# USING UML IN A NON-SOFTWARE DESIGN TASK: CREATING AN ELECTRONIC SOFTWARE ENGINEERING HANDBOOK

**Sergiu Dascalu**[1] **Marcel Karam**[2] **Muhanna Muhanna**[1] **Salyer Reed**[1]

[1] Department of Computer Science & Engineering
University of Nevada, Reno, USA
{dascalus, muhanna, sreed}@cse.unr.edu

[2] Department of Computer Science
American University in Beirut, Lebanon
marcel.karam@aub.edu.lb

**Abstract:** This paper describes a design experience in which UML was used in a non-traditional way, that of modeling an electronic software engineering handbook. The handbook was created by eleven students who took a graduate course in software engineering during Spring 2006 at the University of Nevada, Reno, USA. While all other course projects involved developing software applications, the electronic handbook project required putting together a comprehensive repository of student reviews on significant software engineering articles, thus creating an expandable technical report on the discipline's current landscape and future directions. By accessing this repository, one is able to adequately peruse a plethora of information on various software engineering topics and better comprehend the discipline's vernacular. This paper covers the UML-based specification and design process of the electronic handbook and provides details of the end product. Furthermore, it shows that UML can be used as a powerful modeling tool outside the software development domain. Several pointers to future developments are also presented in the paper.

**Keywords:** software engineering, electronic handbook, UML, specification, design.

## 1 INTRODUCTION

Software engineering (SE) encapsulates a broad range of subjects and themes. Because of its vastness and intricacies, the process of software engineering can become distorted in large applications, resulting in project delays and failures. To ensure systematic development and better maintain project direction, modeling languages such as the UML (Unified Modeling Language) [1, 2] are utilized during various process activities, including specification, analysis, design, deployment, and evolution.

UML resides, traditionally, in the realm of software engineering for the purpose of system representation. By transforming system components into a comprehensive model, ambiguity is removed and visualization and understanding of the system are enhanced. Thus, if utilized properly, UML is a powerful design tool used by many for the purpose of software modeling, abstraction, and refinement.

An impressive trait of UML is its ability to adapt and extend. As system components become obvious and requirements of the system are clarified, they are assimilated into the software system model, composed of several sub-models and described using the UML notation. UML's adaptability and extensibility, ensured by the language's in-built mechanisms and its own design philosophy [1, 2, 3] drive further growth and contribution.

This paper describes our recent effort to harness the power and flexibility of UML outside its traditional domain of application (that of software development), in a project concerned with the creation of an electronic handbook (compendium) entitled "Research in Software Engineering: Current Landscape and the Road Ahead".

This book, a technical report of rather large size (currently, over 200 pages), has been put together by 11 students who took in Spring 2006 the graduate course CS791z Topics on Software Engineering at the University of Nevada, Reno (UNR), USA [4] under the advisement of the course instructor, Dr. Sergiu Dascalu. The main idea behind the handbook was to create an expandable repository (collection) of reviews written by students on current relevant software engineering research literature. The reviews were organized in several major topics, each of which having dedicated a chapter in the book (as detailed later in this paper). Each chapter was created through the joint contribution of the students working on a specific assignment, e.g., the first assignment was on "major research directions on software engineering", the second on "centers of excellence in software engineering," and so on.

Typically, for each class assignment each student had to review two or three publications (journal or conference papers) on the assignment's specific topic. Having all reviews organized in chapters put together in a single handbook has the benefit of providing future students with a comprehensive reference material easily available as course material (the handbook will be made available online, on the course website [4]). Furthermore, this technical repository of SE reviews is intended to grow each year through the contribution of new series of students who will work on new topics (and thus will write new chapters of the handbook). Yet another significant benefit of the SE handbook is that when doing their assignments the students are more careful in selecting material and writing their reviews, as the idea of being co-authors of the handbook entails additional responsibility, which, as we have noticed first-hand, the students are taking quite seriously.

An interesting experience related to creating this electronic handbook on SE topics was provided by its construction process, for which two graduate students, Muhanna Muhanna and Salyer Reed, co-authors of this paper, were responsible. While all other class projects involved software modeling and implementation, Muhanna and Salyer had to create an electronic book, not a software application per se, but still a product (or "system") that required significant specification, design, implementation, and integration. The idea came rather naturally to use UML for modeling this book in a similar way a software product would be modeled, following traditional engineering phases (specification, design, implementation, integration, and evolution) applied in this case to developing and maintaining a non-software product.

