ISSN: 1545-679X

Information Systems Education Journal

Volume 5, Number 19 http://isedj.org/5/19/ June 5, 2007

**In this issue:**

# Inserting Requirements Traceability into the Capstone Sequence

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

**Abstract:** Computing programs typically have a capstone course or sequence wherein students customarily marshal their academic skills and specify, design and develop a modern, often web-based, application. The process guiding this development may be a light-weight methodology or a heavy-weight methodology – or somewhere in between; the development may be implemented using a traditional, procedural approach or an object-oriented approach. But in any event, stakeholder needs must ultimately be mapped into a requirements specification that supports follow-on design and implementation. This paper provides the motivation, mechanism, and empirical data showing how the infusion of requirements traceability into a capstone sequence is low in cost, high in value, and easy to implement.

**Keywords:** capstone software development; requirements traceability

**Recommended Citation:** Roggio (2007). Inserting Requirements Traceability into the Capstone Sequence. *Information Systems Education Journal, 5* (19). http://isedj.org/5/19/. ISSN: 1545-679X. (Also appears in *The Proceedings of ISECON 2006:* §2152. ISSN: 1542-7382.)

This issue is on the Internet at **http://isedj.org/5/19/**

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)

This paper is part of the group that was selected for inclusion in the journal based on preliminary ratings in the top 30% of papers submitted, and a second review placing it in the top 15% by persons unconnected with the conference or the journal, and whose names are withheld to preserve their anonymity.

# Inserting Requirements Traceability into the Capstone Sequence

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

**ABSTRACT**

Computing programs typically have a capstone course or sequence wherein students customarily marshal their academic skills and specify, design, and develop a modern, often web-based, application. The process guiding this development may be a light-weight methodology or a heavy-weight methodology – or somewhere in between; the development may be implemented using a traditional, procedural approach or an object-oriented approach. But in any event, stakeholder needs must ultimately be mapped into a requirements specification that supports follow-on design and implementation. This paper provides the motivation, mechanism, and empirical data showing how the infusion of requirements traceability into a capstone sequence is low in cost, high in value, and easy to implement.

**Keywords:** capstone software development; requirements traceability

## 1. INTRODUCTION

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 software design practicum or other such designator, whereas within CIS programs, software development is often called Systems Analysis followed by Systems Design and Implementation or perhaps it is called Senior Project 1 and 2. It seems that in CIS programs the sequence is often a two-course undertaking. Often considered the capstone sequence, instructional approaches are many and varied. Regardless of the program or the sequence within which the software project is required, there will be some methodology to guide the development. Typically, the development will have milestones or deliverables that typically student teams must submit for presentation and/or grading. While the approach to the software development process may differ from program to program, one thing remains constant. Customers have requirements, and it is ultimately satisfying those requirements that the application must accommodate.

Of all the reasons attributed to project failure, industry consensus centers on the management of requirements: their accurate capture and modeling, embracing changes to requirements as a fundamental part of a development process, and ensuring that the deployed application does, in fact, satisfy the customer requirements. But in order to be certain that the requirements are satisfied, requirements must undergo tracing; that is, the life of a requirement needs to be traced from its initial form to its ultimate validation in the deployed application.

Great expenditures of time and effort have been made in the name of requirements traceability. In truth, requirements traceability is a controversial subject. From the engineering perspective, it is highly desirable, and tracing requirements assures all customers that the application is indeed addressing the proper needs; it assures project managers that the developers are addressing the right problems and developing the right application; it assists managers in knowing that the time and energies are directed toward satisfying specific needs; and, as change might occur, it assists in aligning the delivered system with current needs, not ones elicited in the past.

From a business perspective, however, the need to trace requirements is often viewed quite differently. From a cost benefit perspective, one must be certain that the costs to trace requirements do result in benefits. Further, tracing requirements can be an exhaustive and expensive undertaking. The benefits must be carefully weighed against the risks of undertaking little, if any, tracing. Clearly, when someone is busy tracing requirements, his/her efforts could be involved in specific design or implementation matters. So the business community faces the complex, multi-faceted issue and must decide if requirements traceability is really worth the effort. At a minimum, requirements traceability must be low in cost, high in value, and easy to implement (Ambler, 1999).

