ISSN: 1545-679X

Information Systems Education Journal

Volume 4, Number 64 http://isedj.org/4/64/ August 28, 2006

In this issue:

Robust Software Development: A Technical Approach Using the Rational Unified Process

Robert F. Roggio
University of North Florida
Jacksonville, FL 32224 USA

Abstract: Most computer science (CS) and computer information sciences (CIS) programs require one or more courses in software development. Within computer science programs, the courses are normally entitled software engineering or senior design project, whereas within CIS programs, software development is often called Systems Analysis and Design and is (more often than in CS programs) a two course sequence. Often considered a capstone sequence, there is a wide range of instructional approaches. In many cases the chosen approach is derived from the academic unit within which the CIS program is offered. Schools of Business, Schools of Arts and Sciences, or Schools of Engineering often approach the sequence differently. This paper presents a comprehensive approach to teaching a two-course software development sequence in a CIS program taught within a College of Computing, Engineering, and Construction. The sequence contains a modest treatment of business concepts coupled with heavy emphasis on a disciplined development process using the Rational Unified Process (RUP) in deference to more traditional instruction which often emphasizes business concepts with less emphasis on software development. A brief discussion of topics found in more customary approaches is followed by a detailed description of eleven project deliverables required in the author's approach. The paper concludes with student feedback and lessons learned.

Keywords: capstone software development, process, IBM Rational Unified Process, RUP

Recommended Citation: Roggio (2006). Robust Software Development: A Technical Approach Using the Rational Unified Process *Information Systems Education Journal, 4* (64). http://isedj.org/4/64/. ISSN: 1545-679X. (Also appears in *The Proceedings of ISECON 2005:* §2362. ISSN: 1542-7382.)

This issue is on the Internet at **http://isedj.org/4/64/**

The **Information Systems Education Journal** (ISEDJ) is a peer-reviewed academic journal published by the Education Special Interest Group (EDSIG) of the Association of Information Technology Professionals (AITP, Chicago, Illinois). • ISSN: 1545-679X. • First issue: 8 Sep 2003. • Title: Information Systems Education Journal. Variants: IS Education Journal; ISEDJ. • Physical format: online. • Publishing frequency: irregular; as each article is approved, it is published immediately and constitutes a complete separate issue of the current volume. • Single issue price: free. • Subscription address: subscribe@isedj.org. • Subscription price: free. • Electronic access: http://isedj.org/ • Contact person: Don Colton (editor@isedj.org)

EDSIG activities include the publication of ISEDJ, the organization and execution of the annual ISECON conference held each fall, the publication of the Journal of Information Systems Education (JISE), and the designation and honoring of an IS Educator of the Year. • The Foundation for Information Technology Education has been the key sponsor of ISECON over the years. • The Association for Information Technology Professionals (AITP) provides the corporate umbrella under which EDSIG operates.

Robust Software Development: A Technical Approach Using the Rational Unified Process®

Robert F. Roggio
Department of Computer and Information Sciences
University of North Florida
Jacksonville, FL 32224
broggio@unf.edu

Abstract

Most computer science (CS) and computer information sciences (CIS) programs require one or more courses in software development. Within computer science programs, the courses are normally entitled software engineering or senior design project, whereas within CIS programs, software development is often called Systems Analysis and Design and is (more often than in CS programs) a two course sequence. Often considered a capstone sequence, there is a wide range of instructional approaches. In many cases the chosen approach is derived from the academic unit within which the CIS program is offered. Schools of Business, Schools of Arts and Sciences, or Schools of Engineering often approach the sequence differently. This paper presents a comprehensive approach to teaching a two-course software development sequence in a CIS program taught within a College of Computing, Engineering, and Construction. The sequence contains a modest treatment of business concepts coupled with heavy emphasis on a disciplined development process using the Rational Unified Process (RUP®) in deference to more traditional instruction which often emphasizes business concepts with less emphasis on software development. A brief discussion of topics found in more customary approaches is followed by a detailed description of eleven project deliverables required in the author's approach. The paper concludes with student feedback and lessons learned.

Keywords: capstone software development, process, IBM Rational Unified Process, RUP

1. INTRODUCTION

There are a number of outstanding books that many computer programs use to teach software development. These books typically support a two course structure, where the first part usually consists of systems analysis while the second part focuses on systems design and implementation. Many undergraduate programs require such a capstone sequence prior to graduation. The backgrounds of students may vary considerably depending upon the academic unit within which the CIS program resides. Programs administered within an engineering college will usually require students to have taken courses in programming, data structures, database processing, communications, networking, architecture, and perhaps some Internet programming. Programs administered in a School of Business generally require more coursework in business topics typically offered in departments such as Management, Marketing, Information Systems, Decision Sciences, Accounting and Finance, and Economics. Regardless, this culminating sequence in undergraduate studies is oftentimes the place where students are challenged to marshal their knowledge and develop a real-life business application. In some programs, this application may synthesize many business concepts involving people, procedures, information, and process. Other applications may require an approach that involves capturing and modeling user requirements, developing a design, and ultimately implementing the design us-

ing various modeling languages and programming technologies.

At The University of North Florida (UNF), the Information Systems (IS) program is administered in a CIS department within the College of Computing, Engineering and Construction. The department offers undergraduate degrees in computer science, information systems, and information sciences. Undergraduate programs in computer science and information systems are accredited by ABET/CAC. (An MS in CIS with tracks in software engineering, information systems, and computer science is also available.)

Students in the Information Systems (IS) program are required to take an eighteen-hour business minor. Additionally, many IS students minor in computer science. The business courses are taught by various departments in the College of Business Administration, which is AACSB-accredited.

Until recently the capstone IS courses, called Senior Project 1 and Senior Project 2, have been taught using the Whitten book (Whitten, 2001), which the faculty found very satisfying and up to date.

