A WEB DEVELOPMENT FRAMEWORK FOR TEAM-BASED PROJECT COURSES

J. Patrick Fulton, University of Louisville, jpatrick.fulton@gmail.com
Ted J. Strickland Jr., University of Louisville, ted.strickland@louisville.edu

ABSTRACT

The complexities of team-based web development make it challenging for student teams to deliver solutions that may be sustained over multiple semesters. A web development framework based on open source tools and Agile practices may be combined to create a standardized development environment. Use of the framework may provide students with rich system development experiences and deep knowledge of software design principles and patterns. Interactive features may be developed each semester for inclusion in an evolving web application used by nonprofit organizations.

Keywords: Web development framework, CIS course projects.

INTRODUCTION

Team-based software development presents a variety of challenges that must be addressed by introducing both methodology and technology into the software development process. These challenges are compounded when team-based software development is undertaken in a classroom setting. The constraints of time, development experience, familiarity with development technologies, and leadership practices found in classroom settings create special concerns.

In our senior-level software development course (CIS 420), which is comparable to the Physical Design and Implementation in Emerging Environments course (IS 2002.9) of the IS 2002 Model Curriculum [9], students participate in a team-based development project. The product of their efforts, a software application that was developed, and is expanded upon across multiple semesters, is a content management system with features that support online interactions. The application is designed for use by nonprofit organizations. The system provides a mix of common solutions [14] that address processes common to nonprofit organizations; e.g., donation management, volunteer monitoring, an online store, email communications, and news article management. At the end of each semester, teaching assistants perform an evaluation of the code produced by students and prepare a release. This release is then deployed to a production environment. By the end of the Spring 2009 semester, 12 organizations have participated in some stage of system evaluation or are using the system.

The development of a production system based upon the efforts of students presents a series of special challenges. Sustainability of the solution becomes a paramount objective. Students must follow practices that allow their contributions to be continued easily by students in future semesters. The quality of their output must be very high. Few software defects are tolerable and issues that result in production downtime become particularly undesirable. By bringing production requirements into a classroom setting, the course design has made it possible to expose students to experiences and practices that they will likely encounter in employment settings.

The first obstacle is related to configuration or change management. Each developer must be able to access a current version of the system under development in order to perform his work. Similarly, his changes must become available to other developers of the system and conflicts created by altering the same set of files must be resolved. Without a change management system or central repository of code in place, developers have no guarantee that their changes are relevant to the most current version of the system. To avoid duplication of labor and wasted effort reconciling changes made by each developer, a change management, or version control, system must be put in place.“Cost isn't an issue as good quality open-source tools are available” [6].

A single, shared location for the storage of system code is a necessary step. However, ensuring that all code exists in a single accessible location does not ensure that the code placed in that repository works together. As multiple developers work in parallel, they must ensure that their collective work product may be integrated together to form a single working system. In many development teams, subgroups or feature teams are formed to work together on closely related sets of features. These subgroups are tasked with integrating their changes on a periodic basis. Often, this process is undertaken at the end of the week. However, as system complexity increases, so does the amount of time needed to integrate the changes made by each subgroup. The process of integration can quickly overtake the development of new code. Ideally, developers continuously integrate their changes into the system as a whole. At a minimum, a continuous integration process should ensure that all code committed to the source code

repository should compile correctly. A more robust approach executes a series of automated tests against the system's source code base. In addition to ensuring the system builds, a continuous integration process should ensure that the code committed to the repository executes successfully against the test suite.

Most modern system design projects will experience an inevitable set of changes of system requirements as incremental versions of the software are presented to end users and project stakeholders. Rigid or inflexible system designs degrade in the face of these changing requirements. "If our designs are failing owing to the constant rain of changing requirements, it is our designs and practices that are at fault." [11, pg. 107]. Software must be developed with design agility in mind in order to respond to an environment of change. Design with agility in mind typically means the application of agile design principles and design patterns. However, these topics are often beyond the scope of an undergraduate course in software development.

