

In this section, we describe the entities of the framework in detail and then deal with the quality concern of each of the identified semiotic levels.

2.1.1. Entities

As Figure 1 shows, an artifact is intimately related to the domain, the modeling language, and the stakeholders. We therefore begin with the discussion of these entities.

Artifact
An artifact is an abstract, often simplified, representation of some real-world entity that consists of statements made in a modeling language, which in our case is UML.

Domain

A domain is an aspect under study. A UML artifact will usually be targeting some application area (real-time systems, information systems, and so forth), which could belong to a problem (requirements, analysis) domain or a solution (design) domain in software, and could address either the static (structural) or the dynamic (behavioral) aspect of the system.

UML as a Modeling Language

UML is a standard defined by the Object Management Group (OMG) where the current adopted version is 2.0. Over the years of use, several shortcomings of UML have been exposed (Morris and Spanoudakis 2001) that can impact the quality of an artifact based on it; this is elaborated in the following as necessary. An evaluation of UML with respect to the semiotic framework of (Lindland et al. 1994; Krogstie 1998) is given in (Krogstie 2001).

Since its inception in the late 1990s and its roots in software design domain, the evolution of UML has come a long way. There have been three noteworthy initiatives during this period that are relevant with respect to quality of UML artifacts: support for extensibility, meta-modeling, and the movement towards single source (executability).

To accommodate emerging areas where modeling plays a crucial role, properties of existing diagram types have been reviewed and modified accordingly, and the number of diagram types has been increased. Action Semantics UML Extensions, that let an engineer express actions as UML objects, have provided much needed support for distributed computing environments. In the case of embedded systems, the introduction of new diagram types in UML 2.0 has facilitated communication between software and hardware designers and between system engineers and project stakeholders. It was also realized within OMG that, at any given point in time, the default support provided by UML constructs might not be sufficient for all domains. The introduction of the UML profiling mechanism has enabled extensions of UML in the form of application-domain-specific profiles. Some of the profiles of broad interest and applicability, such as that for enterprise application integration, have been standardized by OMG.

By acknowledging that UML is only one of the many possible types of modeling languages, UML has been placed within the context of meta-modeling as defined by the Meta Object Facility (MOF) that is part of a higher level of abstraction of the Model Driven Architecture (MDA)

framework (Kleppe at al. 2003). This has improved interoperability among tools and fostered design reuse, thereby potentially increasing productivity.

One way of optimizing effort, minimizing redundancy and possibility of errors, is to have the control relegated to a single source. Given a UML design artifact, it is possible today to generate source code (although, being general-purpose, it may need modifications) for certain programming language bindings. This has been further promulgated with the proposals such as Executable UML (Mellor and Balcer 2002) for using tools to directly translate abstract application constructs into executable entities, thus removing the layer for coding altogether.

Stakeholders

This is the set of participants (people or programs) that have a stake in the artifact. These participants can be in the (non-mutually exclusive) roles of producers and consumers, including users. An engineer architecting the artifact and a UML modeler belong to the category of producers. A project manager, an end-user, and an image renderer belong to the category of consumers.

The entities in the Figure 1(a) of the framework are not isolated and are intimately related to each other in many ways that gives rise to a connected graph. These relationship types are discussed next.

2.1.2. Relationship Types

The relationship types in the framework that involve the artifact directly are considered as *primary* and the ones that impact the artifact indirectly are considered as *secondary* (indicated by solid and dashed lines in Figure 1, respectively).

The primary relationship types are Artifact-Syntax, Artifact-Semantics, and Artifact-Pragmatics.

Artifact-Syntax. This relationship represents the syntactic level of the artifact and is concerned with the form of symbols. The quality consideration at this level is the *Syntactic Quality*.

Artifact-Semantics. This relationship represents the semantic level of artifact and is concerned with the meaning of symbols. The quality consideration at this level is the *Semantic Quality*.

Artifact-Pragmatics. This relationship represents pragmatic level of artifact and is concerned with the practical knowledge needed by the artifact to use UML for communicative purposes. To do that requires choosing from among the given possibilities in the contextual usage of symbols to express a single meaning. The quality consideration at this level is the *Pragmatic Quality.*

The relevant secondary relationship types are Domain-UML Suitability, Domain-Stakeholder Suitability, and UML-Stakeholder Suitability.

Domain-UML Suitability. Any evaluation of appropriateness must be carried out along two lines: (1) the scope of the UML as given by the normative specification, and (2) the criteria for evaluation. The unification of UML and aggressive marketing has led to the use of UML in domains that were never in its scope (the "one-size-fits-all" syndrome). UML has been heralded (Larman 2002) as a conceptual (domain) modeling language in software engineering. But without a formal (mathematical) foundation for its syntax and semantics that is desirable for complex reasoning tasks and computational tractability, UML is not suitable for such purpose. Such instances should hardly be seen as weaknesses of UML but rather as its improper use. However, there are limitations of the use of UML in domains that are supposed to be within its grasp. For example, although there is much support for the use of Use Cases with UML (Rosenberg and Scott 1999), it has been shown that UML graphical notation alone is not sufficiently expressive in representing complex cases (Glinz 2000). Furthermore, it is not also readily possible to indicate universally unique identifiers in UML. Therefore, despite the claims (Botaschanjan et al. 2004), UML is not powerful enough to entirely represent the problem domain (and therefore is not a complete requirements specification language).

Domain-Stakeholder Suitability. Ultimately every UML artifact is supposed to represent some concept in the domain. Therefore, the stakeholder knowledge of the domain and conversely the relevancy of the domain to a stakeholder, are necessary.

UML-Stakeholder Suitability. A stakeholder must have appropriate knowledge of UML for either producing or consuming the resulting artifact.

Having examined the entities and the relevant primary and secondary relationship types from a high-level view, we now focus specifically on addressing the three levels of quality in the primary relationship types.

2.2. Micro-Architecture of the Quality Framework

The quality consideration of each semiotic level is associated with a goal, which can often be met in a concrete manner only when addressed at lower-level criteria. These criteria may be external to an artifact as it interacts with its environment or internal to an artifact when it is considered in its own right without the impact of any external influence. Furthermore, these criteria can be dealt with realistically when there are certain mechanisms (such as principles, methods, techniques, or tools) in place. We now look at each of these aspects in depth.

Goals

A project without clear goals will not achieve its goals clearly (Gilb 1988). A goal captures the intent of the artifact with respect to quality. A goal may or may not be entirely satisfied (at all, or within the stipulated time within the artifact life cycle) and a feasibility analysis for an acceptable level of realization may be necessary.