The Whitten text has chapters devoted to Information System Building Blocks, Information Systems Development, Requirements Discovery, Process Modeling, Database Modeling, Systems Design, Input/Output Prototyping, Project Management and more. I have personally used this book and it continues to be the text of choice by some faculty members who also teach the capstone sequence. System Architect®, ERWin®, Oracle®, Java, JavaScript, HTML and some XML are typical technologies used to support development of a real-world application.

The graduates of UNF computing programs (both computer science as well as information systems) are expected to serve in a variety of capacities in the workplace. Most of the IS graduates (in contrast to MIS graduates produced by the College of Business Administration) are expected to enter the workplace and become involved in some phase of software development. The local region continues to have a significant reliance on mainframe technologies (COBOL, database, etc.) while migrating in many cases to the newer architectural models that are mostly highly-distributed, multi-platform, client-server approaches. For these graduates, their minor in business courses serves them well. But the lack of familiarity with the more modern software development approaches demanded attention.

As a result, an alternative approach to the well-established approach was developed and implemented for the last two years in selected section offerings of the capstone sequence. This approach incorporates the Rational Unified Process (the RUP®), a disciplined development process, with Rational Rose as the primary support tool for capturing a number of models and views developed during the two semesters. Because the students have had a number of programming courses (COBOL, File Structures, Java programming, Data Structures with OOP) plus at least one course in Database Processing, courses in networking, Internet Programming and more, the technology base for the students is well-established. Applying these technologies with the framework of an industrial-strength process such as the RUP® appears to have favorably addressed the need of modernizing the capstone sequence experience for our students and regional constituency.

While a number of topics covered in the Whitten book continue to be included in this approach, they are addressed within specific deliverables. Prototypes are built, executed, and evaluated. Stakeholder needs and features are documented using a Vision document. Business Modeling and domain modeling are parts of required deliverables. Use Case specifications developed in Microsoft Word® and Use Case Diagrams developed and captured in Rational Rose® are used to model the functional requirements; architectural systems design is captured through subsystems in interaction diagrams at various levels of abstraction. Fully-attributed lists and schema are developed. Databases using Oracle 9i are built, and middleware (java.sql) assists in interfacing.

Some important business concepts are either not covered or covered weakly. Business Process Reengineering (BPR), concepts of project management, Data Flow Diagrams (DFDs) and Entity Relation Diagrams (ERDs) are only briefly mentioned. Topics such as cost-benefit analysis return on investment (ROI), decision tables, and different compo-

nents of feasibility are weakly covered. Other diagrams (Fishbone diagrams, PERT, Gantt Charts, Microsoft Project®, for example) are not included. Some of these topics, however, have been covered in business courses the students have already taken.

2. A SHIFT TOWARD THE IBM-RATIONAL UNIFIED PROCESS (RUP)

Although there remains a good deal of traditional software development using modified waterfall models as well as a number of incremental and/or evolutionary approaches occurring world-wide, many newer project development efforts in the workplace are shifting toward use of the object-culture and a supporting process that uses modern development tools and notations. Graduates of many CIS programs are expected to not only have an appreciation of the business enterprise, but they must be equipped with problem-solving skills, analysis and programming skills (particularly using object oriented languages), UML, and knowledge about different development processes. While no single development approach fits all environments, many enterprises are training their software project managers and software developers in OO languages and the effective utilization of a disciplined, incremental and iterative process in particular, the Rational Unified Process (RUP) ® together with associated tools.

The RUP® is defined as a "use-case driven, architecture-centric, iterative development process" (Kruchten, 2004). These three descriptors characterize the process well. But it is the real integration of these principles and practices into the day-to-day development activities that will more likely provide for the more reliable delivery of software on-time, within budget that meets or exceeds customer requirements.

Three years ago, the North Florida Rational Users Group (http://www.nf-rug.com) was formed in Jacksonville, Florida to provide a forum for professional software developers, managers, business analysts, and students to meet and discuss myriad issues of common interest related to software development. This user group is currently the largest group in number in the contiguous United States (Roggio, 2004) and arose out of genuine need to share experiences and problems. This group meets monthly on the campus of UNF and has guest speakers, group discussions, and workshops.

With this backdrop a redesigned course sequence Senior Project 1 and 2 has been revamped and offered with tremendous success in the North Florida market. The sequence is continuously undergoing change and refinements, and suggestions are always welcome.

3. DEVELOPMENT FRAMEWORK FOR CIS SENIOR PROJECTS

The course descriptions break down the many activities of the sequence into eleven deliverables, whose contents approximately follow RUP guidelines. (One may view my no-frills web page that includes slides, deliverables, examples of student work, and more at: http://www.unf.edu/~broggio/Spring2005.html).

The lectures and deliverables have been modified each time the course sequence has been offered – in the true iterative spirit of this process. It is doubtful that a firm set of lectures and deliverables will ever be completed. And, while no locked-in, final set of student deliverables is anticipated, the following scenarios are offered in the spirit of collegial sharing – taking (if desired) what might fit into a program, discarding other parts, and, most importantly, improving and refining the deliverables for different needs and constituencies graduates may serve.

The textbooks used for the two course sequence are listed in references (Kruchten, 2004), (Kulak, 2004), (Lethbridge, 2001), and (Quatrani, 2004). These books represent topics/examples dealing with visual modeling, the RUP, use case design and evolution. The Lethbridge book represents a more rigorous software engineering book used more in the second semester for software architecture, design patterns, and frameworks.

