Issues in Information Systems
Volume 18, Issue 1, pp. 118-127, 2017

ADDING MVC TO THE CAPSTONE MIS SYSTEMS DEVELOPMENT COURSE

Thom Luce, Ohio University, luce@ohio.edu

ABSTRACT

This paper reviews continuing attempts to upgrade the capstone MIS systems development course at Ohio University. The latest iteration of the course includes live-client projects using model-view-controller development with ASP.NET MVC and agile Scrum. Leading up to the course revision we review what MVC is and some of the benefits ascribed to ASP.NET MVC development as compared to ASP.NET webforms development. We describe features of MVC development in the Visual Studio environment, including the use of Entity Framework Code First and Bootstrap styling. The paper concludes with an overview of how students are introduced to MVC, Entity Framework and Bootstrap and how they then take that knowledge and use it to develop a live-client project through a series of Scrum Sprints.

Keywords: Capstone MIS course, Systems development, MVC, Entity Framework, Bootstrap, Scrum.

INTRODUCTION

The MIS program at Ohio University is based heavily on hands-on experiences designed to both balance and enhance the theoretical aspects of the program. The program exists in an AACSB accredited Business school with over half of all undergraduate courses taken by all business students and the MIS major, after a switch to semesters several years ago (Luce and Matta, 2010), is limited to six courses that can be completed in three semesters.

Prerequisites to the capstone systems development course include a course in data management/database/BI and an introductory programming course. The database course uses Microsoft SQL server and touches on simple database driven webforms development using Microsoft Visual Studio and ASPNET data bound controls. The programming class uses Visual Studio to teach programming fundamentals with C# and includes an extensive section on code-based database activities. Much of the course relies on single-page applications but the final assignments introduce communication between multiple-page applications using Sessions, query stings and the Windows Communication Foundation (WCF).

Because of available technologies and time limitations under quarters, the capstone course initially focused on analysis and some prototyping. Microsoft's introduction of Active Server Pages in 1996 (Microsoft, 2017) made it possible to implement web pages for some of the prototypes but many problems still existed for our relatively non-technical students. These problems included the need to manage all data on the client and server side, the lack of any unified security/authentication system, the requirement to mix client side and server side code on the same page and the need to write database operations directly against Microsoft ActiveX Data Objects (Microsoft, 2017b) and other related technologies.

The release for the first .NET Framework and the first version of ASP.NET in 2002 and subsequent upgrades to both made it possible for our students to start developing full-featured web sites. With the release of the ASP.NET Framework 2.0, Microsoft introduced code-behind pages allowing the separating of business logic from interface logic. They also introduced the ViewState to manage flow between web cycles, Membership controls for creating user accounts and related security functions, data bound controls for easy manipulation of simple database operations, validators that reduced the need to hand-code simple data validation requirements, AJAX for asynchronous operations, and much more. The sum of these changes made it possible for our students to create more robust, fuller-featured applications than ever before.

Our students generally did not come to the capstone class with any design background. They had little or no exposure to HTML and even less to CSS. They knew about simple MasterPages but otherwise had little sense of how to develop useful and aesthetically pleasing web sites. To solve those problems, we modified the course adding a focus on standards based development and the use of standards compliant templates starting in 2013 (Luce, 2104).

Prior to 2014 the course was taught using a simple systems development life cycle, or more accurately an evolutionary prototype - spiral approach mix with three or four major deliverables over the course of the semester. Students had difficulties determining what to include in each iteration and instructors and students both had difficulties determining how much had been accomplished, what needed additional work and what was complete. Agile development using Scrum was introduced in 2014 to deal with these problems (Luce, 2016). The creation of a product backlog, sprint planning sessions and a sprint backlog along with reviews and retrospectives helped to manage these problems.

The capstone course uses a live-client, team-based development project. Prior to 2015 projects were solicited from student groups, departments and colleges on campus, local non-profits, student family businesses and other student connections. Vetting the projects for consistency was difficult at best and most semesters saw a wide range of difficulty levels among projects.