Feasibility Analysis

UML artifacts are often the means to the end (executable software) and their quality control carries extra cost (in form of time, effort, and resource commitment) on top of considerations for software quality. We also need to prioritize and make trade-offs among the criteria and corresponding mechanisms. Therefore, an economic, organizational, and technical feasibility analysis (Boehm et al. 2001) is necessary for a realistic realization of the quality goal. One of the criticisms of widely-used goal-based measurement programs in organizations such as the Goal-Question-Metrics (GQM) method (Van Solingen and Berghout 1999) is that they do not address the issue of feasibility. We view feasibility as an all-encompassing layer that manifests itself on almost all aspects of the UML quality framework in order to make it practical.

Criteria

The criteria are manageable quality characteristics and can be divided into two parts: the internal and external attributes of a UML artifact, as

in ISO/IEC 9126-1 Standard (ISO 2001). An external attribute is an extrinsic property of an artifact that can be addressed only on the basis of how it relates to its environment. An internal attribute is an intrinsic property of an artifact that can be addressed purely on its own irrespective of its environment. Often, external attributes can be expressed in terms of internal attributes (Fenton and Pfleeger 1997). The criteria could be associated with weights that could represent significance or priority given by the organization or artifact producer.

The relevant external attributes are User Preference, Clarity, Consistency, Simplicity, Familiarity, Interoperability, and Standardization.

User Preference. Studies have shown (Purchase et al. 2001) that users, due their beliefs or otherwise, may prefer one diagram type to the other for the sake of understanding, even though they may be semantically equivalent for most purposes (like the UML Sequence Diagrams and Communication Diagrams).

Clarity. By clarity, we mean either semantic clarity (sensible use of UML notations) or visual clarity (legibility of graphic or readability of text).

Consistency. By consistency, we mean any activity that is not ad-hoc and depicts coherence. This includes visual coherence and making use of conventions where necessary.

Simplicity. We can view simplicity as a complement of complexity: anything that leads to non-determinism (unpredictability) from a human viewpoint is usually considered as complex.

Familiarity. By taking a subjectivist epistemological position (Shanks 1999), we accept that our understanding of the world depends on our prior knowledge and experience, and therefore introduce familiarity as one of the attributes.

Interoperability. The same artifact could be processed and viewed by different tools. Interoperability requires the artifact to work in all environments it is supposed to.

Standardization. Standardization, when applied well, is known to contribute towards quality improvement (Schneidewind and Fenton 1996).

The relevant internal attributes are UML Primary Notation, UML Secondary Notation, Size, Structure, and Format.

Primary Notation in UML. The UML defines four kinds of primitive elements: icons, 2D symbols, paths, and strings. These are used in combination to create compound structures such as the 13 diagram types that UML 2.0 defines. An UML artifact essentially extends this notation.

Secondary Notation in UML. The secondary notation (Petre 1995) in graphical programming is defined as the use of layout and perceptual cues to clarify information or to give hints to the stakeholder. The UML secondary elements that affect the comprehensibility of artifacts are color, directionality, labeling, level of abstraction and refinement, morphology, positioning, typography, and white space.

Size. By size of a UML artifact, we mean both the area (dimensions) that a UML artifact occupies and the file size. The former will depend on the use of the secondary notation (morphology and white space). The latter will depend on area of a UML artifact and the export format being used. UML artifacts for similar projects may reuse existing constructs in whole or in part, verbatim or slightly modified. It will of course be important that reused constructs blend in well with the newer ones.

Structure. By structure, we mean the geometric structure of a UML artifact. There are structural patterns such as symmetry and anti-symmetry that impact the quality of UML artifact. The structure of a UML artifact will depend on the use of the secondary notation (morphology, positioning, and white space), and to the extent there is internal reuse and coupling. The generalization/specialization relationships lead to internal reuse. Low coupling is a hallmark of "good" design.

Representation Format. In an electronic production of UML artifacts, an author is usually working with some text (vector) or binary (raster) data format rendered as a graphic. The nature and choice of a format can directly impact the quality of a UML artifact during production and subsequent transmission. Although usually more compact than (uncompressed) text, images in raster formats such as Graphical Image Format (GIF) or Portable Network Graphics (PNG), on magnification tend to

have an incarnation of the "staircase effect" and are non-interactive. From applications of Gestalt psychology to graph drawing (Di Battista et al. 1999), it is known that humans more easily see smooth continuous contours than jagged ones. On the other hand, UML artifacts serialized in vector graphical formats based on Extensible Markup Language (XML) (Bray et al. 2000), a meta-language that provides directions for expressing the syntax of markup languages, circumvent the issues of raster formats and lend themselves to the benefits that are associated with descriptive markup such as support for metadata, hyperlinking, intimate control towards printing, and syntax checking.

XML Metadata Interchange (XMI) (Brodsky and Grose 2002) and Scalable Vector Graphics (SVG) (Ferraiolo et al. 2003) are two XML-based markup languages that are particularly relevant to UML artifacts. XMI, an OMG standard and part of MDA, is a serialization format for UML with the goals of artifact interchange and cross-tool interoperability. SVG is a mixed raster-vector graphical format with the goal of providing means for standard graphics on network devices, including desktop browsers and mobile telephones. (All figures in this paper are based on SVG.) Both these languages inherit aforementioned benefits of XML. XMI-based UML artifacts can also be down transformed into SVG. To a producer, UML artifacts in SVG could be annotated, ported across networks, archived, and maintained with relative ease. To a consumer, they offer legibility (at virtually any level of magnification) and involvability (sophisticated means of interaction, navigation, and searching on virtually any platform). The quality of the means of transformation (script, style sheet, or program) and the resulting markup in SVG graphic is important in its own right, and beyond the scope of discussion here.

Mechanisms

A mechanism is technical or other means that could help improve the quality of the UML artifact. The mechanisms themselves are non-necessarily mutually exclusive. They can be classified in various ways: static or dynamic verification; theoretical or practical; applicable before, during, and after the first iteration of the artifact production process; applicable for quality assessment (evaluation) and/or for quality assurance. There is an inevitable cost involved in the use of any mechanism. Therefore, the scope and limitations of the mechanisms should be discussed for the sake of objectivity and to provide a benchmark for feasibility analysis. Tool

support is necessary to assist some of the mechanisms. We realize that for appropriate application, the stakeholder (producer) knowledge of the mechanisms is important in its own right, and will not address this here.

Training in Use of Primary Notation. Like the use of any other language, acquiring the knowledge of UML would require training. Apart from the publicly available official UML Specifications from OMG, there are other resources in print (Booch et al. 2005; Fowler 2003) or otherwise available via electronic means. However, the three main obstacles that continue to hamper the level of UML training required today are (1) that there are almost no textbooks that provide good exercises (if at all) to practice, (2) with the University curricula already crowded, courses specific to UML are rarely offered in the mainstream, and (3) professional training is often centered around expensive venues in large cities and the cost tends to be beyond the budget of an average student.

