Software Maintainability Improvement: Integrating Standards and Models

William C. Chu,
Dpt. of Computer Science and Information Engineering, Tunghai University, Taiwan

Chih-Wei Lu, Chih-Hung Chang, and Yeh-Ching Chung
Dpt. of Information Engineering, Feng Chia University, Taiwan

Yueh-Min Huang
Department of Engineering Science, Cheng Kung University, Taiwan

Abstract

Software standards are highly recommended because they promise faster and more efficient ways for software development with proven techniques and standard notations. Designers who adopt standards like UML and design patterns to construct models and designs in the processes of development suffer from a lack of communication and integration of various models and designs. Also, the problem of implicit inconsistency caused by making changes to components of the models and designs will significantly increase the cost and error for the process of maintenance. In this paper, an XML-based unified model is proposed to solve the problems and to improve both software development and maintenance through unification and integration.

Keywords: software standards, software maintenance, UML, XML, model unification and integration.

1. Introduction

By involving deeper and deeper to the operations of modern businesses, software systems have been a dominant influence of successful businesses. As the growth of scales and contents of the software systems, most of those software systems have become too complex to be developed by individual efforts. Responding to that situation, the typical process of the software lifecycle, as shown in figure 1, and each of the phases should be proceeded by various working groups with different methodologies. As a result, software development usually involves teamwork and need good communication. However, without the restrictive enforcement of using common standards, most systems are developed in an ad hoc manner which makes software development difficult and costly.

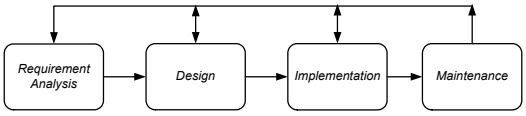

Figure 1. The typical process of the software life cycle

On the other hand, software systems should not only be more flexible and efficient in the process of development, but they also need to be more effective in the process of maintenance. Software maintenance is defined as the modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a changed environment. In the early days of computing (1950s and early 1960s), software maintenance took up only a small part of the software life cycle. In the late 1960s and the 1970s, maintaining those old working software – called legacy systems – started to be recognized as a major activity of the software life cycle [1]. Up until now, the maintenance cost of these working systems has turned to be much higher compared to that of the initial development [3], and the cost of maintenance keeps growing faster since new software gets more and more complicated.

To deal with the demand of effective development, *software standards* are highly recommended because they promise faster and more efficient ways for software development with proven techniques and standard notations. De facto standards, such as *Unified Modeling Language* (UML) [10] or *eXtensible Markup Modeling Language* (XML) [6], are used to reduce the overhead of software inner-communication during the software life cycle and to increase maintainability and

reusability. Another de facto standard, *design patterns* [8] are reusable solutions to recurring problems that occur during software development [2, 5]. From the perspective of improving software development, modern software standards do show their contributions. However, the way that woks out a problem brings up several new problems.

For the first problem, these software standards usually only cover single or partial phases of the software process. For instance, UML provides standard notations for modeling software analysis and design, yet it lacks support in the implementation and maintenance phases. Another example is found in design patterns, which offer help only to the design phase. A third example is component-based technologies, which focus on the implementation phase for the most part. And so on.

For the second problem, the software process consists of the whole software life cycle including requirement, design, implementation, testing, and maintenance phases. The inner-phase consistence promised by standards in their respective phases exhibit the serious inter-phase consistence problems, since currently these standards are proposed by individual organizations and they do not "talking" well to each other. Designers need to spend a lot of manual effort to map and integrate standards in previous phases of the software life cycle to catch the designs in order to proceed the works in following phases.

For the third problem, the similar dilemma of the inconsistency will obsess the maintainers. For the correctness and consistency of the software system after maintenance/modification, two fundamental questions should be answered first. The first, where can we find the right points quickly for modification? And the second, how can we find out the ripple effect (impact analysis) and then sustain the consistency? Since the modeling specifications are isolated with various standards, and the codes with some specified language are in the lower level representation that can be hardly related to the higher level representations like design models, jobs of maintenance are error-prone, inefficient, and costly. Furthermore, without unifying and integrating these standards, the consistency of the models cannot be held, and the extent of automation is very narrow.