Individual programmers develop a personal coding style. However, these styles often carry their own idiosyncrasies and flaws. Additionally, a system that looks like it was developed by thirty separate people, each with their own approach to formatting and naming conventions, is difficult to maintain. A development team must create and enforce a single unified approach to programming style in order to enhance the maintainability of the system and the team's ability to work together.This unified approach much be communicated in the form of examples and documents. A single coding standard must be developed and implemented in order to ensure the maintainability of a system. Many open source projects have taken this approach to formalizing standards for coding style and organization, including the Apache Jakarta Project [2, 17].

Generally, development teams are composed of individuals having varying degrees of experience with the technologies used in the project. However, in a classroom setting, it can be expected that most, if not all, students have little experience with what are often complicated development tools. Modern programming languages and technologies allow the developer to select from a wide variety of approaches to solve a problem. Unfortunately, of the many approaches that may be used to solve a problem, only a few may conform to best practices. Further, best practices are frequently learned through experience. To create a production quality system, student contributions must conform to best practices. As a result, a simplified web development technology is required. An ideal technology limits the number of choices that may be made by developers and it naturally pushes developers towards best practices.

CREATION OF A WEB DEVELOPMENT FRAMEWORK

Two primary tasks were required to create a sustainable development methodology for CIS 420. The first one was to assemble open source development tools and to adopt closely related practices. The second one was to build a custom software development framework. The framework, designated Carnegie, was designed and developed by teaching assistants during the summer sessions of 2007.

Development Environment and Practices

The development environment consists of a shared hosting and testing environment for the system. The environment, shown in Figure 1, includes Microsoft's IIS web server on the application server, Microsoft's SQL Server on the database server, and a Linux-based firewall server. While students are not offered direct access to the web server, security restricted accounts are made available for students to interact with the database server; they are able to design database structures and directly manipulate data stored within the tables for testing purposes.

The first major component of the development environment is a code versioning and change management tool. Subversion was selected for its stability, usability, and status as de facto industry standard, replacing CVS for most modern open source projects [16]. The Subversion code repository provides a single, centralized location for all system source code. The repository is accessible over the Internet, making it possible to quickly and easily download current versions of the source code. Any lab machine or home computer with the Visual Studio C# compiler installed can be configured as a workstation for the project. Students use the repository to keep their working copies of the solution current with the changes made by other members of the class. The instructor and teaching assistants use the repository to review source code and monitor student participation in the development

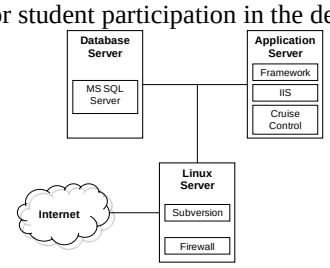

Figure 1. Development Environment

processes. Only code that has been submitted to the repository is evaluated by the instructor.

The second major component in the development environment is a continuous integration server. CruiseControl.NET [5] was selected to fulfill this need. The CruiseControl.NET server is a highly configurable and powerful tool. Its uses can be varied. However, CruiseControl.NET was customized using a series of build scripts to facilitate the processes used by the project. The environment maintains a constant monitor over the status of the version control tool. When a change of the source code repository is detected, the CruiseControl.NET server downloads an updated version of the project. First, this new version is compiled on the application server. Next, all unit tests within the project are executed against the fresh build. Should both of these steps succeed, the updated version of the application is auto-deployed to the web server, making the updated version of the application publicly accessible to all members of the team.

The primary advantages of the continuous integration server are its ability to detect failures to integrate changes successfully into the application and its ability to notify members of the team when these failures occur. For example, a student may commit code to the repository that does not compile. When this problem is detected, all connected members of the team are notified. A web page detailing the nature of the error is created and made available through the web interface to the continuous integration server. Similarly, a code refactoring which breaks the unit test suite might be committed. An error report would be created automatically and made available through the web interface of CruiseControl.NET.

The auto-deployment features of the continuous integration server prove especially useful in a classroom setting. Since new versions of the application are deployed automatically to the web server, there is no need to offer students administrative access to the web server. A single configuration for the web server can be created and all files can be copied into the deployment directories by the integration server. This ensures that the most current version of the application is always the one in deployment. The completion of the testing and auto-deployment process takes approximately 90 seconds. All in-class demonstrations of the application are performed against the auto-deployed version.