Training in Use of Secondary Notation. The producers of UML artifacts need to be trained in the basics of interface design (Shneiderman and Plaisant 2005), information design (Jacobson and Wurman 1999), and interaction design (Preece et al. 2002), which are the mechanical, analytical, and conceptual parts of artifact design, respectively, to get a detailed understanding of the UML secondary notation. We provide one view of appropriate use of the secondary notation:

- *Color.* By associating different colors with constructs in a complex figure, a stakeholder can be informed of the semantic similarity and differences between the constructs with respect to both their structure and behavior. For example, use of color in UML Class Diagrams for Java has been reported to improve the user understanding of the overall design (Coad et al. 1999). Any use of color, however, should take into account the variations in the interpretation of primary colors by computer monitors, contrast between background and foreground, the way people with color vision deficiency view an image (Rigden 1999), and the possibility that diagrams may be printed on a black and white printer.
- *Directionality.* Directionality in UML constructs is critical, especially when representing the progression in time, movement across space, and to illustrate hierarchy. For example, an arrow in a generalization/specialization relationship points upward from the spe-

cialized case to the generalized case to show prominence to the latter. The normative versions of the UML Specifications are in English. The cultures where English is the primary language of use, people usually tend to read from left to right and from top to bottom, and to ease readability long phrases and multiple lines of text need to reflect that. Inevitably, this depends on the positioning of UML constructs.

- *Labeling.* Use of application domain terminology in text labels makes it easier for a non-technical stakeholders or users new to an author-defined UML extension to become familiar with the artifact. A naming convention, such as use of camel case, could be followed consistently.
- *Level of Abstraction and Refinement.* Not all UML constructs are appropriate for exposure to all stakeholders at all times. For example, software macro-architecture using UML Package Diagrams may be much more accessible to a stakeholder interested but not directly involved in design, than the use of UML Class Diagrams. Also, following the well-known 7 +/- 2 principle of short-term memory (Miller 1956), diagrams that are structurally complex could be split into multiple parts, where only the relevant parts are exposed.
- *Morphology.* The morphology or shapes of nodes and vertices in a UML artifact have an impact on how the diagram as a whole is perceived by the user. It is known in graph drawing (Di Battista et al. 1999) that presence of crooked nodes and zigzag vertices are aesthetically unpleasing and cognitively ineffective. In general, users also associate significance with the size of nodes and vertices in a diagram (Comber 1993), which therefore need to be consistent and semantically meaningful.
- *Positioning.* Humans associate positioning of graphical constructs to spatial and temporal relationships. The legibility and stakeholder interpretation of a UML artifact are affected by relative proximity of its nodes and vertices. Structural patterns in relationships such as symmetry and anti-symmetry are visual cues for familiarity and need to be preserved. The proper placement of a text label that is suppose to belong to one node is also important to reduce any ambiguity on part of the reader when placed between two nodes. UML modelers and image renderers in general use the top-left corner of the drawing canvas as the point of origin and reference for their coordinate system and, for the sake of clarity and familiarity, the UML artifacts need to reflect that.

- *Typography.* The choice and the sequence of characters in the use of text (annotation and labels) affect readability. For example, the characters in a name like O0lI1 are hard to distinguish and therefore difficult to read. The choice of fonts used for annotation and labeling depends on a variety of factors (serif versus sans-serif, amount of kerning, font size, and so forth) that are important for legibility. Fonts specifically designed for presentation on paper may in general be hard to read on a computer screen.
- *White Space.* One of the traits of any diagramming style is the introduction of white space at appropriate places. In UML artifacts, white space can be added between nodes, between nodes and vertices, and between labels and boundaries of UML constructs for visual clarity. Shape and positioning of nodes and vertices, and the use of white space complement each other.

The lessons of UML training usually culminate in some form of guidance such as guidelines, frequently asked questions, or patterns. The UML secondary notation is the basis of several style guidelines (Ambler 2003) and patterns (Evitts 2000) specific to UML diagram types. We therefore briefly discuss the utility of these two instruments.

Guidelines. Guidelines have a successful history in streamlining production efforts, and in providing a basis for comparison. The classical guidelines from graph drawing that determine a "good" graph (such as minimizing crossings, optimizing area, minimizing the number of the vertex lengths, keeping vertex length uniform, minimizing bends), also apply to UML artifacts. Guidelines could also serve as a checklist with respect to which the quality of UML artifacts could be evaluated, however, to be effective, the guidelines need to provide an ordinal scale, that is, a *spectrum* (instead of the often Boolean, yes or no) of conformance levels. Guidelines seem to be more useful for those with an expert knowledge than for a novice to whom they may seem very general to be of much practical use.

Patterns. Patterns (Alexander 1979; Lethbridge and Laganière 2001; Larman 2002) are reusable entities of knowledge and experience aggregated by experts over years of "best practices" in solving recurring problems in a domain. Formally, a pattern expresses a relationship between a certain context, a recurring problem, and a proven solution. Patterns are

relatively more structured than guidelines, illustrated by examples and non-examples (anti-patterns), and provide better opportunities for sharing and reuse. Since patterns are restricted by their context, they are specific, and tend to provide concrete solutions that are broadly applicable. There are certain caveats in the pattern approach. According to the COCOMO II cost estimation model (Boehm et al. 2001), reuse comes with a cost of adaptation to new contexts. Before it is documented for public use, pattern discovery requires considerable experience with the domain and its validation needs acceptance of a broader community.

Metadata. Metadata, such as in form of annotation, can provide further explanation on items that are not immediately obvious. Annotating UML artifacts using the UML Note construct can be particularly useful when UML constructs that may not be common or known to the user (such as when UML extensions are being used for the first time), or when stakeholders involved do not have the necessary technical knowledge. By including expressions in Object Constraint Language (OCL) (Warmer and Kleppe 2003), a formal language for UML that is part of MDA and allows software engineers to write constraints and queries over object models, such notes can also add to the intent and semantics of the diagram. These notes could, for example, include the details of pre- and post-conditions of Use Cases in OCL; something that is not possible via UML graphical notations alone.

Pair Modeling. Complex, large-scale artifacts in traditional engineering are usually crafted not by an individual but by a team of architects. To that regard, and inspired by the idea of Pair Programming (Beck 1999; Williams and Erdogmus 2002), we introduce the notion of *pair modeling* where two people participate in crafting a UML artifact. Part of pair modeling can also be viewed as an informal, lightweight monitoring of UML artifacts: every item being drawn (or text being written) is under scrutiny by the partner during the creation process. Since stakeholders from both early and later phases have a stake in producing a "good" artifact, such an activity tends to be cooperative rather than confrontational. The authors applied this cost effective technique with students from academia and industry, resulting in better mutual cooperation and improved productivity. However, for pair modeling to be successful, both the partners need to have a similar level of experience with the application domain and with UML itself.