In this paper, we proposes an XML-based meta-model to unify and integrate these well-accepted standards in order to improve maintainability of the software systems. This paper will discuss the adopted standards, including UML, design patterns, component-based frameworks, and XML. A comparison and mapping of these standards will be presented. An XML-based unified model is proposed to unify and integrate models that are composed with various standards.

The rest of the paper is organized as follows. The related works are briefed in section 2; section 3 introduced the approach to improve software maintainability by unifying and integrating existing software standards; a conclusion will be given in section 4 lastly.

2. The Related Works

Software standards are introduced to improve software development. By using the standards notations and concrete designs provided from widely accepted standards, designers can successfully reduce the complexity of software development. However, software standards caused the problem of inconsistency of the different modeling specifications, and that leads to the difficulty of maintenance. In this section, related methodologies, software standards and studies are surveyed to disclose the problem itself, as well as some noteworthy efforts responding to that demand.

2.1. Object-Oriented Technology and UML

Object-oriented (OO) technology is a landmark of software engineering; it organizes data as objects in ways that "echo" how things appear, behave, and interact with each other in the real world. An object is identified by its individual characteristics and activities, and it plays a role as a reusable, self-operational component in a business information model. OO technologies greatly influence software development and maintenance through faster development, cost saving, and quality improvement [11]. Object-Oriented Analysis and Design (OOA/D) [3] follows the concept of OO technology and thus has become a major trend for methods of modern software development and system modeling. A sign of the maturity of OOA/D is the convergence of object-oriented modeling notations in the form of the Unified Modeling Language (UML) [10].

UML is used to specify, visualize, construct and document the artifacts of software systems. UML defines the following diagrams to build software models and to express important domain-related concepts: *use case diagrams, class diagrams, collaboration diagram*s, *component diagrams*, etc. UML allows the user to easily understand a system analysis or design through these diagrams as well as its widely accepted modeling notations. UML is rapidly

growing to be the first choice of standards for object-oriented modeling in general. However, the lack of formality in UML prevents the evaluation of completeness, consistency, and content in requirements and design specifications [4]. Not only UML, but also all the modeling techniques used in a design need more formalization to achieve system comprehension and integration in software development and maintenance.

2.2. Modeling Transfer and Verification

Current modeling techniques and standards offer explicit definitions and notations to support software development, but few of them have the capability to enable users to verify the completeness and consistency of their work while users shift to other techniques or standards that are needed in the next phases of software development. This leads to limited automation and inefficiency. Some researchers have dedicated their work to improve the situation. In the followings, we will consider three issues related to modeling transfer and verification: modeling understanding, automation, and modeling verification.

Modeling understanding is a technique that helps an engineer compare artifacts by summarizing where one artifact (such as a design) is consistent with and inconsistent with another artifact (such as source) [9]. Other works have developed a software reflection model technique to help engineers perform various software engineering tasks by exploiting – rather than removing – the connection between design and implementation [9]. Based on a similar concept, an engineer might use a reverse engineering system to derive a high-level model from the source code [12].

Although the studies of these three issues try to address the isolated problem of modeling information, mostly they take care of the problems of models in some specified subjects or limited domains. Software models are dynamically changed during the analysis/design, revision, and maintenance phases of the software life cycle. Software tools at each phase usually employ their own formats to describe the software model information. As we can see in the surveys and discussions in the previous sections, the various standards do show their respective contributions in their specialized subjects for software development. Unfortunately, none of these standards is general enough to cover all phases of the software life cycle, thus developers need to adopt more than one standard to accomplish their work. However, because most of the standards offer no connections or compatibility to the others, gaps exist between these standards' applications. Figure 2 illustrates the relationship of the standards along with their positions during the life cycle of software development. The notation ⊠ expresses that there is a need for modeling transfer between successive phases/models in a specific standard; the notation ⊗ points out the absence of consistency from one standard to the others.