It is interesting to note that as part of contractual arrangements, some companies are required to produce artifacts attesting to requirements traceability. But evidence has disclosed that oftentimes these artifacts amount to little more than square filling and are frequently accomplished with some apathy – primarily to avoid potential litigation that could arise from a failed delivery.

Before traceability can be infused into a capstone sequence, it is essential that a conceptual framework is established.

## 2. TRACEABILITY

Traceability may be defined a number of ways, but in the context of requirements traceability it may be defined as "the ability to describe and follow the life of a requirement, in both a forward and backward direction (i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through periods of ongoing refinement and iteration in any of these phrases; Turbit). So, given this definition, it seems that one might be able to rephrase the definition into something simpler: a requirement is simply "something that a computer application must do for its users. It is a specific function, feature, quality, or principle that the system must provide in order for it to merit its existence" (Kulak, 2004).

### What Does This Mean?

Traceability involves tracking the life of a requirement from its initial inception via a stakeholder through to the ultimate implementation of the application. Traceability requires that developers are able to trace a feature of the application back to its initial source(s). Further, one should be able to trace the requirement both forward and backward and at all places in between the initial capture and its ultimate deployment.

It is important to note also that in the spirit of implementing a real traceability discipline or activity, the traceability activity should take place throughout the entire system life cycle to include maintenance. As emphasized in Leffingwell (2002) in their definition of requirements traceability, "to its subsequent deployment and use, and through all periods of on-going refinement and iteration in any of these phases."

### A Requirements Pecking Order

Dean Leffingwell and Dan Widrig (2002) in their article entitled, "The Role of Requirements Traceability in Software Development," discuss in great detail the development and tracing of requirements artifacts. In applications where requirements traceability is deemed necessary, they suggest a pecking order: Needs → Features → Use-Cases.

Needs: Leffingwell and Widrig claim that those different kinds of projects produce different kinds of requirements artifacts. These artifacts can be organized and managed in a number of ways. But requirements typically originate as statements of needs by a number of stakeholders. These needs are reflected in requirements artifacts and are often quite high level and abstract; these artifacts may also often include text or other information relating to the reason for the need. Further, the needs are not always totally solvable by an automated system. While needs may include features that are amenable to automated solution, they may also include statements of organizational procedure, reasons, goals, or even emotion. Example: "We need to do a better job by providing …." Nevertheless, needs are expressed by those who are often 'paying the bill' for development.

Features: As it turns out, needs, which are a requirements artifact, are filtered into Features, and it is these Features, another requirements artifact, that are amenable to computer-based solutions. Features also usually represent requirements in a little more detail. This 'refinement' of needs to features results in our functional requirements. Unfortunately, these are still often captured in a format that is text-based and characterized by statements such as, "The system shall ...." "The system must produce ...." No illegal data shall be stored." And this horribly boring list continues. Feature descriptions may also be accompanied by mock up screen shots, flowcharts, decision tables, formulas and sample computations, and other mechanisms to communicate requirements. Further, these requirements may be both functional (something the system clearly produces, such as a report, table of data, sounding of an alarm) as well as non-functional ("The system must be extensible, portable, secure, maintainable, scalable, etc." such as "the system must interface with our legacy Billing System").

Use-Cases: Many modern methodologies are now advocating the use of the use-case to capture stories of end-user interactions with the system such that these constitute functional requirements. The development of use-cases is a behavioral approach that consists of both static modeling (classes, objects, etc.) as well as dynamic modeling (sequence and communications diagrams). Use-Cases (functional requirements) and Supplementary Specifications (non-functional requirements) constitute what is often called the Software Requirements Specification (SRS).

The use-case specifications can be used to drive the Design Model and the Implementation Model to ultimate deployment. The use-case specifications also support the development of a prototype, can be used to drive the development of test scenarios, and are used as the basis of iteration planning during Construction as well as a host of other related development activities. Most practitioners would agree that the use-case specification constitutes a requirements artifact that all stakeholders can understand from their individual or role perspectives.

## 3. PRACTICAL REQUIREMENTS TRACEABILITY FOR CAPSTONE COURSES

There are many tools available to support requirements traceability (Gotel). Many of these are quite specialized and require complex environments. A number are quite costly. For safety-critical systems, life-dependency systems and hosts of other applications whose reliability must be absolutely certain, it is difficult to understand how such applications might be developed without significant traceability mechanisms in place as an integral component of their development environment. It appears, however, that for smaller systems (smaller is arbitrarily defined by number of use cases fewer than twenty), a simple traceability matrix approach may be used.