Refactoring. UML artifacts may need to evolve for various reasons such as discovery of impurities (symptoms of problems or "smells") or obsolescence (due to changes in UML Specification or that in the parent software objectives). Refactoring is a technique of transforming the internal structure of software for the sake of understandability and maintainability, while preserving its observable behavior (Fowler et al. 1999). Refactoring methods are transformations that provide a systematic way of eradicating the undesirable smells from an artifact while preserving its semantics (observable behavior). In the last decade, refactoring has been extensively applied to the context of source code, and more recently to UML artifacts (Sunye et al. 2001). Refactoring is beginning to have support among UML modelers. However, an issue that remains largely unaddressed in UML refactoring is formally proving the invariant properties of the refactoring methods. It is not always easy to demonstrate the business benefits to management who are usually more inclined towards addition of new features. Often the smell-refactoring methods mapping is not clearly stated. In cases when the UML artifacts are bound to the source code, any refactoring must also take into account any change propagation on modification of the UML artifact and make commitments towards relationship management throughout the software process.

Inspection. Inspection (Ebenau and Strauss 1994) is a rigorous form of auditing based upon peer review that, when practiced well, can help in error prevention in UML artifacts. Inspections have proved to be an effective technique in improving the overall quality of UML Class Diagrams (Conradi et al. 2003). However, there are issues of the cost-benefit ratio and the dependency of adoption based upon organizational process maturity. There is an initial cost overhead, as each participant needs to be trained in the structured review process and there are logistics of checklists, forms, and reports involved. In some process maturity models such as the Capability Maturity Model (CMM) (Paulk et al. 1993), adoption of inspections amounts to achieving at least Level 3, to which a considerable number of organizations do not qualify today.

Testing. Testing is a dynamic verification technique that can ameliorate the tedium involved in human-oriented examinations such as reviews and inspections. For example, dynamic testing of UML design artifacts in which behavioral models are executed can reveal defects at the design level before they are implemented (Trong 2003). Executable UML lends

itself well to testing. The drawbacks of testing UML artifacts are that not all aspects can be tested automatically and that the test suites and harnesses necessary for a thorough testing are not available today.

Measurement. Measurement can provide a means for quantitative evaluation of quality. There are direct and indirect measures for UML that apply to an artifact as a whole as well as to specific diagram types and individual constructs (Chidamber and Kemerer 1994; Harrison et al. 1998; Genero et al. 2002; Kim and Boldyreff 2002; Rufai 2003). These metrics give an assessment of structural complexity of a UML artifact, such as, the number of nodes (classes, states, Actors, Use Cases, Packages, and so forth) and vertices (associations), amount of internal reuse or the lack thereof (inheritance, polymorphism) or degree of coupling; behavioral complexity by counting the number of associations in a Use Case Diagram, messages in a UML Sequence Diagram, entry/exit actions and transitions in a UML State Machine Diagram.

The use of metrics to get informed about quality criteria faces some obstacles. The main challenge in the use of metrics is that improvement in one measure can lead to deterioration in another. In absence of weights of significance associated with each metric (which should not be prescribed apriori by the producer of the artifact) we cannot always make a unique informed decision. For example, if reducing the number of nodes in a diagram means distributing them over more than one diagram, then it cannot be trivially concluded that we have reduced the overall complexity. Most of the metrics are introduced and used on empirical grounds, and are not formally validated against the representational theory of measurement (Fenton and Pfleeger 1997). Not all metric suites provide a precise measurement scale. For non-trivial artifacts, manual calculations using metrics are tedious and require tool support. However, support for metrics in UML modeling tools is not widespread.

Tool Support for Automation and Modeling. UML syntax-sensitive tools or modelers can assist in successfully realizing the other mechanisms of achieving semiotic goals in practice. The public availability of UML Specifications has allowed tool builders that may not be members of OMG to enter the arena and there are several UML modelers in use today. An author's expertise of crafting UML models is ultimately only as good as the tool being deployed. (This is a significant departure from programming where the source code does not have to be bound closely to any specific editor.)

However, as some surveys have shown (Robbins 1999, Krogstie 2001), there are stark differences between commercial and non-commercial tools with respect to their features (conformance to official definition and versions of UML (Eichelberger 2003), implementation of layout algorithms (Purchase et al. 2001), flexibility in altering properties of UML constructs, available import/export formats, support for guidelines, patterns, automatic refactoring, and metrics). This can directly or indirectly affect the quality of a UML artifact. In a recent empirical evaluation of UML modeling tools based on Software Usability Measurement Inventory (SUMI) (Kirakowski 1993) at our institution, it was concluded that high-end commercial tools tend to provide a high degree of user satisfaction. The choice of the modeler being used is therefore crucial. Apart from the features and ergonomics of the modeler, a justified choice must also include considerations of sustainability of the modeler and its degree of neutrality with respect to the artifact import/export format. One of the challenges faced today in the use of most UML modelers is that they do not provide support for other artifacts that are created independently using different means.

With the details of the quality framework in place, we can now address the specifics of applying it to the three levels of semiotics.

3. Addressing Semiotic Quality of UML Artifacts

In this section, we discuss how the entities of the micro-architecture manifest themselves. For each semiotic level, the corresponding goal(s) are given and the scenarios when the goals are not complied with are described. The criteria and applicable mechanisms to realize the goals are then elaborated. Table 1(a)-(c) together summarize the treatment of quality of UML artifacts with respect to the syntactic, semantic, and pragmatic semiotic levels. In that order, the view of quality, and to a certain extent the degree of difficulty of dealing with it, moves in three orthogonal directions: from internal (artifact with no environment) to external (artifact in presence of environment), from single to multiple at a given time, and from non-dynamic (static or constant) to dynamic (variable).

3.1. Syntactic Quality

Syntactic quality is a measure of how well an artifact accommodates its contract with a single constant entity - the UML. The goals, criteria, and

mechanisms that we now describe are driven by this static internal view (Table 1(a)).

Table 1(a): Detailed Outline of the UML Quality Framework: Syntactics.

Goals	Criteria		Mechanisms	
Correctness	**External Attributes** ▪ UML-Stakeholder Suitability ▪ Simplicity ▪ Interoperability ▪ Standardization	**Internal Attributes** ▪ Primary Notation ▪ Format	**Quality Assurance** ▪ Training in Use of UML ▪ Pair Modeling **Quality Evaluation** ▪ Inspection ▪ Measurement o Metrics	Tool Support
			Feasibility Analysis	

Goals. There is only one goal for syntactic quality of a UML artifact: syntactic correctness (Lindland et al. 1994). This means that all statements in the artifact should be according to the syntax as defined by the UML Specification it claims to conform to. The corresponding violations of the syntax will lead to a syntax error caused either by morphological errors or due to syntactic incompleteness. Morphological errors can occur, for example, if UML constructs overlap or if elements from different versions of UML are included in an artifact or if extension mechanisms (stereotypes, tagged values, or constraints) are incorrectly used or if there is a datatype mismatch. Syntactic incompleteness can occur, for example, if one of the nodes at the end of an association is non-existent or if the datatype value in an attribute of a UML Class Diagram is missing. The origin of many of these errors is that UML visual syntax is weak with respect to syntactic disjointness and syntactic differentiability (Morris, Spanoudakis 2001).