New standards will surely keep emerging for new requirements of software engineering. It is clear that modeling information expressed with a specific standard can only show part of the system information from its particular aspect. In this paper, we would rather propose a unified system model to integrate and coordinate various models in different standards with different phases of the software life cycle.

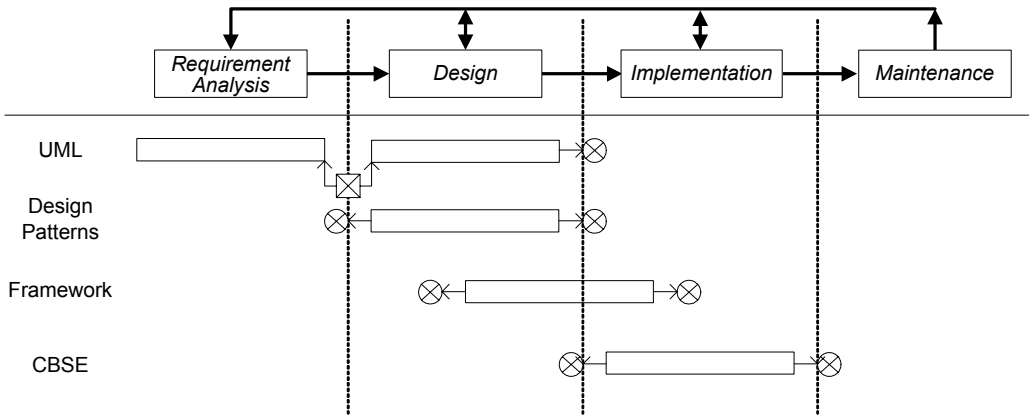

Figure 2. The relationship of some standards and the life cycle of software development

2.3. eXtensible Markup Modeling Language (XML)

XML [7] is a standard language supported by W3C (World Wide Web Consortium) with

many useful features such as application neutrality (vender independence), user extensibility, ability to represent arbitrary and complex information, validation for data structure scheme, and human readability. XML provides the feasibility of the unification and formalization to different levels of concepts and representations of a system.

XML schema is a language which defines structure and constraints of the contents of XML documents. An XML schema consists of a set of type definitions and element declarations. These can be used to assess the validity of well-formed elements and attribute information items, and furthermore may specify augmentations to those items and their descendants.

3. The Approach to Unifying and Integrating Standards

In this paper, the *XML-based Unified Meta-Model* (*XUMM*) is used to define the schema of an *XML-based Unified Model* (*XUM*) – the integration and unification of the modeling information from the adopted standard models, such as analysis and design models represented in UML, design patterns, framework, etc., in each phase of the software life cycle. To avoid confusion with the various uses of the term "model", we refer in this paper to those models composed with a standard as "*submodels*" of our integrated, unified model. We call them submodels because each one characterizes the system partially, in the aspect of a specific phase. Through the transformation of XUMM, a submodel can be transformed into its corresponding XML representation, which we call a "*view*" of the XUM.

As shown in Figure 3, based on XUMM, submodels are unified, integrated, and represented as views of an XUM. Semantics in each submodel should be described explicitly and transferred precisely in XUM.

In our approach, an XUM is employed to facilitate the following tasks:

1) The capturing of modeling information of models and transforming into views of XUM.
2) The two-way mapping of modeling information among models and XUM views.
3) The integration and unification of modeling information of different views in XUM.
4) The support of systematic manipulation.
5) Assisting the consistency checking of views represented in XUM.
6) The reflection of changes of view information in XUM to models in each phase.

The details of XUMM as well as XUM will be discussed in the following sections.