In the fall of 2015 we started collaborating with the Columbus office of Centric Consulting, a midsized national consulting company. They provided us with a project that was challenging and interesting to students. They came to campus to launch the project, provide an initial set of product backlog items and setup and populated Trello task boards (Trello, 2017) for use in our Sprints. They also supported some of their consultants acting as Product Owners for our Scrum teams. In contrast to previous semesters where every team worked on a different project, our partner suggested that all teams work on the same project in a "Centric-Ohio Challenge" with a plaque awarded to the best solutions at the end of each semester.

At the beginning of our first semester working with Centric, they offered to provide us with a project template the teams could use as a starting point. The starting point provided was an ASP.NET MVC (Model View Controller) project template (Microsoft, 2017h) created in Microsoft Visual Studio. At that point in time no one in the department, including the course instructor had any experience with MVC. It was quickly determined that the learning curve was too steep to master in a semester that had already begun. We also lacked an understanding of what MVC was, why companies might choose it over other technologies such as ASP.NET web forms, and what problems MVC development solved when compared to ASP.NET web form development.

WHAT AND WHY OF ASP.NET MVC

ASP.NET MVC is Microsoft's implementation of the MVC design pattern in Visual Studio. MVC, the model-view-controller pattern separates important software issues into three distinct concerns (business data model, user interface and processing control). The MVC model with its separation of concerns was first proposed and implemented by Trygve Reenskaug (Reenskaug, 1987) while working on Smalltalk at the Xerox Palo Alto Research Center. Microsoft released its first implementation, ASP.NET MVC1 in 2009, and the current version, ASP.NET MVC5 in 2013 (DotNetTricks, 2017). Microsoft is in the process of releasing the latest version, ASP.NET Core (Microsoft, 2017e)
Some of the advantages (Stack Overflow, 2017) of using ASP.NET MVC over ASP.NET Webforms include 1) complete control of the HTML (there are no ASP user controls generating HTML that the developer cannot easily modify), 2) easy separation of concerns, no ViewState (significantly improving bandwidth utilization), easy integration with JavaScript (including JQuery) and easy integration with test driven development. Current implementations of ASP.NET MVC fully embrace Bootstrap (Bootstrap, 2017) and incorporate easy data modelling with Entity Framework (Microsoft, 2017g)
Controllers
In an ASP.NET MVC web site incoming requests are directed to a Controller by a Routing module (Vihite, 2017) which is fully configurable by the developer. Controllers typically contain several Action Methods which can receive and process the request. As an example, a controller called OrdersController, may contain an action method to list all orders along with methods to implement basic CRUD functionality: create, read, update and delete orders. Action methods, by default, respond to HTTP Get requests but the addition of an [HttpPost] attribute allows them to respond

to Post requests. Controllers may interact with Models and return an ActionResult. Action results can redirect to another action method or route, return Json, Javascript or other objects or return Views and PartialViews (Shaikh, 2015).

Models

Models hold business domain specific information and business logic. In ASP.NET MVC models reside in a Models folder and are represented by public classes with public properties. Figure 1 shows two simple model classes representing pet owners and pets.