This paper reports on our experience with creating the SE electronic handbook and provides further evidence that UML is a powerful modeling notation that can be used with significant advantages in the development of products and artifacts other than "software products".

In its remaining sections this paper is organized is organized as follows: Section 2 briefly reviews related work on applying UML for modeling non-software systems, Section 3 presents the main chapters of the SE handbook, Section 4 provides details of the handbook's UML model, Section 5 presents excerpts from the end product (the electronic handbook), and Section 6 finalizes the paper with several planned directions of future work and our concluding remarks.

## 2 RELATED WORK

Traditionally, UML has been used extensively for modeling software applications. Examples are abundant in the SE literature and over the last years industry practitioners have relied on UML as the main tool for designing software-intensive systems. The authors of this paper have also used UML heavily for developing their software projects, as for example reported in [5, 6].

As originally indicated by Booch, Rumbaugh and Jacobson, "the Unified Modeling Language (UML) is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive systems" [1]. The newest version of OMG's UML specification (*Superstructure*, formal document version 2.0) further points out that "UML is a language with a very broad scope that covers a large and diverse set of application domains" [7].

Although it has been acknowledged for long that UML is also suitable for business modeling and the development of a large variety of non-software products, the scientific literature contains rather few detailed reports on applying UML on such cases. Among the ones we surveyed, Torchiano and Bruno tackle the modeling of enterprise systems using UML [8], Holt addresses the use of UML in systems engineering [9], Eriksson and Penker focus on business modeling with UML [10], and McNay describes using UML in the e-commerce domain [11].

To the best of our knowledge, there is however no published report on applying UML to modeling and assembling an electronic handbook. Hence, although our work is rather preliminary, this paper provides an account of our experience on using UML in this particular type of application. Furthermore, even though the handbook we created is a specific one (on SE research), the adaptation of employing UML to designing any other type of electronic book should be straightforward and easy.

## 3 THE HANDBOOK AND ITS COMPONENTS

In essence, our "system" consists of the handbook developed by the eleven students enrolled in Spring 2006 in the course CS791z Topics in Software Engineering at UNR. Each chapter focuses on a specific area of software engineering research (and, occasionally, practice), as detailed below.

### 3.1 Chapter 1: Research Directions in Software Engineering

Software engineering is a continuously developing field in which advances occur daily. As these improvements are built upon current software engineering principles and practices, so, too, these advancements will one day be the foundation for further growth and enhancement – this is a typical iterative improvement process in software engineering.

This chapter describes directions and intent of current research efforts in software engineering. By reading this chapter, one should be able to accurately identify current research and development directions in SE.

### 3.2 Chapter 2: Major Software Engineering Centers on Geographical Areas

Software engineering is a broad field that blankets many cultural and geographical regions. In this chapter, the globe was divided into 11 geographical segments, each student being assigned a particular region. For each specific region three prominent research centers were examined in detail and several other were briefly inspected. Each student wrote a comprehensive review, summarizing important projects, initiatives, and practices pertaining to the surveyed centers.

By assembling all the reviews into a single chapter, one is able to quickly "partition" the world and examine various projects and achievements particular to a specific geographical region.

### 3.3 Chapter 3: The Software Development Lifecycle

The software development lifecycle (the software process) encompasses a number of typical phases [12, 13]. From conception to retirement, the software process is a complex cycle of intertwined, interdependent activities. Each student was assigned a specific phase of the software lifecycle and required to become well acquainted with the phase, its activities, and its terminology. To accomplish this task, students were asked to peruse various resources – primarily the IEEE and ACM digital libraries as well as the Internet – to familiarize themselves with their assigned topics. Upon successful assimilation of knowledge, students effectively summarized relevant scientific publications about their assigned phase in the software lifecycle. By merging the combined efforts of all students, the complete software lifecycle has been covered.

### 3.4 Chapter 4: Domain-Specific Software Engineering

Software is rapidly becoming an integral component in many fields of human activity. There exists a mutual benefit between the software engineering and the specific field: software runs as a set of instructions, simplifying tasks and computations in that given field, and software engineering benefits by gaining insight from the feedback provided by the field's experts (the users of the software).