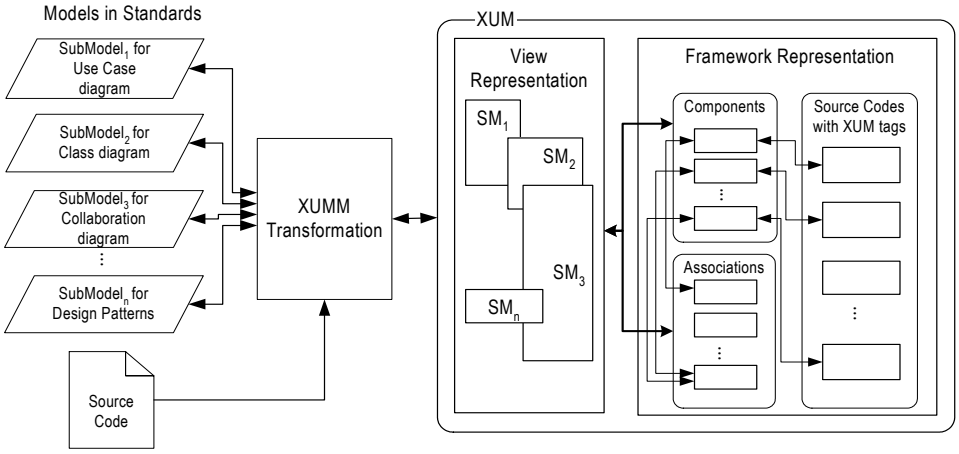

Figure 3. The unification and integration of models into XUM

3.1. XML-based Unified Meta-Model (XUMM)

Figure 4 shows the relationship of views in XUM. The major merits of XUM are (1) the modeling information used in models (views) of each phase of the software life cycle and (2) the interaction and relationship of models (views). Both are explicitly defined and represented in XUM.

The relationship of the XUMM with an XUM is like the DTD with an XML document. XUMM defines the schema (definitions) of an XUM. Three kinds of *elements* defined in XUMM are used to describe the constitution of an XUM; they are *component, association,* and *unification relation*. Any object in an XUM is

identified as a component. Components and associations are used to describe the semantic information of model objects and their relationships respectively. The third kind of element, unification relation, is used to describe the relationship of different views.

According to the three kinds of elements, three primitive schemas are defined in XUMM respectively – *ComponentType*, *AssociationType*, and *Unification_linkType*. The ComponentType schema defines the necessary modeling semantic information and the types that are used to describe components in our unified model.

The AssociationType schema defines the necessary information and the types that are used to describe the relationships of components.

In order to show the relationship of the integration and unification of views in XUM, *Unification_linkType* is defined. Unification_linkType schema defines the hyperlink relations between elements in an XUM using a set of xlinks.

Based on the purposes of Unification_linkType, three types of links are defined further – *Integration_link, Abstraction_link and Sourcecode_link*. The *Integration_link* is used to link a set of components and/or associations that have the same semantics but may be named or represented differently in different views. The *Abstraction_link* is used to link a component/ association to a view. The view consists of a set of components and their associations; it also represents the details of a specific component at a lower level of abstraction. And the *Sourcecode_link* is used to link a component to its corresponding source code.