The course also requires the availability of IBM's Rational Rose, which can be easily acquired with licenses readily issued through the IBM Scholar's Program for free. At UNF, there are twenty-five floating licenses and a node-locked version for a notebook computer. Installation and maintenance on local servers for student laboratories is the responsibility of the local university, and the license granted by IBM is annual and renew-

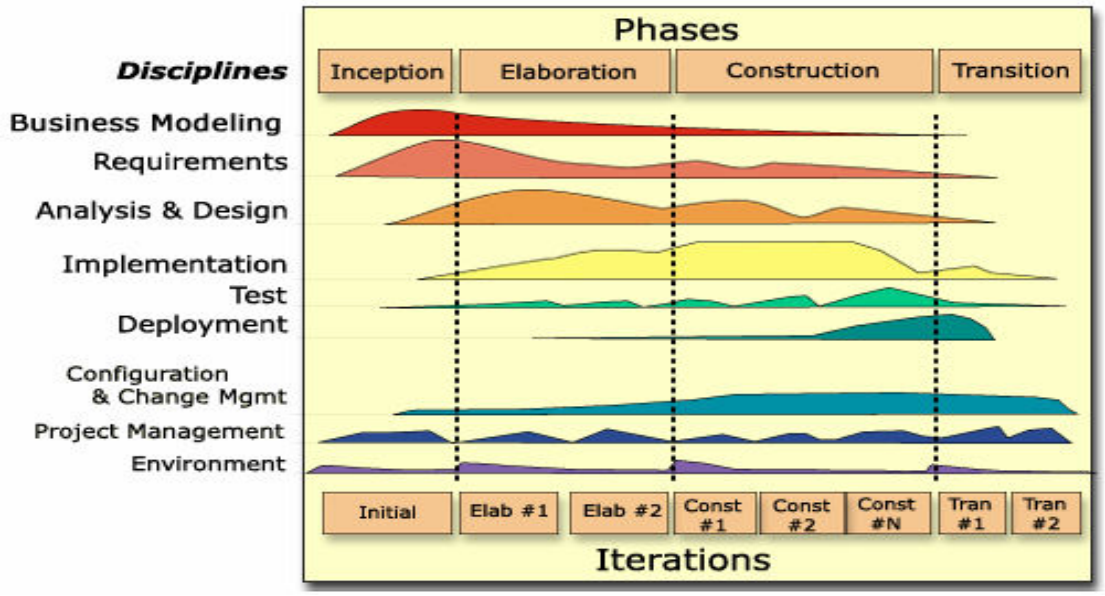

Figure 1. Phases, Iterations, and Disciplines in the RUP (Kruchten, 2004)

able. Deliverables by student teams were also supported using Blackboard, which proved to be excellent for sharing files and similar artifacts in development. Local servers supporting Java Server Pages (jsp) and Oracle 9i® or MySQL® were also made available and maintained by technical support personnel on departmental servers.

4. DELIVERABLES

Framework

Media: As a significant departure from accustomed heavy-paper approaches, all deliverables are submitted on a single CD. The opening window on a team CD (via Explorer or My Computer) is to display separate folder icons entitled, Deliverable #1 through Deliverable 11. That's it.

General Structure: All deliverables contain an Executive Summary which outlines the content of the deliverable; schedule and individual tasking and hours expended on various tasks are included. Within the deliverable folder are two subfolders nominally entitled Artifacts and Management Documents. All Rose Models, Use Case Specifications, and associated Word documents are to be placed in the Artifacts folder, while documents such as the Vision Document, Business Rules, Glossary, and the Risks List are found in the Management Documents folder. Students must include a form that acknowledges grammar and misspellings are unacceptable. To the dismay of a few, this was enforced.

Framework for Deliverables: The beginning of the course sequence includes discussions and readings on best practices of software development such as monitoring change, visual modeling, use-case driven, architectural-centric, iterative development, and establishing baseline architectures. Considerable time is spent discussing RUP features with particular emphasis on time-boxed iterations, phases, cycles, core and supporting disciplines and workflows (Figure 1). Included also are the basics of the Unified Modeling Language (UML) notation and the fundamentals of 'object culture' itself.

Deliverable #1. Business Case; Domain Analysis: Objective: To understand the structure and dynamics of the organization itself. This is realized in part via a business vision document. Major business functions and business actors are modeled. Applications cannot be developed without developers understanding and appreciating the environment within which the application will operate.

The Business Case also includes several models. In addition to the business vision document, business use case models and

the Business Object Model are developed and captured in Rational Rose. Also included are a Risks Lists document and a Business Rules document, both text documents. A template of downloadable document formats for a variety of documents is available at http://jdbv.sourceforge.net/RUP.html.

Deliverable #2. Domain Model, Vision Document and Statement of Work (SOW): Objective: Establish the domain model, a vision document for the application to be developed and a statement of work. The Domain Model is really part of the business case and is essential to understanding important features and key abstractions in the environment. (Developing the Domain Model was moved to the second deliverable due to the volume of work and initial learning of tools in Deliverable #1.) The Vision Document is an essential document and must provide a very high level list of user needs and 'features' that the application must accommodate (Leffingwell, 2002). These features are usually text and not behavioral. Sample descriptions of features that an application entitled ClassicsCD.com Web Shop might include are: the need for a secure payment method; need for the customer to be able to easily browse for available titles; ability of the customer to check the status of an order; email notification to customers; to have the catalog highly scaleable to include many titles and effective searching through those titles; and ability for customer to register for future purchases without needing to re-enter personal information.

The domain model, a graphical model and precursor to Use Case specification, is very helpful for use case development, as the use cases will contain terms and perhaps acronyms from the glossary (Deliverable #1) and references to key business abstractions (business entities) from the domain model. The glossary and domain model provide a common basis for understanding the language used in capturing the functionality in use cases.

The abstractions found in the domain model itself contain business entities with their associations, dependencies, attributes and other relationship features. The attributes and relationships among these entities represent key entity connections that serve as strong candidates for understanding the business enterprise and also for reuse across a number of specific applications that may be developed within the business enterprise. The Domain Model is developed in Rational Rose® and captured in a separate folder under the Logical View in the Rose Browser.

The Statement of Work (SOW) addresses the team plan: tasks and responsibilities, tentative schedule and deliverables, assignment of roles, and similar activities. In each deliverable, artifacts from previous deliverables are reviewed and updated as needed.