In this chapter, students investigated the integration of software into various fields of activity, for example in the medical domain, in bioinformatics, and in very large control systems development. The responses from students were quite diverse – each response was unique, providing a plethora of information. By perusing the reviews gathered in this chapter, a reader can construct a good image of software engineering as an "encompassing" discipline, necessary to provide support in many domains of human activity.

### 3.5 Additional Handbook Components

Besides the four chapters included so far in the handbook, other components of the book include a glossary of terms, an index of authors (name index), an index of subject topics (subject index), a list of abbreviations, and a comprehensive list of references. As is the case with the entire compendium, this set of appendices will be the subject of future additions and enhancements.

## 4 UML MODELING OF THE HANDBOOK

UML is a descriptive, graphical language traditionally used in modeling complex software-intensive systems [1, 2]. UML can adequately describe the system, its components, and the components' relationships, both from a static (structural) and a dynamical (behavioral) perspective.

By exploiting the flexibility and descriptive power of UML the electronic compendium has been modeled in terms of specification (requirements, use case diagram, use cases, scenarios) and design (class diagram, system level diagram). Excerpts from this model are provided next.

### 4.1 System Requirements

System requirements are the services and functionality that the system is to provide; once requirements are defined, the system becomes malleable and the development can progress successfully.

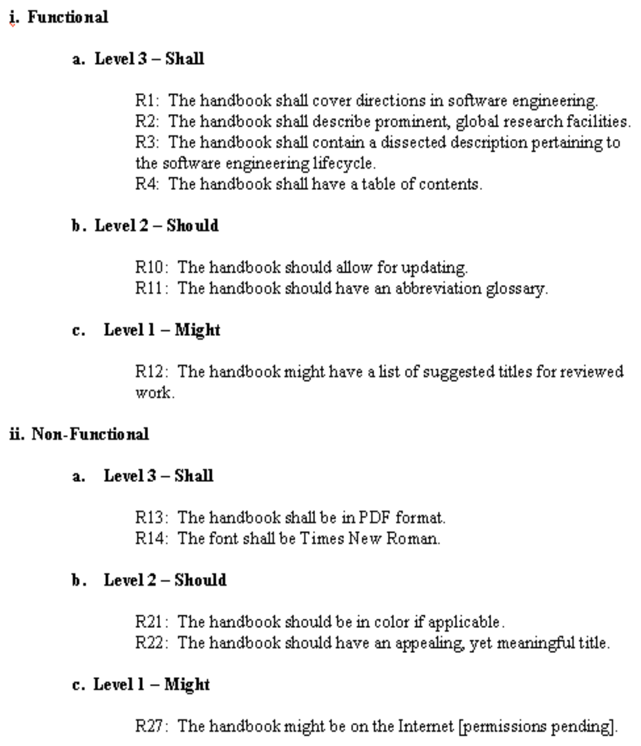
System requirements are divided into two categories: functional and nonfunctional requirements [3, 12, 13]. Traditionally, functional requirements describe the desired behavior of the system. In our electronic handbook application, functional requirements are paralleled to traditional functional requirements for software; in a sense, they are the end deliverables of the system. On the other hand, nonfunctional requirements are constraints imposed on the system, for instance implementation constraints, performance constraints, and usability constraints.

Both the functional and nonfunctional requirements of our system were divided into three levels of priority, introduced respectively by the verbs: "shall", "should", and "might". The "shall level" insists that its requirements will be implemented prior to releasing the system. The "should level" includes requirements that are important to the system, but can be omitted for the current phase without compromising the phase's objectives. Finally, the "might level" includes requirements that may be absent; however, they provide further functionality and aesthetics to the overall system and are to be implemented later.

For our SE handbook application, an abundance of requirements were attained. For brevity and illustration purposes, only several are shown in Fig. 1.

### 4.2 Use Case Diagram and Use Cases

In UML use case diagrams, actors (users) interact with the system. Currently, there are three types of actors in our system: contributors (student writers), assemblers (students in charge of creating and maintaining the handbook), and readers (everyone who reads the handbook). A partial and simplified version of the system's use case diagram is shown in Fig. 2.



**i. Functional**

**a. Level 3 – Shall**

R1: The handbook shall cover directions in software engineering.
R2: The handbook shall describe prominent, global research facilities.
R3: The handbook shall contain a dissected description pertaining to the software engineering lifecycle.
R4: The handbook shall have a table of contents.