External Attributes. The UML-stakeholder suitability discussed above is obviously a relevant attribute. It is easier to introduce and detect errors in a complex diagram with many constructs; hence simplicity is a relevant attribute. For example, it is useful to split large Use Cases into multiple sub-Use Cases. With respect to the syntax, interoperability and standardization are somewhat related. Although all syntax must conform to the standard UML definition, it is possible to produce a syntactically correct UML artifact that is not interoperable with any of the UML modelers and therefore would not be of much use from the viewpoint of a consumer.

Internal Attributes. The UML primary notation and format are relevant internal attributes. With respect to the UML primary notation, there must be a match between any construct used and the version of UML to which the artifact claims conformance. The format itself has no direct bearing on correct syntax, but when errors occur, then some formats may be easier to deal with than others.

Mechanisms. There are three syntactic mechanisms: error prevention, error detection (and location), and error correction. Error prevention is primarily useful for morphological errors and detection is useful for syntactic incompleteness. Any mechanism deployed must be feasible. There is no substitute for proper training in UML for error prevention. Static or dynamic verification techniques could be used for syntax checking. Pair modeling can help in error prevention, while inspections can help in error detection and subsequent correction in UML artifacts. Error prevention and detection is readily possible with the use of a UML modeler that provides dynamic checking of syntax. (This would not be possible if the model was being crafted using an ordinary illustration or drawing program.) Automatic error correction with a modeler is possible for only trivial cases such as when the modeler knows that there must be a unique association missing between only two given constructs (say, an Actor and a Use Case). It should be noted that direct manual authoring of markup is verbose and prone to error. However, if the underlying format of a UML artifact is serialized in XML, error correction (with a regular expressions-based pattern matching script running under a shell environment along with human input) on the command line is possible in most cases. This can be particularly convenient and efficient compared to modeler-based correction if identical errors in several models need to be corrected. In some cases, such as spelling or grammatical error in a UML Note construct, human intervention may be the only option as modelers do not normally process purely natural language text. A simple indirect measure such as the syntax error ratio or error cost (Cost per Number of Errors Detected) can help determine the effort required for error correction.

3.2. Semantic Quality

Semantic quality is a measure of how well an artifact accommodates its contract with a single variable entity - the domain that it represents. The

goals, criteria, and mechanisms that we now describe are driven by this dynamic external view (Table 1(b)).

Table 1(b): Detailed Outline of the UML Quality Framework: Semantics.

Goals	Criteria		Mechanisms	
Validity and Completeness	**External Attributes** ▪ Domain-UML Suitability ▪ Clarity o Semantic Clarity ▪ Consistency o Visual Coherence ▪ Familiarity	**Internal Attributes** ▪ Primary Notation ▪ Secondary Notation ▪ Size ▪ Structure	**Quality Assurance** ▪ Training in Use of UML ▪ Training in Use of Secondary Notation ▪ Metadata o Annotation ▪ Pair Modeling **Quality Evaluation** ▪ Checklist ▪ Inspection ▪ Measurement o Metrics	Tool Support
Feasibility Analysis				

Goals. There are two complementary goals for semantic quality of a UML artifact: semantic validity and semantic completeness (Lindland et al. 1994). Validity means that all statements made by the artifact are correct and relevant to the domain under study; an artifact containing an extraneous statement will result in a non-validity. Completeness implies that the artifact contains all the statements about the domain that are correct and relevant; an artifact that contains an incomplete statement according to the domain will result in non-completeness. Validity indicates necessity while completeness indicates sufficiency with respect to the domain. Both the goals are open-ended and can be achieved in their entirety in only but trivial cases. A main source of non-validity is attempts to retrofit UML constructs to concepts in the domain for which they are not suitable. As discussed in Section 2.1.2, it is difficult to be semantically complete in UML for problem domains due to its limited expressive power. Therefore, a feasibility constraint (Lindland et al. 1994) must be applied to the goals for them to be realistic.

External Attributes. The domain-UML suitability discussed above is obviously a relevant attribute. Semantic clarity is necessary in the use any combination of constructs and in the text being used. For example, the text included in a UML Note construct should make sense with respect to the domain. Consistency is necessary for both validity and completeness. Familiarity from past projects can help avoid similar errors.

Internal Attributes. The UML primary and secondary notation, artifact size, and structure are the relevant internal attributes. With respect to primary notation, for example, a UML Class Diagram representing a desk having an attribute `legs` with a negative value would be a semantic error. An appropriate choice of diagram types for representing structural and behavioral aspects along with their suitable level of abstraction and refinement is necessary. Colors if used can indicate semantic similarities and differences between concepts, and labels must attempt to capture domain terminology. Size is relevant as, for example, in a hierarchy of UML Class Diagrams, the diagram with the largest dimensions could signify the main class. Aspects of (in)dependency of objects, message passing, and so forth, are often captured in relationships, giving an artifact a structure that can help identify certain semantic errors. For example, a UML Sequence Diagram that consists of only requests and no responses would result in a semantic error with respect to most domains.

Mechanisms. Training in both the use of UML primary notation and the secondary notation is required. At any given time, a UML artifact must avoid domain co-occurrrence, that is, try to represent only one of problem or solution domain. Pair modeling can help prevent such semantic errors and inspection can help correct them. Proper use of domain-specific annotations can encapsulate intent and can also help highlight semantic errors. There are modelers that can perform consistency checking, albeit only at a basic level as they tend to be domain-neutral.

3.3. Pragmatic Quality

Pragmatic quality is a measure of how well an artifact accommodates its contract with respect to multiple variable entities - the stakeholders. The goals, criteria, and mechanisms that we now describe are driven by this dynamic external view (Table 1(c)).

Table 1(c): Detailed Outline of the UML Quality Framework: Pragmatics.