### Volunteers in Medicine (VIM)

The matrix traceability approach was used for the Volunteers in Medicine (http://www.vim-jax.org) Project undertaken by two groups of software development students at The University of North Florida from August 2005 through April 2006. The teams were formed and charged to specify, design, and implement a new medical information system for VIM, an organization in Jacksonville that provides pro bono medical services to the working uninsured. Almost everyone associated with the VIM is a volunteer (there are four paid employees). They have doctors, nurses, nurse practitioners, and others who give freely and generously of their time. Much of their funding is from philanthropy and state or government grants based on usage statistics based on office visits, patient needs, appointments, laboratory equipment needs, etc. The VIM application system was developed to manage all volunteer information, patient information, and provider information by providing a very user-friendly, non-intimidating, learnable interface designed to support various VIM needs.

Needs: Using the approach of identifying and capturing Needs expressed by a variety of stakeholders (business manger, volunteer coordinators, and office workers), a matrix of needs was produced by the development teams. Interviewing the individual stakeholders and documenting these real needs resulted in the matrix shown in Figure 1 (see

Appendix). These eight broad needs seemed to reflect the most significant VIM priorities.

Features: As most 'needs' are enunciated, they often contain content that does not directly translate into features that can be accommodated by a computer application. Yet, the real needs that must be accommodated by an application are "embedded" in these statements. So, subsequent to building a Needs Matrix, needs were carefully analyzed and mapped into features amenable to automated solution. The Features represent the true functional requirements, albeit abstract. The mapping was developed starting with each need and gleaning the specific features. To be certain that each need mapped to features and that each feature was traceable to a specific need, both forward and backward traceability matrices were developed as shown in Figure 2 and Figure 3.

Developing these traceability matrices was an extremely beneficial exercise for the students in so many ways. First of all, after stakeholder acceptance of these Features, it assured the teams that their efforts were directed to satisfying a discrete, finite number of features all bought-into by the stakeholders. It also helped to assure students that time expended in accommodating these features was directed at real needs and not frills or "nice-to-haves." Development of these matrices also assured the stakeholders that the development teams had indeed captured, documented, and understood the required features. From a project management perspective, the matrices provided a mechanism to track project progress and plan iterations, and the many details therein, such as individual team member responsibilities.

Use-Cases: Features, captured in a variety of formats, are universally-accepted requirements artifacts and have been so for many years. Typically, however, they often represent long lists of "correct" (we hope) yet boring requirement details. As an alternative requirement artifact, the use-case provides stories of user-interactions with an application, thus communicating requirements in a mode much more understandable to both customers and developers alike. An index of use-cases developed for the VIM project is provided in Figure 4.

Using the approach advocated by Leffingwell and Widrig (2002), features were mapped into Feature to Use-Case Traceability Matrix and Use-Case to Feature Backward Traceability Matrix (Figures 5 and 6). Here again, it was an imperative to ensure all features were indeed captured in some use-case, and that each use-case traced back to one or more features. Since the mapping is essentially from one requirements artifact to another, care had to be taken so that no requirements were lost or diluted.

### Beyond Requirements Artifacts

The VIM project continued using the traceability matrix approach to ensuring all requirements were captured and traceable back to features and needs. But the approach was also continued forward into the development of an Analysis Model, where each use-case was mapped into a series of analysis classes: boundary, control, and entity classes. This approach showed the structural relationship among the analysis classes working together to provide required relationships. Interaction diagrams (sequence diagrams) using analysis classes were developed to see the behavioral relationships.

Tracing the use-cases into use-case realizations is considerably beyond the scope of this paper, but there are a number of excellent papers that address these and related issues from a number of practical perspectives. While some provide an approach to tracing requirements into design such as by using collaborations of features (e.g., Leffingwell, 2002), others imply that this approach may be intractable. Regardless, more sophisticated tool support would be needed whatever approach is taken.

Traceability matrices can be readily developed to trace the different use-case scenarios to specific test cases. The interested reader is referred to Leffingwell (2002).