The figure illustrates the use of Data Annotations (Microsoft, 2017i) to improve the readability of web pages generated based on a model, for example, the "lastName" field will be displayed as "Owner's last name" on any web page that uses the field. The figure also shows the use of Data Annotations for validation. Fields marked as [Required] must contain data before a record can be stored or updated. Annotations such as [DataType(DataType.PhoneNumber)] and [RegularExpression…] add specific validation requirements to data entry.

```
public class Owner
{
    0 references | 0 exceptions
    public int ID { get; set; }

    [Required]
    [Display(Name = "Owner's first name")]
    0 references | 0 exceptions
    public string firstName { get; set; }

    [Required]
    [Display(Name ="Owner's last name")]
    0 references | 0 exceptions
    public string lastName { get; set; }

    [Display(Name ="Owner's email address")]
    [RegularExpression(@"^[A-Za-z0-9]([_\.\-]?[a-zA-Z0-9]+)*)@([A-Za-z0-9]+)(([\.\-]?[a-zA-Z0-9]+)*)\.([A-Za-z]{2,})$",
        ErrorMessage = "Email is not valid")]
    0 references | 0 exceptions
    public string eMailAdress { get; set; }

    [Display(Name ="Mobile phone number")]
    [DataType(DataType.PhoneNumber)]
    0 references | 0 exceptions
    public string phoneNumber { get; set; }

    0 references | 0 exceptions
    public ICollection<Pet> pets { get; set; }
}
1 reference
public class Pet
{
    0 references | 0 exceptions
    public int ID { get; set; }

    [Display(Name="Name of pet")]
    [Required]
    0 references | 0 exceptions
    public string petName { get; set; }

    [Display(Name ="Breed of pet")]
    [Required]
    0 references | 0 exceptions
    public string breed { get; set; }

    [Display(Name ="Pet owner's name")]
    [Required]
    0 references | 0 exceptions
    public int ownerID { get; set; }

    [ForeignKey("ownerID")]
    0 references | 0 exceptions
    public virtual Owner petOwner { get; set; }

}
```

Figure 1. Sample Model Class Files

By default, a property called, ID, is chosen as the primary key of an object created. If the model does not have an ID property then a property in the form of classnameID is selected. For example, if the Pet class didn't have an ID property the default key would be PetID. The designer can override the defaults by placing the [Key] annotation in front of the desired primary key property. Foreign keys are also inferred by their names and associated classes but this can be overridden with the [ForeignKey] attribute as shown in the Pet class.

Models can create navigation between classes. The "ICollection<Pet>" property of Owner allows navigation from an owner to all his or her pets. The "virtual Owner petOwner" property in Pet allows navigation from a specific pet to the pet owner's information.

ASP.NET MVC5 uses Entity Framework to map classes like those shown in Figure 1 to relational databases. Version 6 of Entity Framework (Microsoft, 2017c) supports three main approaches to this mapping; Model First, Database First and Code First. The Model First approach starts with a model created with an EF modeling tool. Database First creates models from existing databases and Code First uses models like the one shown in Figure 1 to generate appropriate database tables and properties. One major benefit of the Code First approach is the ease with which the model can be updated without needing to manually rebuild the underlying database. This is accomplished by enabling Data Migration (Joshi, 2017) which can both update the database and migrate existing data to the new model.

Entity Framework and Data Migration depend on a class called the Database Context. Figure 2 shows a sample context class. Additional code to support data migration and to provide initial seeding of the database tables may be found in the context file of more complex applications.

```
using System.Data.Entity;
using IACIS2017.Models;

namespace IACIS2017
{
    1 reference
    public class IACISContext : DbContext
    {
        0 references | 0 exceptions
        public IACISContext() : base("name=csIACIS")
        {
        }
        0 references | 0 exceptions
        public DbSet <Owner> owner { get; set; }
        0 references | 0 exceptions
        public DbSet <Pet>  pet{ get; set; }
    }
}
```

Figure 2. Sample Context Class File

Views

A view uses a combination of HTML and CSS along with code to generate the HTML returned to the user. The View Engine that does this in MVC5 is called the Razor View Engine (Microsoft, 2017f). Figure 3 shows part of a view used to produce a table of owners.

```
@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.firstName)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.lastName)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.eMailAdress)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.phoneNumber)
        </td>
        <td>
            @Html.ActionLink("Edit", "Edit", new { id=item.ID }) |
            @Html.ActionLink("Details", "Details", new { id=item.ID }) |
            @Html.ActionLink("Delete", "Delete", new { id=item.ID })
        </td>
    </tr>
}
```

Figure 3. Sample View showing Razor Syntax

The Razor view engine uses "@" to switch between HTML and code. This sample code shows a for each loop used to create HTML table rows for each record in the owner model. Coding elements beginning with @Html are known as HTML helpers. The @Html.DisplayFor code uses Lambda expressions to extract field data from the current record and display it on the page as text. The @Html.ActionLink code creates hyperlinks (anchor tags) to different action methods in the controller. In each case the hyperlink contains the ID of an owner.