Deliverable #3. Use Case - Façade Iteration; Initial User Interface Prototype: Objective: to develop a set of façade use cases for the new application and to develop an initial user interface prototype. While considering the merits of a number of different formats and templates, students create a first-cut set of use cases by subscribing to façade level use case formats found in (Kulak, 2004). In identifying use cases themselves, emphasis in developing the façade level use case specification is on a use case name in verb-object format and identification of actors. Specific scenarios are not developed at this time. But triggers, pre and post-conditions, assumptions and links to business rules document and the business risks document are included. An Actors package and a Use Case package are developed within the Use Case View in the Rose browser. Within the Use Case package, use case diagrams are drawn to accompany the façade use case textual specifications, developed in Word®. The emphasis on this introductory experience with use cases is to provide familiarization with the structure and format of use cases and to set the stage for developing more mature use cases for a comprehensive specification.

Students are able to select the technology of their choice in designing an initial user interface prototype. The emphasis on a first cut prototype of the user interface is to expose the student to the importance of understanding what 'utility' and 'usability' of a user interface really entail. Recognition that to the end user, the user interface 'is' the application is stressed!

From the application's development, the objective of developing the user interface is to ensure that the functionality captured in the use cases is accommodated and understood by all stakeholders. Oftentimes the devel-

opment of the interface may indicate features not captured in the use case. While the full interface is not developed here, basic functionality is shown. For a number of students, this deliverable is often an exercise in learning / using technologies with which they are unfamiliar.

Deliverable #4. Fully Developed Use Case Models and Activity Diagrams: The objective of Deliverable #4 is to fully develop the use case specifications including all scenarios; that is, the basic course of events (happy path) and identification of alternative paths and exception paths. A basic flow developed first. Additional scenarios are developed second. The use cases are documented in a complete (but certainly not 'final') form. Activity Diagrams (one per use case) is also required for the deliverable. Starting with this deliverable, the amount of work required increased significantly for the teams.

The development of the basic course of events and both alternative and exception flows in use cases requires substantial team effort. Extension points were introduced and used to link to alternate scenarios as well as any sub-flows as appropriate. Differences between 'included' and 'extended' use cases are covered. Scenarios containing glossary terms or references to business entities in the domain model have these terms 'bolded' for emphasis (Figure 2). This technique provides visible linkage to descriptions found in these artifacts.

Use cases diagrams are grouped by major key functions into specific, named packages in the Rose Browser within the Use Case View. The use case specifications continue to be developed in Word®; the use case diagrams are developed in Rose. No attempt was made at this time to capture non-functional requirements; rather, the emphasis is on recognizing use cases as behavioral models (stories) of actor(s) interacting with the application.

Activity Diagrams for each use case capture the essence (all scenarios) of a use case. This requires additional modeling using the Rose browser to associate the activity diagram with a specific use case. The use of activity diagrams is open to debate, as some practitioners do not build them at all. Other practitioners use both use case specifications and activity diagrams; still others develop activity diagrams and then discard them once the entire use case specification is developed. The author views the activity diagrams as a single visual model of a use case with all of its scenarios. Regardless, it is a worthwhile experience for students to develop activity diagrams.

Deliverable #5. Developing the Analysis Model and Capturing Non-Functional Requirements: As the last deliverable in the first semester, the objective of this deliverable is to require team members to carefully study the scenarios of each use case and to develop analysis models (structural and behavioral) of each use case. Using the narratives of the use case specifications together with the user interface prototype, students develop a structural model, which is an analysis model using a set of analysis classes (boundary, control, and entity) for each use case in accordance with RUP® philosophy. This static model supports each use case. The behavioral model is realized via a sequence diagram and is required to model the basic course of events in each Use Case. The structural model presents a grouping of classes that in RUP® technology are referred to as the View of Participating Classes (VOPC). This VOPC provides a model of classes and the relationships (associations and dependencies) needed among them to realize the functionality captured in the Use Case. While classes in an analysis model are typically incomplete, a realistic first cut is undertaken. Classes are developed using a 'responsibility approach' where data together with methods that need the data are encapsulated. The notion of separation of concerns (boundary, control, entity) is emphasized. Those analysis classes that may / may not morph into design classes and others that may be accommodated via reverse engineering during design are discussed. The models themselves are developed in Rose. A sample VOPC is presented in Figure 3. Not a great deal of time is spent on analysis modeling. But this activity serves as an important bridge to the Design Model.

While the behavioral model (captured in a sequence diagram) is realized by a collaboration of objects and their responsibilities shown by message passing, the student is thrust into the detailed nature of object culture – knowledge that becomes vital during

Use Case Number:	003	
Use Case Name:	Maintain Agent Listings	
Actor (s):	Agent, RDBMS, Web Viewer, Account Rep, System Manager	
Maturity: (Façade/Focused/…	Focused	
Summary:	The sequence of actions available to maintain an Agent's listings. The Actors can add, delete, or update a listing from here.	
Basic Course of Events:	**Actor Action** 1. Actor Agent selects maintain listings. 3. Actor Agent enters **user ID** and **password**. 5. Actor RDBMS replies with information if found. 8. Actor RDBMS retrieves and supplies to the system the Actor's **personal information** and list of **listings**. **{Modify Listing}** 10. Actor Agent selects create listing **…(more)**	**System Response** 2. System asks for user authentications 4. System receives Actor's **user ID** and **password** entry and sends request to RDMS. 6. System authenticates user ID and password. **{User not Authenticated}** 7. System requests from RDMS Agent's **personal information** and list of listings. 9. System presents Agents' page. 11. System presents blank input page. **… (more)**
Alternative Paths:	See end of document.	
Exception Paths:	E1. If user not authenticated, System asks for username and password again. Repeat 5 times till user authenticates. If not authenticated display error, then return to flow of events. E2. If Actor tries to create listing without entering data. System asks for required fields to be entered. Returns to flow of events.	
Extension Points:	**{User not Authenticated}** see exception E1. **{Missing required fields}** see exception E2. **{Modify Listing}** see alternative path **A1** **{Change Listing Status}** see alternative path **A2** **{Delete Listing}** see alternative path **A3**	
Triggers:	Actor has selected the option to maintain listings.	
Assumptions:	Actor has entered the proper URL into the browser. Actor has successfully authenticated and logged in.	
Preconditions:	Actor Agent has successfully authenticated and logged in. Agent of interest has been identified.	
Post Conditions:	Agent is returned to maintain listings menu.	
Reference: Business Rules:	4.2.1 Agent Account 4.2.2 Fee 4.2.3 Agent Listings 4.2.4 Delinquent Agent 4.2.5 Security	
Reference: Risks	2.2 Human Risk: Unauthorized manipulation of content (illegal hacker) 2.3 Human Risk: Legal liability due to misrepresentation of property 2.5 Technical Risk: Data Stagnation 2.6 Technical Risk: User GUI not user friendly	
Author(s):	Team One	
Date:	08/Nov/2004	