**b. Level 2 – Should**

R10: The handbook should allow for updating.
R11: The handbook should have an abbreviation glossary.

**c. Level 1 – Might**

R12: The handbook might have a list of suggested titles for reviewed work.

**ii. Non-Functional**

**a. Level 3 – Shall**

R13: The handbook shall be in PDF format.
R14: The font shall be Times New Roman.

**b. Level 2 – Should**

R21: The handbook should be in color if applicable.
R22: The handbook should have an appealing, yet meaningful title.

**c. Level 1 – Might**

R27: The handbook might be on the Internet [permissions pending].

**Fig. 1:** Functional and Non-Functional Requirements

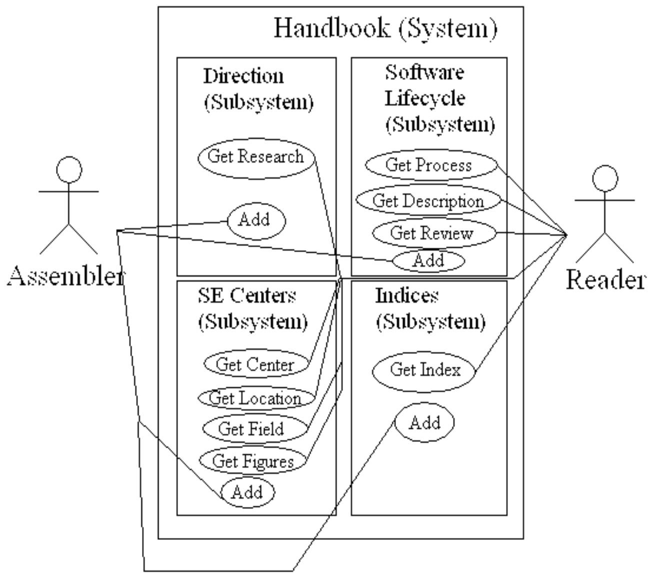


**Fig. 2:** Use Case Diagram of the SE Handbook (partial)

Contributors are the students. They partake in the creation of the handbook by submitting articles and reviews for processing and categorizing.

Assemblers are the individuals responsible for merging contributors' articles and producing the compendium. Through meticulous and careful editing, the assemblers undertook the responsibility of combining the submissions into a useful and aesthetically pleasing handbook. The assemblers are also obligated to upkeep and maintain the compilation, for the system is dynamic and new components are continuously added.

Finally, the readers – who are the consumers, or end-users – delve into the contents of the handbook. The handbook, as previously mentioned, is a tool, a roadmap to a variety of topics in software engineering research.

The use case diagram depicts the interaction between the actors and the system. Use cases provide means for implementing system requirements based on user interactions. For instance, as the handbook is dynamic, assemblers must have methods for adding new items to it; therefore, the "add item" requirement must be addressed in a use case.

Use cases are parts of the use case diagram. They describe particular interactions between the actors and the system. For illustration purposes, several use case descriptions are provided below. Larger and more detailed descriptions of use cases can also be produced following the template suggested in [3].

UC1 (Get Research): This allows the reader to obtain information on a specific research direction in software engineering.

UC2 (Add Direction): This allows the assembler to add a new SE research direction to the compiled list of directions.

UC3 (Get Center): This allows the reader to find the name of a research facility (center) with a focus on software engineering.

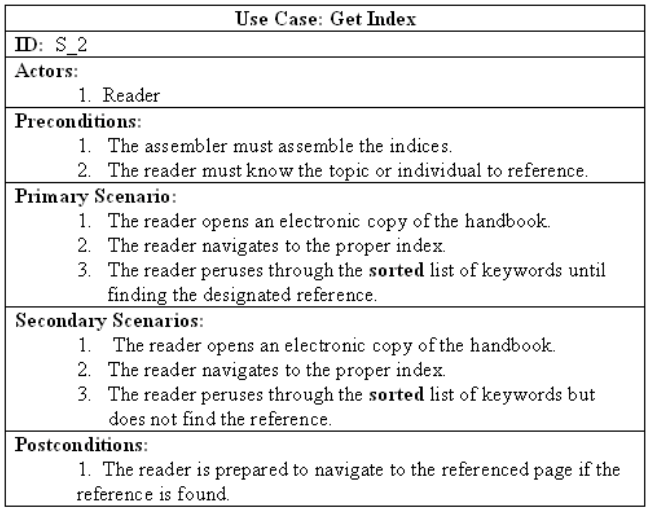
UC8 (Get Phase): This allows a reader to locate information on a specific phase in the software lifecycle.