Creating Controllers

When a controller is created in Visual Studio the developer choses the type of Scaffolding (Harrison, 2017) to add. This selection allows the developer to create an empty controller, create a controller with read/write actions or to create a controller and views with Entity Framework. If creating a controller and views with Entity Framework is selected, the developer supplies the name of the model class they want to use along with the name of the data context class. The developer also has the option to generate Views and to include a layout page which is like a MasterPage in Webform development.

Controllers created following this pattern have several action methods that implement basic CRUD functionality. There are methods to create new records, methods to read and display records, methods to update records and finally methods to delete records. Each method has a corresponding View created by the system. All methods and views can be customized by the developer as needed.

Figure 4 is the Owners/Create View showing the Display names entered as data annotations in Figure 1. The figure also shows some of the validation messages created by the data annotation.

Figure 4. Create View Showing Required Field and Data Type Validation

Figure 5 shows a dropdown list automatically created because of the navigational properties in the model. In this case, the actual data stored by the model is the ownerID but the data displayed is the owner's first name. A small coding change will allow first and last name in the dropdown.

Figure 5. Create View Showing Dropdown List Enabled by Navigational Properties in the Model

THE NEW/REVISED COURSE

As previously discussed, students entering our capstone systems development course have one data management and SQL course and one course introducing C#. Most students have no additional programming background and most have little or no understanding of HTML and even less of Cascading Style Sheets.

Because of this we start with a series of in-class examples and exercises to review HTML and CSS. Materials for these exercises are drawn from the internet, especially from W3Schools (W3Schools, 2017).

Next, we introduce ASP.NET MVC, create a ASP.NET Web Application project using the supplied MVC template, selecting Individual User Accounts and indicating that we do not want hosting to the cloud. The default template creates a Home Controller, a shared layout containing menu bar items for the Home page as well as Contact and About pages. The menu bar also has links to a Microsoft Identity (Microsoft, 2017d) login page and a registration page. All pages are formatted with Bootstrap styling. We spend time looking at the application's file structure, the three default

pages and the layout page and how the Bootstrap styles are applied to the pages. The first assignment is to create a simple templated project and to customize the home, about and contact views and the layout view to the student's own interest or taste.

During the second phase, we introduce the Entity Framework (EF) and Code First. We talk about how to create a model, how keys are generated, and how to create a context class. For the second assignment students are required to select a simple many-to-many relationship, create an EF model for their choice then create controllers for each object in the model. Once that is done they add links to the three controllers, either on the layout menu or by adding links on the Home page. Finally, they are asked to run the application and add several records to each table.

The final part of the introduction to MVC looks at data annotations for display and validation of model data and at components of Bootstrap for formatting of views. The assignment requires students to add appropriate display and validation annotations to their model. It also requires that they add some additional bootstrap styling to their pages. Specifically, they are asked to do some table styling, use one or more panels or alerts and add some icons and color to their pages.

At the completion of the three exercises students are placed in teams and our client comes in to launch the main project for the semester. During the project launch students are introduced to the overall goals of the project and are given an initial set general product backlog items on a Trello task board.

The remainder of the semester is a series of two week sprints. Each sprint begins with a sprint planning session during which students study their product backlog, add new items based on feedback from their product owner or instructor, assign story points to backlog items using planning poker or other techniques, and select items for the current sprint. After updating their Trello task boards, they submit a link to the task board for grading.

The next four or five class meetings are development sessions, often with additional short lectures and demonstrations on topics the students might find relevant to the project. These topics include discussions on how to use Microsoft Identity, how to link user information to login accounts, how to upload and display images, how to secure pages and how to control who had access to pages, how to incorporate a rich text editor on their pages, how to implement searching, how to do data migrations, how to create dropdown lists with first and last names, how to display names on pages that initially only displayed primary keys, etc. During this time students are encouraged to have a daily scrum meeting during class time but the nature and length of the classes typically makes that difficult.