## 4. CONCLUSIONS: TRACEABILITY IN CAPSTONE COURSES – THE GOOD, THE BAD, AND THE UGLY

There is no doubt that the efforts undertaken by students in tracing requirements through a series of requirements artifacts were very definitely worthwhile. Each deliverable (there were eleven deliverables spread over two semesters) had a traceability component. The cost was minimal in that no specific commercially available tools were used. Further, because the application developed was small in the number of use-cases, the matrix approach to requirements traceability was a viable approach.

Once analysis classes were developed (not shown in this paper but shown in Roggio (2006), continuing the traceability approach using matrices into design did, in fact, become terribly cumbersome, even for this small application. To develop use-case realizations for each scenario in each use-case required the development of a number of design classes. Further, the user interface and the control functions became complex. While one of the teams did attempt to develop a traceability map from use-cases to design classes, the number of design classes was very large in number. As valuable as the traceability matrix approach was for the specification and analysis of the application, using the traceability matrix approach into design was abandoned and not readdressed further. While this change in strategy was justifiable for design, departing from the traceability matrix approach this was a mistake, as none of us had the foresight to see its value in tracing use-case scenarios to specific test cases. Use of traceability matrices in testing would have benefited this effort by providing evidence that all scenarios underwent some degree of coverage testing. Hind sight is always 20-20 it seems.

It is important to have everyone "on board" if the team is to undertake viable requirements traceability. The importance of requirements traceability was clearly articulated from the beginning of the project. In discussions addressing the best practices of software engineering, the management of requirements and the failure to embrace changing requirements receive top billing. The advantages of an iterative development process, the notion of time-boxed iterations, and several other development fundamentals were stressed prior to even embarking on the project. Not impaired by experience, the students did not question the cost effectiveness of the traceability exercises and readily saw the value as they realized that the costs were little and the benefits were great. The average total time spent on traceability, as estimated by the teams, was about twenty hours over two semesters. The strategy was clear: traceability was integral to the process, and the understanding of its importance was shared equally by all team members.

Despite many advances in the area of requirements traceability, failure to trace requirements remains a serious problem today. Many feel that infusing a traceability culture into the organization helps to mature the organization with the concomitant improvement in productivity. Tracing the life of a requirement may be an onerous undertaking for some applications, and there is no single solution to this noble goal. According to Scott Ambler (1999), traceability is difficult, but a mature approach to requirements traceability may be the difference between organizations that are successful at developing software and those that are not.

## 5. REFERENCES

Ambler, Scott, (1999) "Tracing Your Design," April, www.sdmagazine.com/documents.

Gotel, Orlena C. Z. and Anthony C.W. Finkelstein, "An Analysis of the Requirements Traceability Problem" oczg; acwf @doc.ic.ac.uk.

Kulak, Daryl and Eamonn Guiney, Use Cases – Requirements in Context, Addison-Wesley, Second Edition, 2004. ISBN 0-321-15498-3.

Leffingwell, Dean, Don Widrig, (2002) "The Role of Requirements Traceability in Software Development," http://www.therationaledge.com/content/sep_02/m_requirementsTraceability_dl.jsp.

Roggio, Robert F., (2006) "Front End Requirements Traceability for Small Systems," Pacific Northwest Software Quality Conference (accepted).

Turbit, Neville, "Requirements Traceability," The Project Perfect White Paper Collection, http://www.projectperfect.com.au

## APPENDIX

| ID | NEED |
|---|---|
| N1 | Record Repository |
| N2 | Scheduling |
| N3 | Tracking and Report |
| N4 | Remote Access |
| N5 | Provider Access to Patient Records |
| N6 | Mass Communication |
| N7 | Central Control for Setting Up Authorized Access |
| N8 | Protection Against Unauthorized Access |

**Figure 1. Needs List**

| ID | NEED | FORWARD TRACEABILITY |
|---|---|---|
| N1 | Record Repository | F1, F2, F5, F32 |
| N2 | Scheduling | F6, F7, F9, F11, F12, F13, F31 |
| N3 | Tracking and Report | F14, F15, F16, F17, F18, F20, F22, F32 |
| N4 | Remote Access | F23 |
| N5 | Provider Access to Patient Records | F24, F25 |
| N6 | Mass Communication | F26 |
| N7 | Central Control for Setting Up Authorized Access | F33 |
| N8 | Protection Against Unauthorized Access | F29, F30 |