Goals	Criteria		Mechanisms	
Comprehension	**External Attributes** ▪ Domain-UML-Stakeholder Suitability ▪ User Preference ▪ Clarity o Semantic Clarity o Visual Clarity ▪ Consistency o Visual Coherence ▪ Simplicity ▪ Familiarity ▪ Interoperability ▪ Standardization	**Internal Attributes** ▪ Secondary Notation ▪ Size o External Reuse ▪ Structure o Internal Reuse o Coupling ▪ Format	**Quality Assurance** ▪ Training in Use of UML ▪ Training in Use of Secondary Notation ▪ Metadata o Annotation ▪ Pair Modeling ▪ Refactoring **Quality Evaluation** ▪ Checklist ▪ Inspection ▪ Measurement o Metrics	Tool Support
Feasibility Analysis				

Goals. There is only one goal for pragmatic quality of a UML artifact: comprehension by stakeholders (Lindland et al. 1994). Despite the claims (Dong and Yang 2003), knowledge represented using UML is neither automatically nor directly accessible to humans by the mere fact that the artifacts are graphical in nature (Gurr and Tourlas 2000). The goal of comprehension is open-ended. For a non-trivial artifact, it is not realistic that each stakeholder will be able to comprehend each statement made by the artifact in its entirety at all times. This motivates the adoption of *feasible comprehension* (Lindland et al. 1994). A comprehension is feasible if the cost of reducing the misunderstood statements in an artifact does not exceed the drawbacks of retaining them. We also note here that comprehension is (ISO 2001) a necessary prerequisite for quality factors like learnability, maintainability, satisfiability, and usability.

External Attributes. The triad of domain-UML-stakeholder suitability is a relevant attribute. Users may prefer one diagram type to the other for the sake of understanding. The significance of clarity, consistency, and simplicity towards understanding are well-known in cognitive psychology. Familiarity can improve comprehension by reusing knowledge. For example, a stakeholder in a role of non-technical participant will be more comfortable with names such as `RegisteredUser` rather than design or implementation-specific `ID-123`. Interoperability is necessary as same artifacts could be processed and viewed by different tools. Standardization reduces unpredictability on part of stakeholders.

Internal Attributes. The UML secondary notation, size, structure and the format being used are the relevant internal attributes. Much of the discussion on secondary notation in Section 2.2 applies and, along with size and format, is important for clarity and simplicity. The external reuse of the constructs has a direct impact on familiarity for a stakeholder that may have already interacted with these constructs in the past. High coupling usually leads to a complex structure that in turn can have negative impact on comprehension.

Mechanisms. Training in both the use of UML primary notation and the secondary notation is required for the producers to make proper use of the latter. Non-technical stakeholders and consumers would need training in the UML primary notation but could be helped further with proper annotations for unfamiliar terminology. The style guidelines of (Ambler 2003) and patterns of (Evitts 2000) can serve as a basis for a checklist against which an informal pragmatic quality evaluation can be carried out. Pair modeling and inspection (where feasible) can tackle most internal attributes. Refactoring aims to improve structure with the goal of increasing comprehension but obviously applies only to the existing base of artifacts. Some UML modelers also have internally implemented their own set of style guidelines and refactorings that can be applied (semi)automatically. The object-oriented metrics for coupling of (Chidamber and Kemerer 1994; Harrison et al. 1998) have been implemented in some modelers.

3.3.1. Example

Figure 2 illustrates two artifacts with UML Class Diagrams where one is a pragmatic quality improvement over the other. There are several pragmatic quality issues in the artifact in Figure 2(a). There is no rationale for differences in node sizes or their positioning. The color variations are unjustified. Node 1 label has low visibility on a low-to-medium resolution computer monitor and on a black and white printer. The relationships are unclear (whether nodes 1 and 2, or nodes 3 and 4 are related). It is inconvenient to place text labels on vertices (1,4) and (3,2) as they are non-horizontal.

Figure 2(b) can be viewed as a result of a sequence of simple refactoring methods (Consistent Node Size, Consistent Color, and Use Grid Architecture) of 1(a). The background of nodes in (b) has been set to a

uniform color different from its surroundings (white on paper) but at a very different value from that of the labels.

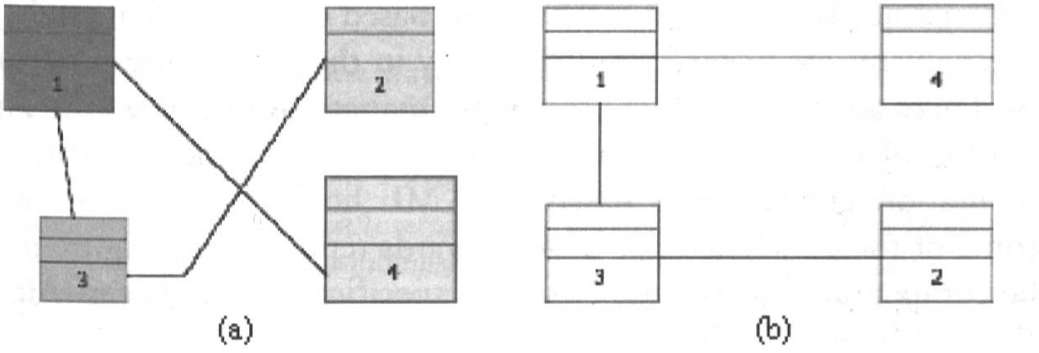

Figure 2: The pragmatic quality issues of the artifact in (a) are ameliorated in (b).

4. Conclusion

UML artifacts are organizational assets in software process environments that embrace them and must strive for high quality to be useful for their target stakeholders. In conclusion, we make the following recommendations to an organization that values the production and long-term viability of UML artifacts.

For any software project, domain engineering is a must, and it helps decide the suitability (if at all) and the extent of UML use: it is always the underlying domain that should drive the deployment of any modeling language and UML is not a silver bullet. Unfortunately, this aspect is often ignored (Offen 2002) and a rush to production leads to a myriad of issues later in the process.

If the UML artifacts are central to an organization, they must be treated accordingly. When they are significant in number, their production process should be systematically planned. This is possible by setting up a sub-process within the process of the target product and is no different from other artifacts such as software process documentation. As a prerequisite, the producers of UML artifacts must be trained in both the fundamentals of UML and its secondary notation.

It is worthwhile investing in a UML modeler that provides facilities that help carrying out the mechanisms for targeting semiotic quality. For the sake of longevity, it is crucial that the life of a UML artifact is not tied to that of a modeler.

Since criteria are not all equal in all situations and there are no perfect mechanisms, a feasibility analysis prior to any decisions is necessary. One possible approach in this case is to prioritize the criteria based on organizational needs and adopt mechanisms based on that. Fortunately, not all mechanisms are needed simultaneously in the production process.

There are two problem areas that emanate from this work. The integration of UML artifacts in a larger context of a generic software documentation environment based on XML holds promise. For example, some of the deficiencies of UML towards representing requirements can be compensated by the use of formal specification languages (Alagar and Periyasamy, 1998). This would give rise to heterogeneous documents that consist of diagrams, mathematical expressions, tables, and so forth, in descriptive markup. The issue of the quality of resulting markup itself then eventually arises. The limitations of UML such as towards conceptual data modeling or reactive systems have led to the introduction of other visual languages such as Object Role Modeling (ORM) (Halpin 1998) and Use Case Maps (UCM) (Buhr 1998) for modeling software. Addressing quality of artifacts in these languages would be of interest. We anticipate these as future research directions.