UC9 (Get Description): This returns a short description of a specific phase in the software lifecycle.

UC12 (Get Index): This allows a reader to peruse a specific index of the handbook.

## 4.3 Scenarios

In essence, scenarios are instances of use cases [1, 3]. They are concise and precise descriptions of possible interaction between the actors and the system. Each use case has one primary (or most regular) scenario and several –often many– secondary (less frequent, or exceptional) scenarios. Prior to executing a scenario, in some cases preconditions must be satisfied – once these conditions are fulfilled, the scenario may commence. An example of scenario developed for our SE handbook is presented in Fig. 3.



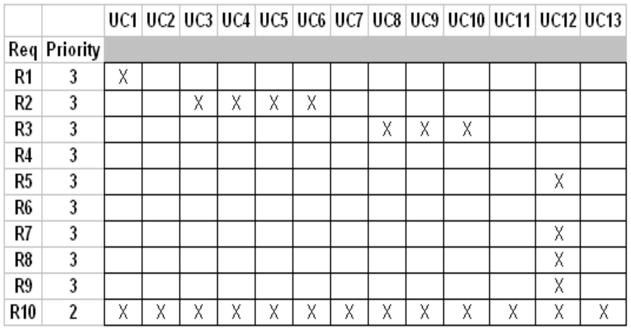
| Use Case: Get Index |
|---|
| ID: S_2 |
| **Actors:**<br>1. Reader |
| **Preconditions:**<br>1. The assembler must assemble the indices.<br>2. The reader must know the topic or individual to reference. |
| **Primary Scenario:**<br>1. The reader opens an electronic copy of the handbook.<br>2. The reader navigates to the proper index.<br>3. The reader peruses through the **sorted** list of keywords until finding the designated reference. |
| **Secondary Scenarios:**<br>1. The reader opens an electronic copy of the handbook.<br>2. The reader navigates to the proper index.<br>3. The reader peruses through the **sorted** list of keywords but does not find the reference. |
| **Postconditions:**<br>1. The reader is prepared to navigate to the referenced page if the reference is found. |

**Fig. 3:** Example of Scenario for the SE Handbook

In modeling our electronic compendium scenarios were useful to identify and describe in detail a large variety of interactions between the actors (writers, assemblers, readers) and the system (the electronic book).

### 4.4 Requirements Traceability Matrix

This traceability matrix (partially shown in Fig. 4) is a mapping of requirements onto use-case scenarios and vice versa. It is a useful tool that can be used through product development to trace back implementation and design to requirements specification.



| | | UC1 | UC2 | UC3 | UC4 | UC5 | UC6 | UC7 | UC8 | UC9 | UC10 | UC11 | UC12 | UC13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Req | Priority | | | | | | | | | | | | | |
| R1 | 3 | X | | | | | | | | | | | | |
| R2 | 3 | | | X | X | X | X | | | | | | | |
| R3 | 3 | | | | | | | | X | X | X | | | |
| R4 | 3 | | | | | | | | | | | | | |
| R5 | 3 | | | | | | | | | | | | X | |
| R6 | 3 | | | | | | | | | | | | | |
| R7 | 3 | | | | | | | | | | | | X | |
| R8 | 3 | | | | | | | | | | | | X | |
| R9 | 3 | | | | | | | | | | | | X | |
| R10 | 2 | X | X | X | X | X | X | X | X | X | X | X | X | X |

**Fig. 4**: Requirements Traceability Matrix (partial)

By using the matrix shown in Fig. 4, the handbook's requirements and use-cases are clearly interrelated. The matrix maps requirements to use cases, thus assisting in the detailed construction of the compendium.

### 4.5 Architectural Design

High-level architecture in UML is a visual depiction of the system's objects and their relationships. To model the handbook, a class diagram was produced. Elements within the hierarchy are classes; each class has designated properties (attributes) and functions (operations). Through inheritance a parent class passes its traits to its children, which in turn are able to append their own properties and functions. Fig. 5 presents the main components of the class diagram developed for the SE handbook.