Students use these development environment tools in a repeatable procedure as they develop the application. They used the following procedure several times per development cycle:

1. Download a current version of the project files from Subversion.
2. Perform changes to the source code.
3. Compile locally.
4. Run an update to integrate any changes that were made by other students while steps 2 and 3 were being performed.
5. Compile locally to integrate changes.
6. Commit changed, added, and deleted files to the current version of the application stored in Subversion.
7. Verify that the committed changes pass all tests performed by CruiseControl.NET.
8. View the new version of the application on the development web server to verify that changes have not created any runtime errors that could not be caught by the tests performed by CruiseControl.NET.

In addition to these development environment tools, a series of written coding standards were produced. These standards describe detailed practices, techniques, and naming conventions for use when coding the system. Students are provided these standards early in the semester and are given the opportunity to review them and to ask questions. A component in the evaluation of student work is conformance with the written standards. Students are expected to ensure that their code meets these standards and resembles existing code within the system.

Development Framework

The Carnegie framework is a custom framework for web development that offers an alternative to ASP.NET.

MVC Design Pattern – The Model-View-Controller (MVC) design pattern is used often to represent system architectures [7], including web development frameworks. A summary of popular server-based web development frameworks, especially the view and controller elements of MVC, is provided by Vosloo [19]. Web development frameworks based on hybrids of the MVC design pattern have been developed [3, 10], some using C# and Visual Studio.NET [3, 13]. In addition, the MVC pattern has been used as the basis for designing semantic web frameworks [4, 13].

The Carnegie framework is a MVC architecture, as shown in Figure 2. The framework is a series of discrete components divided into layers. Each component is responsible for a series of separate concerns that allow elements of the application to be

isolated and decoupled from one another during implementation [3]. The framework can be seen as being divided into three types of components: model components, controller components and view components. Model components represent information about the domain and its business logic elements. View components represent objects that are responsible for rendering the presentation layer in a markup language such as HTML. Controller components are responsible to handling input to the application, manipulating the model, and selecting the view components that are necessary to render the results of the request. Some components at each level are provided by the framework. Other components at each level must be implemented by the student to fulfill an assigned development task.

A Simplified Environment – ASP.NET is a powerful, but complicated, technology. Our experience suggests that it is too complicated to use in a senior-level course without significant programming experience, which often results from co-operative education assignments or employment in the industry. While it is certainly possible to develop ASP.NET applications using an MVC approach, it is difficult and it requires an understanding of the subtleties of the technology. An effective implementation of MVC using ASP.NET requires an understanding of the ASP page lifecycle, dynamic

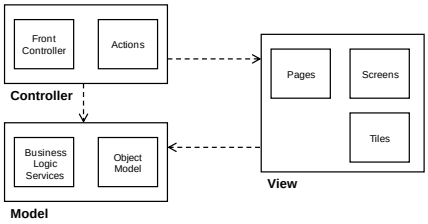

Figure 2. Framework Viewed from an MVC Perspective

control creation, and the use of ObjectDataSource. Many of these topics are beyond the scope of programming courses of the typical IS curriculum.

The primary purpose of the framework is to constrain the number of options available to developers. When viewed at a high level, many development tasks within the framework are well defined and formulaic. As features within the framework have matured, suitable examples of coding practices have emerged, providing several references for students. The highly structured nature of code written within the framework makes it possible to quickly convey best practices and to form easy-to-follow standards for development. The development of applications based on the framework does not require an advanced understanding of the .NET language features, .NET APIs, or the more complicated web development technologies available for the .NET platform. By developing within the framework, students are able to engage in team-based software development of complex features while producing high-quality code.

Four Layer Architecture – The framework is organized into four primary architectural layers, as shown in Figure 3. Code placed in each layer is given a defined set of responsibilities. By separating the system into layers, reusability can be significantly enhanced. Additionally, a layered architecture simplifies the responsibilities of the classes that compose the system.