Acknowledgements

The author would like to thank Olga Ormandjieva and Hsueh-Ieng Pai (Concordia University, Montreal, Canada) for early impetus, and the reviewers for their detailed comments and helpful suggestions on this paper.

References

ALAGAR, V.S. & PERIYASAMY, K. (1998). Specification of Software Systems. New York: Springer-Verlag.

ALEXANDER, C. (1979). The Timeless Way of Building. Oxford: Oxford University Press.

AMBLER, S.W. (2003). The Elements of UML Style. New York: Cambridge University Press.

ARLOW, J.; EMMERICH, W. & QUINN, J. (1998) Literate Modelling - Capturing Business Knowledge with the UML. First International Conference on the Unified Modeling Language (<<UML>> 1998). Mulhouse, France. London: Springer Verlag.

BECK, K. (1999). Extreme Programming Explained: Embrace Change. Uppler Saddle River, USA: Addison-Wesley.

BOEHM, B.W et al. (2001). Software Cost Estimation with COCOMO II. Uppler Saddle River, USA: Prentice Hall.

BOOCH, G.; JACOBSON, I. & RUMBAUGH, J. (2005). The Unified Modeling Language Reference Manual. Second Edition. Uppler Saddle River, USA: Addison-Wesley.

BOTASCHANJAN, J.; PISTER, M. & RUMPE, B. (2004). Testing Agile Requirements Models. *Journal of Zhejiang University Science* 5/5: 587-593.

BRAY, T. et al. (2004). Extensible Markup Language (XML) 1.0 (Third Edition). W3C Recommendation. Cambridge, USA: World Wide Web Consortium.

BRITTON, C.; KUTAR, M.; ANTHONY, S. & BYARKER, T. (2002). An Empirical Study of User Preference and Performance with UML Diagrams. Proceedings of Symposia on Human Centric Computing Languages and Environments. Arlington, USA. Washington: IEEE Computer Society.

BRODSKY, S. & GROSE, T. (2002). Mastering XMI: Java Programming with the XMI Toolkit, XML and UML. Hoboken: John Wiley & Sons.

BUHR, R.J.A. (1998). Use Case Maps as Architectural Entities for Complex Systems. IEEE Transactions on Software Engineering 24/12. Washington: IEEE Computer Society: 1131-1155.

CARLSON, D. (2001). Modeling XML Applications with UML. Uppler Saddle River, USA: Addison-Wesley.

CHIDAMBER, S. R. & KEMERER, C. F. (1994). A Metrics Suite for Object Oriented Design. IEEE Transactions on Software Engineering 20/6. Washington: IEEE Computer Society: 476-493.

COAD, P.; LEFEBVRE, E. & DELUCA, J. (1999). Java Modeling in Color with UML: Enterprise Components and Process. Uppler Saddle River, USA: Prentice Hall.

COMBER, T. (1993). The Importance of Text Width and White Space for Online Documentation. Honours Thesis. School of Multimedia and Information Technology. Southern Cross University, New South Wales, Australia.

CONALLEN, J. (2003). Building Web Applications with UML. Second Edition. Uppler Saddle River, USA: Addison-Wesley.

CONRADI, R. et al. (2003). Inspection of UML Diagrams using OORT - An Industrial Experiment. European Conference for Object-Oriented Programming (ECOOP 2003). Darmstadt. New York: Springer-Verlag.

DA SILVA, P.P. & PATON, N. (2003). Improving UML Support for User Interface Design: A Metric Assessment of UMLi. ICSE 2003 Workshop on Bridging the Gaps between

Software Engineering and Human-Computer Interaction. Portland, USA.

Di Battista, G.; Eades, P.; Tamassia, R. & Tollis, I.G. (1999). Graph Drawing: Algorithms for the Visualization of Graphs. Uppler Saddle River, USA: Prentice-Hall.

Dong, J. & Yang, S. (2003). Extending UML to Visualize Design Patterns in Class Diagrams. Fifteenth International Conference on Software Engineering and Knowledge Engineering (SEKE 2003). San Francisco. New York: ACM Press.

Douglass, B.P. (2004). Real Time UML: Advances in the UML for Real-Time Systems. Uppler Saddle River, USA: Addison-Wesley.

Ebenau, R.G. & Strauss, S.H. (1994). Software Inspection Process. New York: McGraw-Hill.

Eichelberger, H. (2003). Nice Class Diagrams Admit Good Design? Proceedings of the 2003 ACM Symposium on Software Visualization. San Diego. New York: ACM Press: 159-216.

Eppler, M.J. (2001). The Concept of Information Quality: An Interdisciplinary Evaluation of Recent Information Quality Frameworks. *Studies in Communication Sciences* 1/2: 167-182.

Eriksson, H.-E. & Penker, M. (2000). Business Modeling with UML: Business Patterns at Work. Hoboken, USA: John Wiley & Sons.

Evitts, P. (2000). A UML Pattern Language.Town: Macmillan Technical Publishing.

Fenton, N.E. & Pfleeger, S.L. (1997). Software Metrics: A Rigorous & Practical Approach. Toronto: International Thomson Computer Press.

Ferraiolo, J.; Fujisawa, J. & Jackson, D. (2003). Scalable Vector Graphics (SVG) 1.1 Specification. W3C Recommendation. Cambridge, USA: World Wide Web Consortium.

Fowler , M. et al. (1999). Refactoring: Improving the Design of Existing Code. Upper Saddle River, USA: Addison-Wesley.

Fowler, M. et al. (2003). UML Distilled: A Brief Guide to the Standard Object Modeling Language. 3rd Edition. Upper Saddle River, USA: Addison Wesley.

Genero, M.; Miranda, D. & Piattini, M. (2002). Defining and Validating Metrics for UML Statechart Diagrams. Sixth ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE 2002). Malaga. New York: Springer-Verlag

Gilb, T. (1988). Principles of Software Engineering Management. Upper Saddle River, USA: Addison-Wesley.

Glinz, M. (2000). Problems and Deficiencies of UML as a Requirements Specification Language. Proceedings of the 10th International Workshop on Software Specification and Design (IWSSD-10). San Diego. Washington: IEEE Computer Society.

Green, P. & Edwards, M. (2004). Enhancing UML to Support the Specification of Behavior for Embedded Systems-on-a-Chip. Presentation at the UML-SoC Workshop 2004. Theme: UML for SoC Design. San Diego. Washington: IEEE.

Gurr, C. (1999). Effective Diagrammatic Communication: Syntactic, Semantic and Pragmatic Issues. *Journal of Visual Languages and Computing.* 10/4. Cambridge, UK: Academic Press: 317-342.

Gurr, C. & Tourlas, K. (2000). Towards the Principled Design of Software Engineering Diagrams. Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000). Limerick. Washington: IEEE Computer Society: 509-518.

HALPIN, T.A. (1998). A Comparison of UML and ORM for Data Modeling. Proceedings of 3rd IFIP WG8.I International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design (EMMSAD'98). Pisa. Hershey, USA: IDEA Group.