**Figure 2. Forward Traceability Matrix: Needs to Features (F = 'Feature')**

| ID | FEATURE | BACKWARD TRACEABILITY |
|---|---|---|
| F1 | The system will allow the user to add, delete, and update patient records | N1 |
| F2 | The system will allow the user to add, delete, and update volunteer records | N1 |
| F5 | The system shall enable approval of personal profile changes by the business administrator of volunteer coordinator | N1 |
| F6 | The system shall enable the insertion and deletion of volunteers into the schedule | N2 |
| F7 | The system shall enable the insertion of deletion of providers into the schedule | N2 |
| F9 | The system shall enable the insertion of patient qualifying and examination appointments into the work schedule | N2 |
| F11 | From the schedule, the user will be able to access information about a particular appointment. This in | N2 |
| F12 | The system shall allow the rescheduling of appointments missed by a patient | N2 |
| … | … | … |
| F29 | The system shall authenticate users for certain parts of the application based on user ID | N8 |
| F30 | The system shall not allow users to access system resources until the user has "clocked" in. | N8 |
| F31 | The system shall enable the insertion of patient qualifying appointments into the work schedule | N2 |
| F32 | Maintain chronic medical conditions treatment. | N1, N3 |
| F33 | The system will allow an authorized user system administration capabilities such as creating and deleting users and reset forgotten user passwords. | N7 |

**Figure 3. Backward Traceability Matrix: Features to Needs**

| USE CASE NO: | TITLE |
|---|---|
| UC-01 | Schedule Patients |
| UC-02 | Schedule Volunteers |
| UC-03 | Schedule Qualification Consultation |
| UC-04 | Maintain Patient Records |
| UC-05 | Maintain Volunteer Records |
| UC-06 | Research Statistics |
| UC-07 | Send Group Email |
| UC-08 | Authorize User |
| UC-09 | Perform Administrative Tasks |

**Figure 4. Use-Case Index – VIM Project**

| ID | FEATURE | FORWARD TRACEABILITY |
|---|---|---|
| F1 | The system will allow the user to add, delete, and update patient records | UC-04 |
| F2 | The system will allow the user to add, delete, and update volunteer records | UC-05 |
| F5 | The system shall enable approval of personal profile changes by the business administrator of volunteer coordinator | UC-04, UC-05 |
| F6 | The system shall enable the insertion or deletion of volunteers into the schedule | UC-02 |
| F7 | The system shall enable the insertion or deletion or providers into the schedule | UC-02 |
| F9 | The system shall enable the insertion of patient examination appointments into the work schedule | UC-01 |
| … | … | … |
| F30 | The system shall not allow users to access system resources until the user has "clocked" in. | UC-08 |
| F31 | The system shall enable the insertion of patient qualifying appointments into the work schedule | UC-03 |
| F32 | Maintain chronic medical conditions treatment. | UC-04 |

**Figure 5. Traceability Matrix: Feature to Use-Case – VIM Project**

| ID | USE CASE | BACKWARD TRACEABILITY |
|---|---|---|
| UC-01 | This use case is started by the Clinical Director or a Volunteer and it allows them to schedule patient appointments. | F9, F11, F12 |
| UC-02 | This use case is started by the Clinical Director or Volunteer Coordinator to schedule volunteer work hours. | F6, F7 |
| UC-03 | This use case is started by the Clinical Director or a Volunteer to schedule a qualification consultation appointment. | F31 |
| UC-04 | This use case is started by either, the Office Coordinator, Volunteer Coordinator or a Volunteer to create, update, and delete patient records. | F1, F5, F24, F25, F32 |
| UC-05 | This use case is started by the Volunteer Coordinator and it allows her to edit volunteer records. | F2, F5 |
| UC-06 | This use case is triggered by the Business Administrator to look up, compile and print out organizational statistics. | F13, F14, F15, F16, F17, F18, F19, F20 |
| UC-07 | This use case is started by the Business Administrator and it allows him to mass email all the volunteers in the database. | F26 |
| UC-08 | This use case is started by any user of the system and it allows them to gain access on site or remotely, and check authentication. | F29, F30 |
| UC-09 | This use case is started by the Business Administrator and it allows him to create and delete users, and to reset user passwords. | F33 |

**Figure 6. Traceability Matrix: Use-Case to Feature – VIM Project**