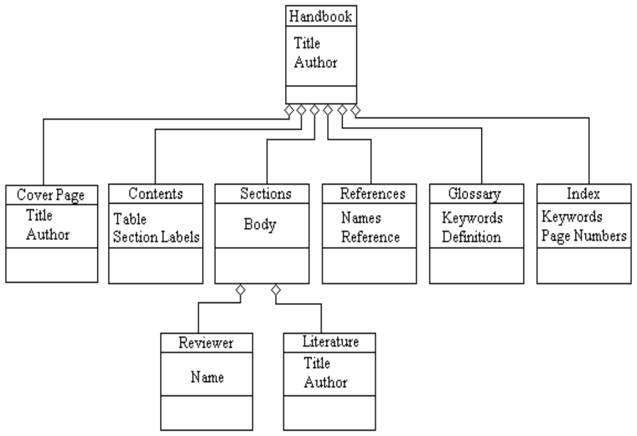


**Fig. 5**: Class Diagram of the SE Handbook (partial)

## 5 THE END PRODUCT

The end product of our work is the initial version (or, the first edition) of the electronic handbook on SE research topics. Currently put together as a PDF file, it will be soon made available online via the instructor's course website at UNR. While the design layout and the general contents for the 2006 edition have been completed some additional proofreading and editing is needed before public presentation. Excerpts from this compendium are shown in Fig. 6 (the cover page) and Fig. 7 (the table of contents).

## 6 FUTURE WORK AND CONCLUSIONS

Naturally, future work for our SE handbook includes addition of new chapters (on new SE topics) and extension of the existing indexes, the glossary of terms, the list of abbreviations, and the references. In terms of UML modeling, it would be interesting to investigate the suitability and application of other UML constructs to the design and documentation of a generic electronic book. Such constructs could include sequence diagrams, statecharts, activity diagrams, and deployment diagrams. Using UML for designing other non-software products (e.g., hardware components or mechanical devices) is yet another area of possible future exploration.

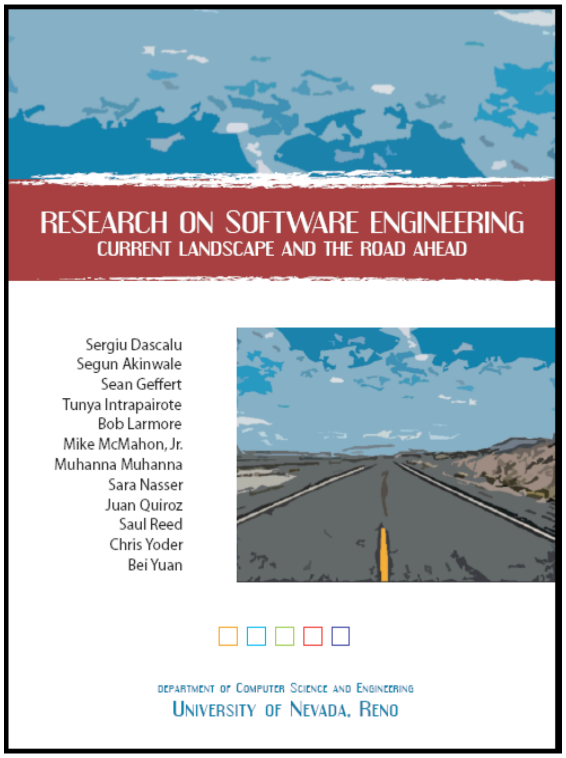
Although the handbook has been the main deliverable of our work, the rewards of this joint effort go beyond the bounds of its publication. The students who compiled the compendium benefited from their experience by assimilating new knowledge in the area of SE and by exercising their analytical and technical writing skills. Furthermore, for the authors of this paper, relying on UML for developing a non-software product has proven to be a useful experience, which opens perspectives for similar endeavors in undertaking other non-software design tasks.

## Acknowledgements

We would like to thank all the students who contributed, as part of their coursework, to the writing of the SE handbook described in this paper. Besides the paper's co-authors, Muhanna Muhanna and Salyer Reed, other students who wrote review material included in the handbook were: Olusegun Akinwale, Sean Geffert, Tunya Intrapairote, Robert Larmore, Mike McMahon, Jr., Sara Nasser, Juan Quiroz, Chris Yoder, and Bei Yuan.