HARRISON, R.; COUNSELL, S. J. & NITHI, R. V. (1998). An Evaluation of the MOOD Set of Object-Oriented Software Metrics. IEEE Transactions on Software Engineering 24/6. Washington: IEEE Computer Society.

ISO. (1994). ISO 8402:1994. Quality Management and Quality Assurance — Vocabulary. Geneva: ISO.

ISO. (2001). ISO/IEC 9126-1:2001. Software Engineering — Product Quality — Part 1: Quality Model. Geneva: ISO.

JACOBSON, R.E. & WURMAN, R.S. (1999). Information Design. Cambridge, USA: MIT Press.

KIM, H. & BOLDYREFF, C. (2002). Developing Software Metrics Applicable to UML Models. Sixth ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE 2002). Malaga. New York: Springer-Verlag.

KIRAKOWSKI, J. (1993). Software Usability Measurement Inventory (SUMI). Human Factors Research Group, University College Cork, Cork.

KLEPPE, A.; WARMER, J. & BAST, W. (2003). MDA Explained, The Model Driven Architecture: Practice and Promise. Upper Saddle River, USA: Addison-Wesley.

KROGSTIE, J. (1998). Integrating the Understanding of Quality in Requirements Specification and Conceptual Modeling. ACM SIGSOFT Software Engineering Notes 23/1, New York: ACM Press: 86-91.

KROGSTIE, J. (2001). Using a Semiotic Framework to Evaluate UML for the Development of Models of High Quality. In: SIAU, K. & HALPIN, T. (eds). Unified Modeling Language: Systems Analysis, Design and Development Issues. Hershey, USA: IDEA Group.

KRUCHTEN, P. (2004). The Rational Unified Process: An Introduction. Third Edition. Upper Saddle River, USA: Addison-Wesley.

LARMAN, C. (2002). Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process. 2nd Edition. Upper Saddle River, USA: Prentice Hall.

LETHBRIDGE, T.C. & LAGANIÈRE, R. (2001). Object-Oriented Software Engineering: Practical Software Development using UML and Java. New York: McGraw-Hill.

LINDLAND, O.I.; SINDRE, G. & SØLVBERG, A. (1994). Understanding Quality in Conceptual Modeling. IEEE Software 11/2: 42-49. Washington: IEEE Computer Society: 42-49

MELLOR, S.J. & BALCER, M.J. (2002). Executable UML: A Foundation for Model-Driven Architecture. Upper Saddle River, USA: Addison-Wesley.

MILLER, G.A. (1956). The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information. *The Psychological Review* 63. Washington: American Psychological Association: 81-97.

MOODY, D.L. & SHANKS, G.G. (2003). Improving the Quality of Data Models: Empirical Validation of a Quality Management Framework. *Information Systems* 28/6. Amsterdam: Elsevier: 619-650.

MORRIS, C.W. (1938/1970). Foundations of the Theory of Signs. International Encyclopedia of Unified Science 1/2. Chicago: Chicago University Press.

MORRIS, S. & SPANOUDAKIS, G. (2001). UML: An Evaluation of the Visual Syntax of the Language. Proceedings of the 34th Annual Hawaii International Conference on System Sciences (HICSS-34).Washington: IEEE

OFFEN, R. (2002). Domain Understanding is the Key to Successful System Development. Requirements Engineering 7/3. New York: Springer-Verlag: 172-175.

PAULK, M.C. et al. (1993). Capability Maturity Model, Version 1.1. *IEEE Software* 10/4. Washington: IEEE Computer Society: 18-27.

PETRE, M. (1995). Why Looking Isn't Always Seeing: Readership Skills and Graphical Programming. *Communications of the ACM* 38/6. New York: ACM Press: 33-44.

PREECE, J.; ROGERS, Y. & SHARP, H. (2002). Interaction Design: Beyond Human-Computer Interaction. Hoboken, USA: John Wiley & Sons.

PURCHASE, H.C. et al. (2001). Graph Drawing Aesthetics and the Comprehension of UML Class Diagrams: An Empirical Study. Australian Symposium on Information Visualisation, Sydney 9. New York: ACM Press: 129-137.

REINGRUBER, M. & GREGORY, W. (1994). The Data Modeling Handbook: A Best-Practice Approach to Building Quality Data Models. Hoboken, USA: John Wiley & Sons.

RIGDEN, C. (1999). The Eye of the Beholder: Designing for Colour-Blind Users. *British Telecommunications Engineering Journal* 17. Hoboken, USA: John Wiley & Sons.

ROBBINS, J. (1999). Assessing UML and Usability. Presentation at Bay Area Roundtable (BART). Irvine Research Unit in Software. Information and Computer Science. University of California, Irvine, USA.

ROSENBERG, D & SCOTT, K. (1999). Use Case Driven Object Modeling with UML: A Practical Approach. Upper Saddle River, USA: Addison-Wesley.

RUFAI, R. (2003). New Structural Similarity Metrics for UML Models. Masters Thesis. King Fahd University of Petroleum & Minerals. Dhahran.

SHANKS, G. (1999). Semiotic Approach to Understanding Representation in Information Systems. Proceedings of the Information Systems Foundations Workshop. Department of Computing, Macquarie University, Australia.

SHANKS , G; TANSLEY, E. & WEBER R. (2003). Using Ontology to Validate Conceptual Models. *Communications of the ACM* 46/10. New York: ACM Press: 85-89.

SHNEIDERMAN, B. & PLAISANT, C. (2005). Designing the User Interface: Strategies for Effective Human-Computer-Interaction. Upper Saddle River, USA: Addison-Wesley.

SCHNEIDEWIND, N.F. & FENTON, N.E. (1996). Do Standards Improve Product Quality? *IEEE Software* 13/1. Washington: IEEE Computer Society: 22-24

SUNYE, G. et al. (2001). Refactoring UML Models. Fourth International Conference on the Unified Modeling Language (UML 2001). Toronto. New York: Springer-Verlag.

TRONG, T.T.D. (2003). A Systematic Procedure for Testing UML Designs. 14th International Symposium on Software Reliability Engineering (ISSRE 2003). Denver. Washington: IEEE Computer Society.

VAN SOLINGEN, R. & BERGHOUT, E. (1999). The Goal/Question/Metric Method: A Practical Method for Quality Improvement of Software Development. New York: McGraw-Hill.

WARMER, J. & KLEPPE, A. (2003). The Object Constraint Language: Precise Modeling With UML. Second Edition. Upper Saddle River, USA: Addison-Wesley.

WILLIAMS, L. & ERDOGMUS, H. (2002). On the Economic Feasibility of Pair Programming. Fourth Workshop on Economics-Driven Software Engineering Research (EDSER-4). Orlando, FL. New York: ACM Press.