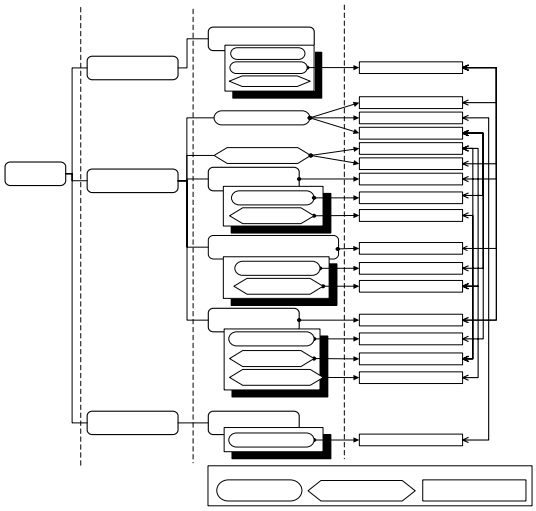

Figure 4. The relationship among views in XUM

Table 1. Mapping of model elements and XUM elements

Models /Standards	***Model elements***	***XUM element Representations***
UML Use Case diagram	Actor	<Actor>
	Use Case	<Usecase>
	Association	<Relationship type="association">
	Generalization	<Relationship type="generalization">
	Extend	<Relationship type="extend">
	Include	<Relationship type="include">
UML Class, Collaboration, Sequence diagram	Class	<Class>
	Attribute	<Attribute>
	Operation	<Operation>
	Interface	<Interface>
	Parameter	<Parameter>
	Association	<Class_Association type="association">
	Composition	<Class_Association type="composition">
	Generalization	<Class_Association type="generalization">
	Dependency	<Class_Association type="dependency">
	Message	<Message>
Design Patterns	Participants	<Participants>
	Structure	<Structure>
	Collaborations	<Collaboration>

Based on the integration, abstraction and sourcecode links, the submodels – adopting various standards that might share some semantics but were not explicitly represented – can be integrated and unified in XUM. Therefore, when a submodel (view) gets changed, the changes can be reflected to other related submodels (views).

Each submodel has its corresponding XUM representation, the view, and its schema is defined in XUMM. Following the transformation of XUMM, transforming modeling information of a submodel into a view of the XUM is not a difficult task. Due to the space limitation, we only show the its mapping rules in Table 1.

3.2. XML-based Unified Model (XUM)

An XML-based Unified model (XUM) is the representation of artifacts of software systems defined in XUMM. These artifacts are the modeling information collected from models of standards used in each phase of the software life cycle. Firstly, each submodel is transformed and represented as a view in XUM. The semantics of submodels are explicitly captured and represented in views of XUM. Secondly,

the artifacts of views are integrated and unified into XUM.

Lastly, the manipulation of views of XUM is through XML's published interfaces based on the Document Object Model (DOM), i.e. DOM is the internal representation of XUM. Therefore, the systematic manipulation of XUM can be accomplished through the manipulation on DOM of XUM.

The unification link plays a very important role in tracking the related elements that reside in different views. These related elements may have abstraction relations, which are linked by abstraction_links. The views that share the same elements are linked by integration_links. The components in views that are related to source codes are linked by sourcecode_links. Based on these links, if any information in each view or any source code gets changed, the affected views can be reflected by following the links.

During software maintenance, modification to any submodel should be detected and reflect the impacts on the related submodels, so the semantics in each model can be updated appropriately according to the modification. Therefore, the consistency checking of modeling information of views can be assisted. Besides, the impact analysis can be applied to the entire software system, including the impact on related source codes, the impact on related design information, and the impact on related requirement information.

4. An Example

In this section, to demonstrate the feasibility of our XUM approach, we have prepared the following example: the development and maintenance of a library subsystem – a book loaning management software.

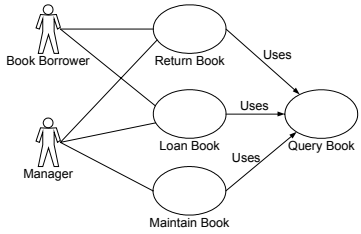

Figure 5. The use case diagram of the system

Suppose that various popular standards have been applied in the phases of the software development process. During the requirement analysis phase, the use case diagram of the system, as shown in Figure 5, is generated by the users. Following the instructions in the previous section, the corresponding XUM is derived and shown in Figure 6. Note that, in Figure 6, the fields of Abstraction_link are currently undefined and marked as "?", since we have not finished the integration and unification for views yet. These fields will be pointing to related views, such as design pattern view, collaboration diagram, and other views with abstraction relationship during integration and unification of views.