Figure 2. Sample Mature Use Case (Student Work: A Real Estate Application)

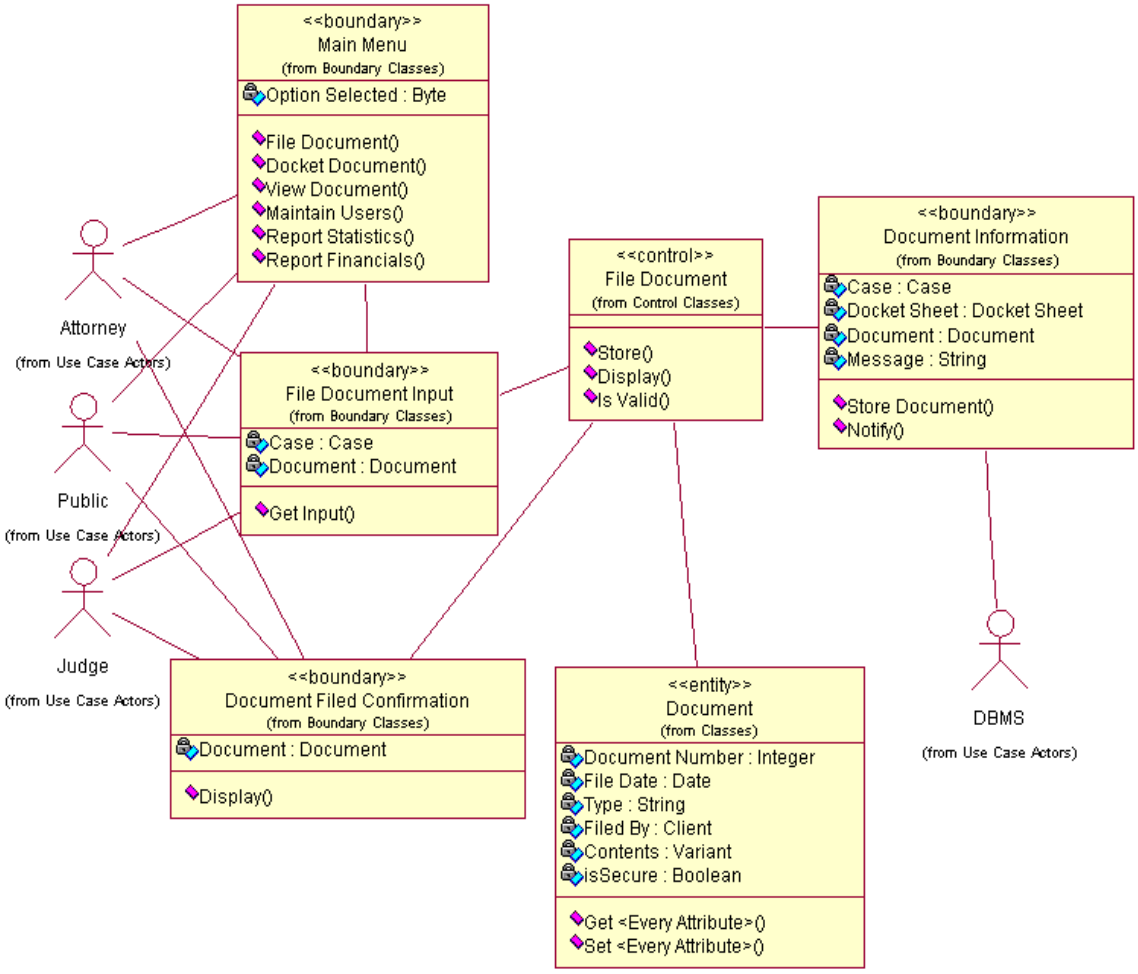

Figure 3. Sample Analysis Model (Student Work: A Court Documents Application)

design, when real design objects are built with many associations and dependencies. Seeing the explicit nature of message passing and object collaboration in meeting the requirements for satisfying a Use Case scenario is eye-opening for many.

Non-functional requirements (e.g. persistency, security, distribution, legacy, and others) are treated rather superficially, captured in a Word® document, and placed in the Artifacts folder of the deliverable. This first formal capture of non-functional requirements serves as a backdrop for much more serious consideration later in design.

Deliverable #6. User Interface Design: The objective of deliverable 6 is to take a second look at the user interface prototype (UI) and be certain that the customer and users are 'on board' with the interface and ensure any additional functional requirements suggested by the interface are now obtained. As the first deliverable in the second semester, this deliverable is less demanding by design but nevertheless important. Teams reestablish meeting schedules and revisit their status in the application development. Teams have a reasonably good set of Use Cases, an Analysis Model, and now an iterated user interface. Students select or extend their interfaces using a variety of technologies (HTML, XML, JavaScript, java server pages, and more). But the main objective is to ease students into the second semester and to rekindle the importance of the UI and attempt to capture a comprehensive set of requirements – as much as possible.