Product owners hold conference calls, Skype or Slack meetings with team members but because they aren't on campus, at the end of each Sprint for a review or retrospective meetings, we created a combined review and retrospective document that all teams complete and submit before their sprint work is graded. This document asked them to review what they accomplished during the sprint and to include a snapshot of their Trello board at the end of the Sprint and before the next planning session. The retrospective part of the deliverable asks them to reflect on the process used, what was good, what was bad, what was ugly and what they could do to improve things for the next time.

The course concludes with a formal presentation to members of the client's team.

SUMMARY AND CONCLUSIONS

The current version of the capstone systems development course at Ohio University introduces students to MVC development in Visual Studio using Scrum to manage the development process. Students learn about the MVC approach through classroom exercises and a series of individual assignments. Once they have the basics, students are divided into teams and presented with a real-world problem which they work on for the remainder of the semester in a series of four or five two-week Scrum Sprints.

MVC based development fits nicely with the Agile/Scrum approach used to manage projects. MVC code-first models are easy to modify as new requirements are revived. Controllers can be quickly rebuilt for new and changing data entities. Scaffolding options build into Visual Studio facilitate changing Views to accommodate changes in the data

entities. Bootstrap allows students to quickly create intuitive designs which are automatically responsive to different devices. The fact that Controllers are independent of each other also facilitates parallel development with different students working on different parts of the project at the same time.

The results reported here were not based on a research design but intended to respond to an industry need. No formal analysis has been performed on the recent changes to the course but we do have anecdotal feedback from both our client and students. Since Centric Consulting has worked with us for the past two years they have observed the results from a webforms based development approach and now from MVC development. The first semester MVC results were viewed as significantly better than results from the previous year's webforms development as indicated by the following quote from our lead contact at Centric Consulting:

> The big push we had to move towards MVC was largely driven by wanting to keep the technologies that the students are using to create the product more current. There is not much of a place for webforms anymore as the technology has shifted towards a more data driven design that seems to be more conducive to the types of projects there are out there. Webforms offered the students an opportunity to be exposed to the C# code, without having to have much understanding of what is happening "under the hood". What we have seen with the switch to MVC from webforms is a fuller understanding of a more modern full stack application. Judging by the presentations that I have seen over the years, the students have seemed to have a better understanding of application development by the end of the semester using MVC as compared to the webforms (e.g. talking about data, controllers, ui as separate, integrated pieces). I also think that the quality of the applications has increased significantly with the switch to MVC (in particular, fall semester 2016 projects were the best I think I've seen yet). The projects tended to have more functionality, and worked more consistently. (Flatt, 2017)

Student comments include:

- I think the MVC framework was a very beneficial concept to learn as it was outside the box of the traditional programming we've been taught. It seemed to be more relevant to real world applications.
- Great course! Seemed relative to real world applications!
- (Author's note – several students thought that MVC should be taught in an earlier course and not introduced as a new technique in a live client project course)
- Overall it was a very good class and I'm glad I took it. I can see myself using MVC to build websites in my future. It is very popular and intuitive.
-

Major benefits of this approach are best summarized by the following quote from our lead contact at Centric Consulting (Flatt, 2017):

> The big win in my mind though has been knowing that the students are being exposed to and asked to create a very real project with a technology that is being utilized in the industry. It also helps to drive home better the understanding of object oriented development as they themselves are creating all the objects and utilizing them rather that the webform's auto generation of all sorts of code behind. Aside from the improved results of the project's application, the students have the opportunity to dive into full stack development, giving them an opportunity to see a more technical side of a major that tends to lean very heavily on the softer side of technology. While development is not everyone's favorite thing ever, it's another opportunity to attract this upcoming batch of professionals to the world of programming. Creating a full stack application is a very non-trivial task, and I think it's awesome that we can put that in the hands of these students as part of their education.