The class diagram, collaboration diagram, and design pattern diagram are created during the design phase seperately. Figure 7 shows the class diagram, while Figure 8a and Figure 8b show the partial XUM representation of used classes and associations in Figure 7 respectively, and Figure 8c shows the XUM representation of the class diagram.

```
<Requirement>
  <UseCase_Daigram>
    <Actor name="Book Borrower"/>
    <Actor name="Manager"/>
    <Usecase name="Loan Book">
        <Abstraction_link xlink:label="A_Loan_Book" xlink:title=
            "Use Case of Loan_Book" xlink:from="A_Loan_Book" xlink:to="? "/>
        <Abstraction_link xlink:from="A_Loan_Book" xlink:to=" ?"/>
        … …
    </Usecase>
    <Usecase name="Return Book">
     … …
     … …
  </UseCase_Daigram>
</Requirement>
```

Figure 6. The XUM specification of the use case diagram

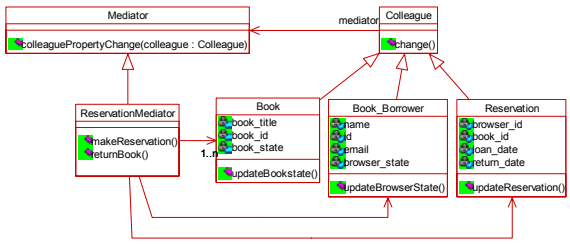

Figure 7. The class diagram of the system

```
    <Class name="Mediator">
      <Integration_link xlink:label="D_Mediator" xlink:title="Class of Mediator"/>
      <Sourcecode_link xlink:from="D_Mediator" xlink:to="S_Mediator"/>
      <Operation name="colleaguePropertyChange(colleague:Colleague)" attribute="public" />
    </Class>
    … …
    </Class>
    <Class name="Book_Borrower">
      <Integration_link xlink:label="D_Book_Borrower" xlink:title="Class of Book_Borrower "/>
      <Sourcecode_link xlink:from="D_Book_Borrower" xlink:to="S_Book_Borrower "/>
      <Attributes name="name" type="String" attribute="private"/>
      <Attributes name="id" type="String" attribute="private"/>
      <Attributes name="E-mail" type="String" attribute="private"/>
      <Attributes name="browser_state" type="Boolean" attribute="private"/>
      <Operations name="updateBrowserState()" attribute="public" />
    </Class>
    … …
    </Class>
    <Association from="Mediator" to="ReservationMediator">
       <Intgration_link xlink:label="Mediator_ReservationMediator" xlink:title="Association:
             Mediator_ReservationMediator"/>
    </Association>
… …
    </Association>
```

Figure 8a. The XUM representation of classes

```
    <Association from="Mediator" to="ReservationMediator">
       <Intgration link xlink:label="Mediator_ReservationMediator" xlink:title="Association:
         Mediator_ReservationMediator"/>
    </Association>
    <Association from="ReservationMediator" to="Reservation">
       <Intgration_link xlink:label="ReservationMediator_Reservation" xlink:title="Association:
         ReservationMediator_Reservation"/>
    </Association>
… …
    </Association>
```

Figure 8b. The XUM representation of associations

```
  <Class_Diagram>
    <Class name="Mediator">
       <Integration_link xlink:href="D_Mediator"/>
    <Class name="ReservationMediator">
       … …
    <Class name="Book">
       … …
    <Class Association from="Mediator" to="Reservation Mediator" type="generalization" client="1">
       <Integration_link xlink:title="Mediator_ ReservationMediator" xlink:lable=" Association of Mediator_
         ReservationMediator" xlink:href="Mediator_ReservationMediator" xlink:from="D_Mediator " xlink:to="D_
         ReservationMediator" />
    <Class_Association from="ReservationMediator" to="Reservation" type="dependency" client="0..n">
       … …
  </Class_Diagram>
```

Figure 8c. The XUM specification of class diagram

There are four arguments that need to be discussed further in this case study. First, the way to capture modeling information from submodels and then transform them into view representations in an XUM is quite systematic and straight forward as long as the mapping rules between two representations are well-defined in XUMM. In our approach, each submodel adopting a software standard should have its corresponding view representation. The views carrying and sharing information from the global information repository – the XUM – can explicitly and completely define the semantics of components and their relations, which may be implicitly or incompletely represented in the standard submodels.