The framework enforces the use of an n-tier architecture composed of a presentation layer, a business logic layer, a data access layer, and a database persistence layer. Each layer may remain relatively independent of the other layers and can be replaced with alternate versions. For example, the independence of the database layer makes it possible to use any supported relational database. Microsoft's SQL Server can easily be replaced with an Oracle database server.

Communication between layers in the system takes place laterally. Layers may only interact directly with layers that are adjacent to it. All communication begins with the presentation layer. A request is made by the user through a web browser to the presentation layer. The request is then marshaled to other layers as needed to fulfill the request. It is not necessary for every layer to be involved in the fulfillment of a request. For example, a user may request a page that contains only static content and does not require the application's state to change or a query to be made. In this case, the request can be completed without the involvement of any layer other than the presentation layer. The presentation layer of the application would simply locate the template associated with a static content page and render its output. Since no data or business logic would be required to complete the rendering of the page, no other layers would need to be involved.

Presentation Layer – Objects within the presentation layer are responsible for interacting with the user. They render screens and pages and produce the user interface. Additionally, the presentation layer is responsible for intercepting and dispatching requests made by the user to other layers within the system. While the framework's current design supports a user interface rendered in HTML, the presentation layer could be replaced or supplemented by additional presentation layers that support other user interface formats. A Windows forms application could provide an alternate presentation layer for an

application that made requests for business logic using web services or a RPC technology such as .NET Remoting. Similarly, a set of presentation layer code that rendered a VRML interface to provide a three-dimensional front end to the application could be developed. Only changes to the presentation layer of the application would be required.

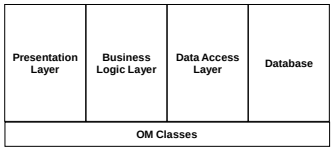

Figure 3. Four Layer Architecture and Object Model Classes

Business Logic Layer – Classes within the business logic layer provide customized queries for data. Also, the business logic classes are responsible for all changes to the application's state. For example, a presentation layer component might be designed to display a list of customers. A business logic component would provide a method that returned a series of customer objects acquired from the data access layer. The customer objects would then be sent to the presentation layer where they would be rendered to the user's browser. When the user performs an action that changes the persistent state of the application, the business logic layer must provide an object with a method that is responsible for implementing the change. In an e-commerce application developed within the framework, the user adds products to a shopping cart stored in the session and eventually checks out to complete an order. When the user places an order, records must be stored in the database to represent the contents of the order. The business logic layer would provide a method that performs this action through an interaction with the data access layer.

Splitting business logic into dedicated classes simplifies the reuse of code. Many modules within a complex system deliver services that are useful to other modules within the system. For example, a system might include a mass email management tool that includes logic capable of sending automated emails. Other modules within the system are likely to need to send emails to users during their own workflows. An e-commerce module might need to send an email to the customer to confirm the contents of a recent order. By separating the logic responsible for emailing into a business logic service, it is possible for the e-commerce module to easily consume and reuse the email logic that has been placed in the email business logic components.

Data Access Layer – Unlike some three-tiered application architectures where direct queries to the database are made by the business logic layer, the Carnegie framework introduces a data access layer and support for a series of object model classes that represent an object-oriented view of data from a relational database perspective. The data access layer is responsible for managing these object model classes and for providing a generic query mechanism that is independent of the platform used in the database layer. Traditionally, the business logic layer would interact directly with the database platform using an ADO or JDBC API. In Carnegie, all direct communication with the database is performed by the data access layer. This layer also introduces an object-oriented paradigm for data access. Instead of interacting with multidimensional arrays of data returned by a database access interface, records from the database are represented by simple data transfer objects. Queries are not prepared in a series of static, hard-coded SQL strings. Instead, a series of object-oriented tools are provided by this layer, including a query-by-example tool and a criterion-based query API. Once object model classes are instantiated and populated by this layer, they may be passed and used in any layer of the framework. This characteristic makes object model classes unique and independent of any specific layer.