REFERENCES

Reenskaug, T. (1978). *MVC, XEROX PARC 1978-79*. http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html Accessed 18 April 2017.

Bootstrap (2017). *Getting Started*. http://getbootstrap.com/getting-started/ Accessed 21 April 2017

DotNetTricks (2017). *A brief history of ASP.NET MVC framework*. http://www.dotnettricks.com/learn/mvc/a-brief-history-of-aspnet-mvc-framework Accessed 18 April 2017.

Flatt, J. (2017). Centric Consulting, *Personal Communication,* April 30, 2017.

Harrison, N. (2017). *Using Scaffolding to Create MVC Application with Visual Studio.* https://www.simple-talk.com/dotnet/asp-net/using-scaffolding-to-create-mvc-applications-with-visual-studio/, Acceded 21 April, 2017.

Joshi, N. (2017). *Perform Code First Migration in ASP.NET MVC 5*. http://www.c-sharpcorner.com/UploadFile/4b0136/perform-code-first-migration-in-Asp-Net-mvc-5/ Accessed 21 April, 2017.

Luce, T. (2014). Using Standards and Best Practices to Help Students with Limited Technical Skills Create Full-Featured Web Sites. *Issues in Information Systems, 15*(1), 209-216.

Luce, T. (2016). Adding Agility to the MIS Capstone. *Issues in Information Systems, 17*(1), 119-127.

Luce, T., Matta, V. (2010). Academic Calendar Change Induced Curriculum Change. *Issues in Information Systems, 11*(2), 146-151.

Microsoft (2017). *Active Server Pages Tutorial*. https://msdn.microsoft.com/en-us/library/ms972337.aspx Accessed 16 April, 2017.

Microsoft (2017b). *Microsoft ActiveX Data Objects.* https://docs.microsoft.com/en-us/sql/ado/microsoft-activex-data-objects-ado Accessed 16 April, 2017.

Microsoft (2017c). *Entity Framework (EF) Documentation*. October 23, 2016, https://msdn.microsoft.com/en-us/library/ee712907(v=vs.113).aspx, Accessed 21 April, 2017.

Microsoft (2017d). *Getting Started with ASP.NET Identity*. https://www.asp.net/identity Accessed 21 April, 2017.

Microsoft (2017e). *Introduction to ASP.NET Core.* Accessed 18 April 2017.

Microsoft (2017f). *Introduction to ASP.NET Web Programming Using Razor Syntax (C#)* https://docs.microsoft.com/en-us/aspnet/web-pages/overview/getting-started/introducing-razor-syntax-c Accessed 21 April, 2017.

Microsoft (2017g). *Introduction to Entity Framework*. https://msdn.microsoft.com/en-us/library/aa937723%28v=vs.113%29.aspx Accessed 21 April 2017.

Microsoft (2017h). *Learn About ASP.NET MVC.* https://www.asp.net/mvc Accessed 17 April, 2017.

Microsoft (2017i). *Part 6: Using Data Annotations for Model Validation*. https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions/mvc-music-store/mvc-music-store-part-6. Accessed 21 April 2017.

Shaikh, S. (2015). *Learn different types of Action Results in MVC*. November 15, 2015, https://www.eduonix.com/blog/web-programming-tutorials/learn-different-types-of-action-results-in-mvc/ Accessed 21 April 2015.

Stack Overflow (2017). *Biggest advantage to using ASP.NET MVC vs web forms*. http://stackoverflow.com/questions/102558/biggest-advantage-to-using-asp-net-mvc-vs-web-forms, Accessed 18 April 2017.

Trello (2017), https://trello.com. Accessed 17 April 2017.

Vihite, Ch. (2017). A Detailed Walkthrough of ASP.net MVC Request Life Cycle" (sic), http://blog.thedigitalgroup.com/chetanv/2015/06/30/a-detailed-walkthrough-of-asp-net-mvc-request-life-cycle/ Accessed 21 April 2017.

W3Schools (2017).*Tutorials*. https://www.w3schools.com, Accessed 21 April 2017.