Second, beside the transformation from a submodel to a view, since the standards are wildly accepted for modeling understanding, it is necessary to keep the two-way mapping in an XUM between the submodels and their views in order to project a standard model as needed. In an XUM, the naming of elements in views is the same as that in the corresponding submodels; therefore the two-way mapping can be achieved.

Third, as shown in the XUM representation in previous figures, the unification_links such as Integration_link, Abstraction_link, and Sourcecode_link are used to link the components and associations that share some semantic information. For example, class *ReservationMediator* in the view of the class diagram has links from the views that use this class, such as the view of the design pattern – *Mediator*, the view of the collaboration diagram, etc. Similarly, it has a link to the source code segment that implements the class.

Fourth, when modeling information is not complete, some of the unification_links may be undefined. However, these undefined links are very valuable indications to software engineers, for they indicate that the system is in a situation of incompleteness, so some enhancements are needed.

5. Conclusion

Software standards, such as UML and design pattern, are supposed to offer standard notations or proven techniques for faster and more efficient model constructions for software development. However, none of the standards are general enough to cover all the phases of the software life cycle, and few of them employ compatibility with the others. So, using these isolated standards will cause the problems of integration and consistency of the standards, and especially the more serious problems of maintenance while doing necessary alteration in models of a system.

In this paper, we have proposed an XML-based unified model, called XUM, which can integrate and unify a set of submodels with well-accepted standards of a system into a unified model represented in XML; through the unification and formal representation, XUM can not only assist software development, but also improve software maintenance. The feasibility of the approach has been verified through a set of experiments.

In our future studies, XUM and XUMM will be extended to embrace all the materials of modeling, design, implementation, and documentation for a system. Further experiments for a comprehensive XUM environment and the tool sets are being carried out to accomplish the goal of the enhancement and unification of software development and software maintenance.

References

1. Bennett, K. H. (1993). An overview of maintenance and reverse engineering, *The REDO Compendium*, John Wiley & Sons, Inc., Chichester.
2. Booch, G. (1991). *Object-oriented design with applications*. Redwood City, Calif.: Benjamin/Cummings Pub. Co.
3. Booch, G. (1994). *Object-oriented analysis and design with applications* 2nd ed. Redwood City, Calif. : Benjamin/Cummings Pub. Co., 3-25.
4. Bourdeau, R.H., & Cheng, B.H.C. (1995). A formal semantics for object model diagrams. *IEEE Transitions on Software Engineering*, 21(10), 799-821.
5. Chu, W.C., Lu, C.W., Chang, C.H., & Chung, Y.C. (2001). Pattern based software re-engineering. Handbook of Software Engineering and Knowledge Engineering, Vol. 1, Skokie, IL.: Knowledge Systems Institute.
6. Connolly, D. (2001). *The extensible markup language (XML)*. The World Wide Web Consortium. Retrieved August 21, 2001 from http://www.w3.org/XML
7. Deitel, H., Deitel, P., Nieto, T., Lin, T., & Sadhu, P. (2001). *XML how to program*. Upper Saddle River, NJ : Prentice Hall.
8. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software*. Reading, MA.: Addison-Wesley.
9. Murphy, G.C., Notkin, D., & Sullivan, K.J. (2001). Software reflexion models: bridging the gap between design and implementation. *IEEE Transitions on Software Engineering*, 27(4), 364-380.
10. Object Management Group. (2001, August). *OMG unified modeling language specification*. Version 1.4, Retrieved July 16, 2001 from http://www.omg.org/technology/documents/recent/omg_modeling.htm
11. Rine, D. C. (1997). Supporting reuse with object technology. *IEEE Computer*, 30(10), 43-45.
12. Wong, K., Tilley, S.R., MuÈller, H.A., & Storey, M.D. (1995, January). Structural redocumentation: a case study. *IEEE Software*, 12(1), 46-54.