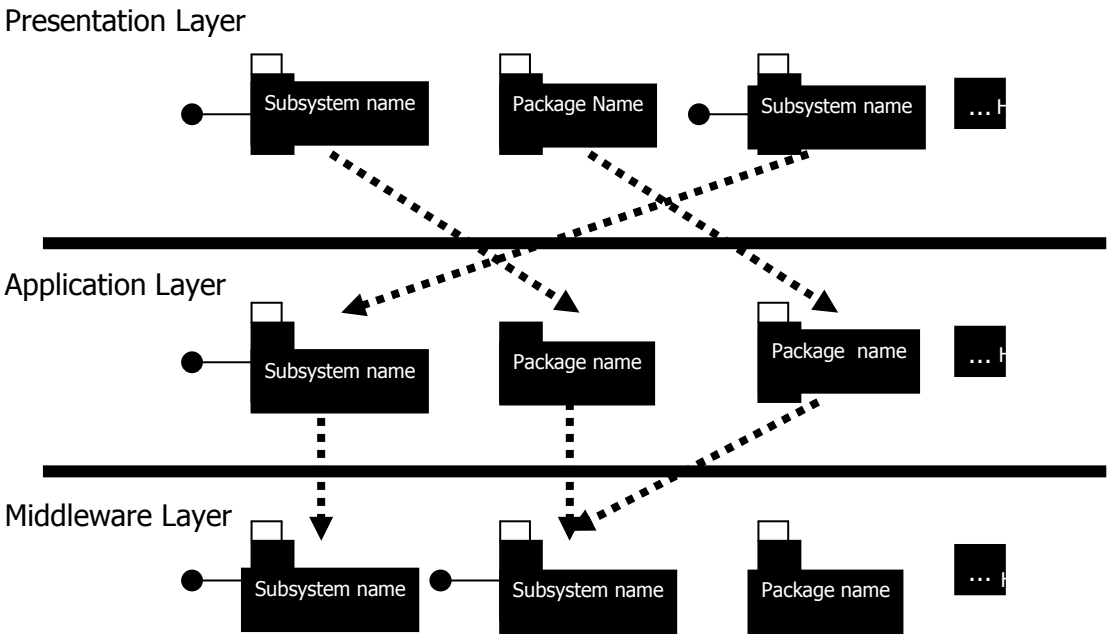

Figure 4. Template: Partial Layered Architectural Approach (generic)

Principles of Usability and Utility are more heavily emphasized than in the earlier deliverable. Verification consists of comparing the interface to use case specifications in order to conclude all the functionalities found in the use cases are in fact accommodated or implied.

Deliverable #7 - Layered Architecture: This deliverable establishes an architectural baseline for detailed design work. As a heavyweight deliverable for students, it contains very critical design philosophy. As approximately the second iteration in the Elaboration Phase of the RUP®, these activities include the identification of a suitable architectural pattern on which to base the software architecture. Once established, (projects in the capstone sequence are normally web-based applications), layers are named, major subsystems (with their interfaces) and packages are identified, and dependencies noted. Essential design principles are rigorously subscribed to in deciding placement of components in layers. Design principles such as divide-and-conquer, coupling, cohesion, reuse, testability, and others provide guidance for design decisions.

For the Model-View-Controller architectural pattern, considerable time is spent on the nature of the layers, their structure, cooperation, design elements within these layers, and dependencies on components within other layers (Figure 4). Notions of subsystems and packages are again stressed, with particular emphasis placed on the subsystem interfaces. Subsystem responsibilities and interfaces are clearly articulated. Accessing the components in packages via their public interface is emphasized to the same extent as subsystems. The notion of a contract and design by contract are again stressed. and included.

Realizing subsystem interfaces (via collaborating components such as other subsystems or packages, or objects inside the realizing subsystem or dependencies on other design artifacts), are captured as much as possible. The critical role of the software architect as opposed to the role of the designer is presented. Once an architectural baseline is established, a large degree of parallel detail design and implementation by team members may ensue.

Students are expected to provide all design models in Rational Rose® (Figure 4) and provide Word® documents containing design decisions as to why specific design elements (subsystems and packages) were placed in specific layers. The architecture is captured in the Rose browser in the Logical View in a package called Layered Architecture.

Deliverable #8, Detailed Design - Use Case Realizations: Deliverable #8 contains two deliverables. Part 1 is the development of the Iteration Plan in sufficient detail to support further detailed design and construction. Part 2 is the development of sequence diagrams (and communications diagrams) for the basic course of events and some additional scenarios in the use cases at the subsystem and package context levels.

As teams progress into the Construction Phase of the RUP, the iteration plan takes on more significance than it had previously. While the nature of iterations and the activities that constitute the initial and earlier iterations were adhered to by the development teams, it is not until this deliverable that a more detailed look at the iteration plan is undertaken and documented. Because a baseline architecture is now established and parallel development can take place within the teams, the specific objectives of an iteration that outlines iteration objectives, required activities, expected artifacts, and criteria for objective assessment become very necessary.

Construction iterations require the team possess a full understanding of the objectives for the current iteration and a 'pretty good' understanding of the objectives of the follow-on iteration. The objectives of this 'next' iteration may be impacted by the assessment of the current iteration, so the next iteration is not 'locked in' until the current iteration is completed and assessed.

The iteration plan to guide Construction is best laid out using a table. It is essential to emphasize that iterations are time-boxed and have definite, measurable objectives that are candidly assessed at the conclusion of the iteration. Because iterations are time-constrained, it is more important to keep on schedule and terminate iterations as in accordance with this schedule even if all objectives of the iteration are not successfully completed. Shortcomings can be rolled into a subsequent iteration. It is more important to keep the development progressing smoothly in an established rhythm than to extend the time of an iteration.

The first iteration is special. The first iterations should address those features or requirements that present the most risk to successful development and (secondly) features addressing core application functionalities. Equivalently, the first iteration should address key features that can cause later breakage. I advise students to address those areas that 'scare' them the most. Thus, I emphasize reducing risk over addressing core functionalities in the first iteration.