Database Layer – The database layer is responsible for the physical storage of data within the system. The framework currently supports a relational database as its data layer. The use of the NHibernate [12] library in the data access layer makes it possible to easily switch from database platform to database platform without regard for the subtle differences in dialect among platforms. While a single relational database is used for persistent storage in this framework, this layer could easily be expanded to support data from more than one relational source or to include data from a legacy system.

OM Classes – While most classes within the framework belong to a single layer, object model (OM) classes are used across multiple layers. The object model represents data and rules from the domain. These classes are mapped by the database access layer to tables within the database layer. An OM class represents a single record from the database and models an entity within the problem domain. For example, OM classes exist to represent users, emails messages, and donations. These classes are used by the presentation layer to display data. In the business logic layer, queries for these objects are performed and methods exist to modify their contents. The database access layer is responsible for fulfilling queries made by the business logic layer with result sets composed of OM classes. At the

database layer level, these classes are persisted as records in physical tables.

Design Patterns – The Carnegie framework is a Front Controller [7] architecture. It employs a number of design patterns in its design including Command [8], Service Locator [1], Intercepting Filter [1], Composite View [1] and Template View [7]. The use of these design patterns in the design of the framework not only allows the introduction of these concepts into the course but also allows the framework to remain flexible enough to accommodate the design requirements of a large number of web applications. While the framework is currently being used in a single application, its design is independent of the applications that are built using it.

From the perspective of a student developer using the framework to develop features for the common solutions application, the framework is composed of a series of interfaces and reusable base classes. When implemented and extended, these classes and interfaces form components that interact with the underlying architecture. The .NET language feature reflection [18] is used by the framework to load classes based on these reusable base classes and implementers of specific interfaces into the architecture and to deploy them to their appropriate layers within the framework. Implementations of the Service Locator pattern are provided by the framework and allow objects to locate other objects on which they are dependent. The complexities of request parsing, object location, object deployment into architectural layers and control over the request lifecycle are hidden from the developer.

USE OF THE FRAMEWORK

The framework was introduced in CIS 420 during Fall 2007. During each of the three successive semesters, course adjustments were made based on the instructor's and student assistants' impressions of framework use in achieving course objectives. Thus, the effectiveness of the framework as a means for enhancing student learning and of building a higher quality system is limited to qualitative judgments.

Substantial course adjustments occurred between academic years. For purposes of comparison, observations corresponding to academic year 2007-2008 were combined to form the Initial group. Observations of course sections taught in academic year 2008-2009 comprised the Revised group. The number of students in each group was comparable, with 37 students participating in the Initial group and 36 students participating in the Revised group.

Development Process

Use of the Carnegie framework provided students in both groups with exposure to modern software development practices and tools. The Agile-like development methodology included individual and/or team-based assignments, short iterations (weekly, 10 per semester), rapid feedback cycles, and voluntary use of pair programming. Practices involved strict adherence to programming standards defined to ensure maintainability of the solution. Advanced C# programming techniques included use of deep inheritance trees, programming to interfaces, use of generics, and creation of custom exceptions. All students gained experience with Subversion's code versioning tool and Cruise Control's integration tool. The framework provided a realistic simulation of a team-based development environment.

Both groups developed interactive web processes, such as site registration, online donation, a calendar of events, event registration, and e-mail communications. Feature development required that students implement design decisions across all layers of the architecture:

1. Table structure, field content, and table relationships in the database layer;
2. Table mappings to objects in the data access layer;
3. Service classes in the business logic layer;
4. HTML templates and associated C# classes in the presentation layer;
5. Action classes as elements of the front controller.

Process Changes

Aside from introduction of the Carnegie framework, students in the Initial group completed the learning activities expected of the previous versions of the course [15]. These activities emphasized interpersonal skills as well as technical skills. Activities included oral presentations during design reviews and submission of written deliverables; e.g., design models and progress reports. Also, team activities were expected, where students were assigned to either feature teams, client teams, or both.

Difficulties were observed with the Initial group's use of the framework. Feature teams struggled with the identification of appropriate development tasks (user stories) and their dependencies. Weekly design reviews often required revisions (revisits) of code, and several new user stories were identified during the reviews. Students struggled with the complexities of development using the four-layer architecture, despite attempts to explain programming techniques during design reviews, class sessions, and help sessions with the teaching assistants. Overall, software quality was lower and development speed was slower than expected.