## References

[ 1] Booch, G., Rumbaugh, J., and Jacobson, I., *The Unified Modeling Language: User Guide*, Addison-Wesley, 1998.
[ 2] OMG's UML Resource Page, accessed May 20, 2006 at http://www.omg.org/uml/
[ 3] Arlow, J., and Neustadt, I., *UML and the Unified Process: Practical Object-Oriented Analysis and Design*, Addison-Wesley, 2002.
[ 4] UNR, CSE Department, CS791z Topics in Software Engineering, course website 2006, accessed May 5, 2006 at http://www.cse.unr.edu/~dascalus/tse2006.html
[ 5] Dascalu, S.M., and Hitchcock, P., "An Approach to Integrating Semi-formal and Formal Notations in Software Specification," *Procs. of the 2002 ACM Symposium on Applied Computing*, Madrid, Spain, 2002, pp. 1014-1020.
[ 6] Kallman, J., Minaie, P., Truppi, J., Dascalu, S.M., and Harris, F.C., Jr., "Software Modeling for Open Distributed Network Monitoring Systems," *Lecture Notes in Computer Science LNCS-3126*, Springer-Verlag, 2004, pp. 158-169.
[ 7] Object Management Group (OMG): *Unified Modeling Language: Superstructure*, version 2.0, August 2005, http://www.omg.org/docs/formal/05-07-04.pdf
[ 8] Torchiano, M. and Bruno, G., "Enterprise Modeling by Means of UML Instance Models," *ACM Software Engineering Notes*, 28(2), March 2003.
[ 9] Holt, I., *UML for Systems Engineering: Watching the Wheels*, 1st ed., Inst. of Electrical Engineers (IEE), 2001.
[10] Eriksson, H.E. and Penker M., *Business Modeling With UML: Business Patterns at Work*, Wiley & Sons, 2000.
[11] McNay, H.E., "UML for E-Business: New Use for Use Cases," *Proceedings of IEEE IPCC 2001*, pp. 245-249.
[12] Sommerville, I., *Software Engineering*, 7th Ed., Addison-Wesley, 2004.
[13] Pressman, R., *Software Engineering: A Practitioner's Approach*, 6th Ed., McGraw-Hill, 2004.


RESEARCH ON SOFTWARE ENGINEERING
CURRENT LANDSCAPE AND THE ROAD AHEAD

Sergiu Dascalu
Segun Akinwale
Sean Geffert
Tunya Intrapairote
Bob Larmore
Mike McMahon, Jr.
Muhanna Muhanna
Sara Nasser
Juan Quiroz
Saul Reed
Chris Yoder
Bei Yuan

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
UNIVERSITY OF NEVADA, RENO

**Fig. 6:** The Handbook's Cover

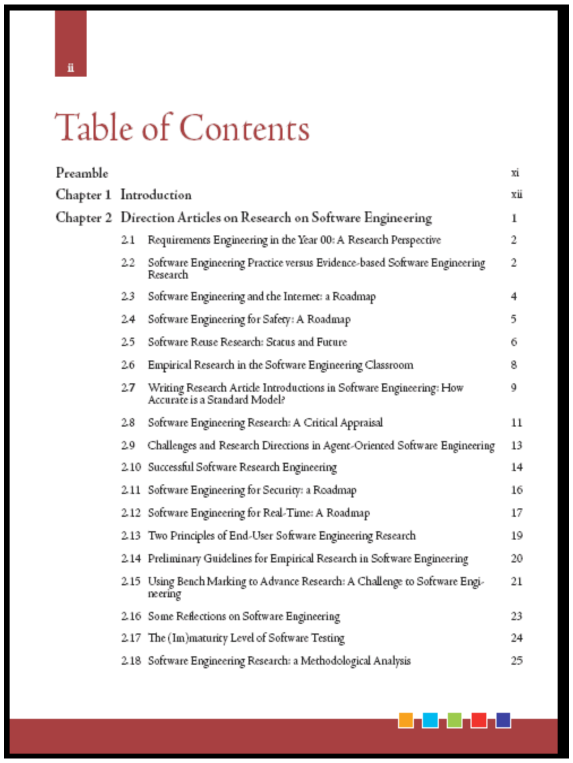

ii

Table of Contents

**Fig. 7:** Table of Contents (partial)