Features that represent the essential learning of new tools or the basics of new technologies or learning about a totally new way of doing business likely present risk. Activities that might be present a degree of risk and/or cause concern to team members might include customer authentication, linking up to remote nodes and platforms, establishing 'secure' sessions and communications between different objects some of which might be brokered. Even some mundane tasks such as the ability to accept a sample browser input, communicate this request to an application server, have this server connect to a database server, and then return a response may present considerable risk for students. Activities such as these present risks to the team as a development unit, and these risks must to be mitigated early. Subsequent to the initial iteration, follow-on iterations address requirements whose design and implementation present steadily decreasing risk.

It is essential for early construction iterations to address key core functionalities as captured in specific scenarios. Planning the contents of iterations is critically important in order to track successful development. Without identification of addressing specific scenarios in the objectives of the construction iterations, it is indeed difficult to trace that all required functionality is accommodated. Traceability of design entities and subsequent implementation back to use case specifications is a must (Reed, 2002).

Deliverable 8 also requires the development of one interaction diagram (sequence diagram or communications diagram) for each use case. At a minimum, the basic course of events is modeled illustrating the collabora-

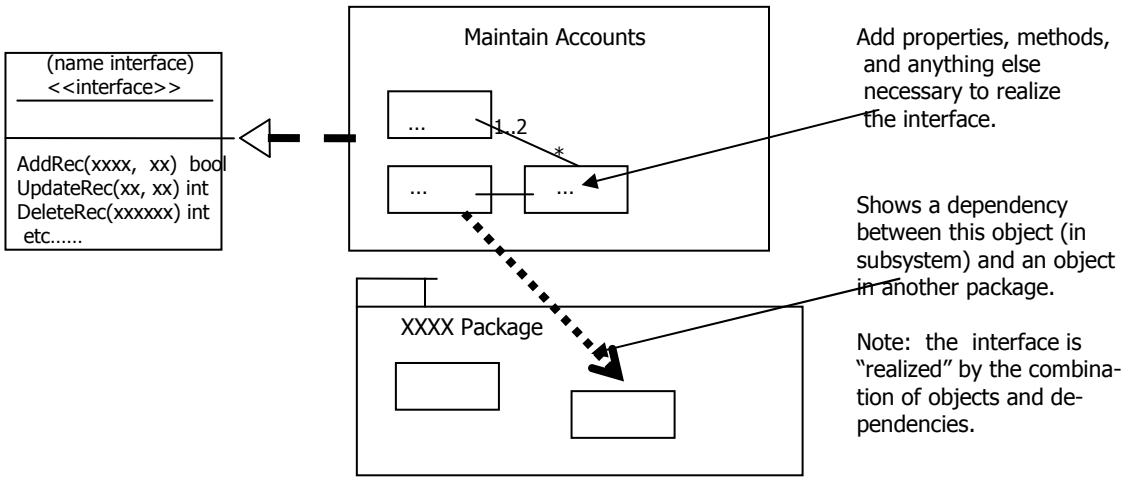

Figure 5. Subsystem Interface and the Realizations – (generic)

tion of design objects required to realize the scenario. In addition to the interaction diagram for each use case, a static (class) diagram, the VOPC, is required that shows the structure of the objects required in the scenario. Their attributes and operations are to be continuously reviewed and modified if necessary.

The required interaction diagrams at this level must show the interacting objects and, for subsystems, their interfaces only. It is important to realize the abstraction of the subsystem at this level and the role it plays in the interaction diagram. The presence of an interface as an object in a sequence diagram is quite a sufficient abstraction at this time. The details of the subsystem design and realization of the interfaces are addressed in Deliverable #9. It is the identification of subsystem responsibilities (and not implementation) that is stressed in Deliverable #8.

The sequence diagrams must be fully annotated with Notes and other design elements as necessary to support Detail Design and implementation. While other scenarios in a Use Case may offer sufficient complexity and require modeling via a sequence diagram and VOPC, this assignment, only specifically required a single scenario (the basic course of events) from each use case be modeled and added to the Rose browser for this application (This will be changed in the future).

Deliverable #9. Detailed Design - Subsystem Design: Deliverable #9 extends the objectives of Deliverable #8 in that each subsystem must now be realized; that is, undergo detail design. Particularly important is the model that indicates exactly how classes or other design entities collaborate in accommodating the responsibilities of the individual subsystems. For example, accessing a relational database system will likely require persistency. So a simple read() method contained in the interface to a subsystem and captured in the context level sequence diagram of Deliverable #8 is 'realized' or elaborated upon by a persistency mechanism that is non-trivial. Collaborating objects designed by the developer coupled with those imported from, say, the java API must be shown, even if the collaborating classes in java.sql are in a different package in a different architectural layer (usually Middleware layer).

There may be a number of subsystems whose interfaces need realization (Figure 5). Further, some of the signatures constituting the interface of a subsystem may not require a persistency mechanism. They may in fact require some computations or some data manipulation or other application-oriented tasks not requiring persistent objects. For a given scenario, Deliverable #9 requires the detailed design of subsystems that were abstracted in Deliverable #8.

Sequence Diagrams from deliverable #8 provide a context in which the subsystems represented only by their interfaces are objects. In Deliverable #9, each subsystem is

modeled by providing a sequence diagram where all entities in the subsystem (or depended upon by the subsystem) that contribute to satisfying a behavior of the subsystem are modeled. These subsystem behaviors may involve a proxy class that represents the subsystem interface used to delegate specific activities within the subsystem itself. These sequence diagrams must be fully annotated and be accompanied with a VOPC. The Subsystem Design is included in the Rose browser in the Use Case Realizations package within the Design Model within the Logical View.

Deliverable #10. Class Design and Implementation 1: Deliverable #10 requires each team to revisit and refine VOPCs for each use case to determine if the connections among objects should be associations or dependencies. As the first iteration in Construction, this deliverable also includes implementing the first and second iterations in accordance with the iteration plan. Detailed assessments of iterations are an absolute and must indicate progress toward completing the application. Feedback from the assessment of iterations 1 and 2 must feed into (rolled into) iteration #3, at which time the goals for iteration #3 can be finalized and its development may start (Deliverable #11).