Review of these issues led to two significant course changes for the Revised group: (1) feature development was guided by the teaching assistants, who wrote user stories and assigned them to students; and (2) technical skills were emphasized over interpersonal skills. Teaching assistants were appointed based on their demonstrated mastery of the framework, and they took on the role of lead programmers, assisting students in learning to use the framework. Identification of user stories and their dependencies shifted from a peer-based activity to supervisor-subordinate activity. Also, formal design reviews were eliminated, as each student (or student pair) conducted an informal code review with a teaching assistant each week. With an Agile-like methodology, the need for formal design models was minimized, and the production code stored in the Subversion repository became the primary form of system documentation.

With the Revised group, the teaching assistants created user stories of increasing complexity during the semester. For example, early in the semester stories involved enhancements of the user interface, requiring only modification of the presentation layer. Stories of intermediate complexity involved modification of database tables, mapping files, and object model classes. By the end of the semester, complex stories required modifications across all four layers, including creation of action classes for the front controller. User stories were assigned so that most of the students obtained thorough exposure to framework conventions for each layer. A small proportion of students were allowed to focus on specialties, such as user interface (presentation layer) or database operations (database layer and data access layer).

Also, the evaluation process for the Revised group included a greater emphasis on code quality. In addition to evaluating the user story's function, scores were assigned for coding style; i.e., easy to read and easy to change [11, pg. 42]. The grading scheme favored the student who produced high quality code for the assigned user story over the student who accepted additional user stories, but produced lower quality code.

In reviewing the impacts of the process changes, we concluded that the contributions of the Revised group surpassed those of the Initial group in the following areas:

1. Students took on more challenging user stories (guided by the teaching assistants);
2. User story revisits focused primarily on code quality, not on proper function;
3. A larger number of the features were completed each semester;
4. A smaller percentage of the code required re-work before transition to production.

DISCUSSION

Introduction of the framework as the development tool in the senior level IS implementation course elevated a curriculum question that went unspoken – where is the balancing point between breadth and depth of development exposure?

The initial version of the course exposed students to a broader set of experiences, combining interpersonal and technical assignments in an attempt to simulate the full set of system development tasks. However, this broad approach did not focus class sessions and student effort on mastering the technical details of the framework; i.e., the quality of the solution did not meet production standards. In the revised version, a higher emphasis was placed on understanding the architectural design and programming practices, and students produced higher quality code. However, they did not get the broader exposure to the interpersonal aspects of system development. The tradeoff is evident – what depth of programming exposure should be provided in a single senior-level course?

When addressing this question, several other questions should be considered:

1. How many courses require that students participate in team projects?
2. How many courses require oral presentations?
3. How may courses require significant written assignments?
4. Do other IS courses provide sufficient coverage of these interpersonal skills?

Also, consider the plight of the aspiring systems analyst in his first full-time position following graduation. As part of a major software development project, he coordinates analysis and design activities with an out-sourced (possibly off-shored) development firm. Is his development knowledge sufficient:

1. To understand the architectural tradeoffs facing the programming team?
2. To appreciate schedule and cost impacts of options presented by the programming team?
3. To assist the team in making architectural and implementation decisions needed to deliver the proposed system on schedule and within budget?

Additionally, our observations caused us to reflect on the connections between introductory programming courses and the senior level IS implementation

course. Many of the assignments in the introductory course are in the form of well-structured problems constructed by the instructor. Typically, they relate to course material that was discussed in class recently. In this context, it is likely that students apply the programming language and tools to create a functional solution for the assignment.

In the senior level IS implementation course, the problem is not as well structured. Now students must develop features, based on imprecise and often contradictory descriptions provided by the client. In many cases, the client may identify a similar feature on another web site, as she describes how her version of the feature needs to differ. On first glance, creation of the user interface elements appears clear, as students have the necessary HTML knowledge or can easily acquire it. In a similar manner, students may deduce data needs, and their knowledge of database design is sufficient to create an appropriate database. The software mechanisms that connect the user interface with the database present the design challenge. For simple problems, the design approaches taught in ASP.NET course may be sufficient. For large systems, students need deeper exposure to software design practices, including design patterns, layered architectures, and class refactorings.