Part of all the Construction iterations includes the designing, development, and implementation of tests. Objective assessment of the iteration is used to measure not only the quality of the iteration but also the evolving quality of the total application. The assessment of the iteration includes a post-mortem: What went well? What went 'less than well?' What features / objectives of the iteration were not met? Other lessons learned?

Source code components used to support iterations 1 and 2 (fully functional code plus test plans, tests developed, implemented tests and their results) are included in the deliverable. All source code must adhere to standards of good programming. These modules are all linked into the Rose Browser within the Component View.

Deliverable 11. Implementation 2: This deliverable was the final one. Formal demonstrations were presented in class by team members. Black box demonstrations provide validation from an end-user perspective and are based on use case scenarios. An updated iteration plan, and source code, a post mortem document, and the team demonstration constituted the deliverable.

The project post mortem addresses exactly how the use cases drove development, how the architecture was central to design and implementation, and reaction of team experiences to time-boxed iterations.

5. CONCLUSIONS

The two course sequence described has been run three times in the IS program at UNF. At the time of this writing, the sequence is starting again with a few changes discussed ahead.

An informal email survey was recently sent out to students who had completed the capstone sequence most recently (Spring 2005). The survey questions were quite general in nature and were distributed to gain informal feedback and suggestions from those students who wished to provide feedback. Questions were 'open-ended.' This allowed respondents to write as much or as little as they wished. About a fifty percent response rate was realized.

While a few suggestions were offered (and are under consideration), the vast majority of responses were extremely positive and encouraging. Recognizing that the students taking this sequence are not business majors and frequently have a moderate to heavy background in computer course work, the results may not be surprising. These students expect to enter a business enterprise (not a scientific or engineering enterprise) and participate in full-time software development activities. Many will start as entry-level programmers and aspire to more senior positions once they mature in their jobs and learn the enterprise.

Interestingly, every respondent stated that they were very pleased that the emphasis in the course was on process rather than on a specific application developed. Several students stated that while they had been required to learn some elementary program design using pseudo-code or flowcharting, none had an overall appreciation of the range of activities involved in developing a total application – from vision documents through to implementation and assessment. As the RUP® is growing very quickly in

popularity in North Florida, a number of students were quickly employed principally because they had gained familiarity with this process and Rational Rose®. A number had to learn additional technologies, such as Java Server Pages, Java Script, servlets, and other programming technologies that further assisted in their marketability.

A number of students cited that since they anticipated entering the corporate workplace in a programming-related job, additional emphasis on business topics would not have been as valuable as the additional software development experiences gained from the instructional approach undertaken in the capstone sequence.

There were several suggestions for improvement. Although most respondents stated that there was sufficient time for programming, most said that this activity was too hurried. Since all members of the teams did not perform the same duties, a couple of respondents were not satisfied with their own individual contribution to programming. A couple of individuals on teams who were somewhat familiar with some of the programming languages / scripts used in the application development dominated the programming activities. This proved to be a detriment to others who wished to have had more time to learn the technologies. More time was needed here.

One respondent cited that he would have liked to have had formal discussions on interview and questionnaire development.

Most respondents also claimed the amount of work needed was far in excess of what they had been anticipating from those who had taken the capstone sequence using a more traditional approach supported by the Whitten textbook. (It is important to note that the capstone sequence continues to be offered at UNF using the Whitten book. This approach does indeed require the development of a total application. Most of these offerings stress a database approach or information engineering approach to software development, but do not require the RUP.) Having taught the sequence both ways, the workload using the non-RUP approach is not light at all.

While in the distinct minority, a couple of students stated that while they enjoyed the opportunity to learn the technologies, they felt that they missed out on the managerial, end-user, and the cost, budget, planning aspects of a project. These students said that it was not their intention to enter the workplace in a programming role.

In retrospect, there are topics that were presented that need more clarification. Lectures need to be leaner, and self and peer reviews, optional until the end of the course, definitely need to be a key component of each deliverable. Time devoted to testing and the development of test plans was woefully insufficient. Criteria for iteration assessment were not spelled out as well as they should have been. More time to learn additional technologies must be provided, and more time must be allowed for implementation.

It is my firm belief that this robust approach to software development will continue to be very successful. Underpinned by team development using specific, state-of-the-art, highly-marketable technologies, the sequence overall appears to prepare UNF's students well for the constituency we serve. Carefully specified deliverables using an iterative approach based on a base-lined architecture will yield a higher quality product.

6. REFERENCES

Heumann, Jim, "The Five Levels of Requirements Management Maturity," http://www.therationaledge.com/content/feb_03/f_managementMaturity_jh.jsp

Kruchten, Philippe, The Rational Unified Process – An Introduction, 3rd edition, Addison-Wesley, 2004 ISBN:0-321-19770-4

Kulak, Daryl, and Eamonn Guiney, Use Cases – Requirements in Context, 2nd edition, Addison-Wesley, 2004 ISBN: 0-321-15498-3

Leffingwell, Dean and Don Widrig, "The Role of Requirements Traceability in System Development," http://www.therationaledge.com/content/sep_02/m_requirementsTraceability_dl.jsp

Lethbridge, Timothy and Robert Langaniere, Object-Oriented Software Engineering, 2nd edition, McGraw-Hill, 2001 ISBN: 0072834951,www.mcgraw-hill.co.uk/textbooks/lethbridge

Quatrani, Terry, Visually Modeling with Rational Rose 2002 and UML, Addison-Wesley, 3rd edition, 2003 ISBN: 0-201-72932-6

Reed, Paul, "Transitioning from Requirements to Design," http://www.therationaledge.com/contentjun_02/m_requirementsToDesign_pr.jsp

Roggio, Robert, "Establishing Computer User Groups in a Metropolitan Area," Proceedings of ISECON'04, Newport, Rhode Island, Nov 2004.

Whitten, Jeffrey L., Lonnie D. Bentley, Kevin C. Dittman, Systems Analysis and Design Methods, 5th edition, Irwin, McGraw-Hill, 2001.