CONCLUSIONS

For the senior level IS implementation course, a web development framework, based on practices and tools adopted from the open-source community, may be created to form a standardized development environment. Exposure to the framework provided students with knowledge of design practices for large systems, including use of a four layer architecture, MVC and associated design patterns, and programming standards. Students also learned how to use configuration management and automated integration tools.

Using the framework, students developed interactive features that allow nonprofit organizations to move their business processes online. Modifications of the course to emphasize technical skills over interpersonal skills were necessary to increase the quality of the students' code and the number of features moved into the production version. The balancing point between depth and breadth of student exposure to system development topics is an open question that depends on several factors, and it is one that may be resolved most appropriately by each IS program faculty.

REFERENCES

1. Alur, D., Crupi, J., Malks, D. (2001). *Core J2EE Patterns: Best Practices and Design Strategies*, Palo Alto: Sun Microsystems Press.
2. Apache (2009). Coding Standards, http://portals.apache.org/development/code-standards.html, updated May 28, 2009.
3. Barrett, R., Delany, S.J. (2004). openMVC: A Non-proprietary Component-based Framework for Web Applications, *ACM: WWW 2004*, New York, May 17-22.
4. Corcho, O., Lopez-Cima, A., Gomez-Perez, A. (2006). The ODESeW 2.0 Semantic Web application framework, *ACM: WWW 2006*, Edinburgh, Scotland, May 23-26.
5. Cruise Control (2009). Welcome to CruiseControl.NET, http://confluence.public.thoughtworks.org/display/CCNET/Welcome+to+CruiseControl.NET, visited June 29, 2009.
6. Fowler, M. (2006). Continuous Integration, http://www.martinfowler.com/articles/continuousIntegration.html, updated May 1, 2006, visited June 29, 2009.
7. Fowler, M. (2003). *Patterns of Enterprise Application Architecture*, Addison-Wesley.
8. Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley.
9. Gorgone, J.T., Davis, G.B., Valacich, J.S., Topi, H., Feinstein, D.L., Longenecker, H.E., Jr. (2002). *IS 2002: Model Curriculum and Guidelines for Undergraduate Degree Programs in Information Systems*, Association for Computing Machinery, Association for Information Systems, and Association of Information Technology Professionals.
10. Kojarski, S., Lorenz, D.H. (2003). Domain Driven Web Development with WebJinn, *ACM: OOPSLA '03*, Anaheim, CA, Oct. 26-30.
11. Martin, R.C., Martin, M. (2007). *Agile Principles, Patterns, and Practices in C#*, Prentice Hall.
12. NHibernate (2009). Relational Persistence for Java and .NET, http://www.hibernate.org/, visited June 29, 2009.
13. Ricci, L.A., Schwabe, D. (2006). An Authoring Environment for Model-Driven Web Applications, *ACM: WebMedia '06*, Natal, RN, Brazil, Nov. 19-22.
14. Strickland, T.J., Jr. (2008). Student-based IT for Nonprofits – An Alternative to Do-It-Yourself IT, *Issues in Information Systems*, Vol. IX(1):146-151.
15. Strickland, T.J., Jr. (2003). Real Projects, Real Lessons – What Students Learn and What the

Instructor Has Learned, *Issues in Information Systems*, Vol. IV(2):693-699.
16. Subversion (2009). http://subversion.tigris.org/, visited on June 29, 2009.
17. Sun (1999). Code Conventions for the Java Programming Language, http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html, updated April 20, 1999, visited June 29, 2009.
18. Troelsen, A. (2005). *Pro C# 2005 and the .NET 2.0 Platform*, Apress.
19. Vosloo, I., Kourie, D.G. (2008). Server-Centric Web Frameworks: An Overview, *ACM Computing Surveys*, Vol. 40(2):1-33.