|  |
| --- |
| Mercer university |
| Project 2 |
| SSE 657 - Object Oriented Project Methods |
|  |
| **Tanya Do & John Robison** |
| **10/27/2014** |

|  |
| --- |
|  |

Table of Contents

[Introduction 1](#_Toc402096099)

[Problem Statement 2](#_Toc402096100)

[Discussion With Customer 2](#_Toc402096101)

[Commonality and Variability Analysis of System 2](#_Toc402096102)

[Features List 3](#_Toc402096103)

[Domain Analysis 4](#_Toc402096104)

[Use Case Diagram 4](#_Toc402096105)

[Design Pattern Selection 6](#_Toc402096106)

[Determining the Architectural Significance of Components 7](#_Toc402096107)

[Risk Analysis and Reduction 9](#_Toc402096108)

[Additional Discussion With Customer 9](#_Toc402096109)

[Design of Core Features and Use of Design Principles 10](#_Toc402096110)

[Home Page 11](#_Toc402096111)

[User Accounts 11](#_Toc402096112)

[Product Listings 11](#_Toc402096113)

[Mock Scenarios of System Interaction 11](#_Toc402096114)

[Account Registration 11](#_Toc402096115)

[Editing Account Information 11](#_Toc402096116)

[Creating a Product Listing 11](#_Toc402096117)

[Editing a Product Listing 12](#_Toc402096118)

[Purchasing a Product 12](#_Toc402096119)

[Appendix A: References 13](#_Toc402096120)

**Table of Figures**

[Figure 1: MakerStore Use Case Diagram 5](#_Toc402096121)

[Figure 2: MVC Interaction Diagram 6](#_Toc402096122)

# Introduction

This report and the project that it corresponds to are intended to display mastery of the concepts contained in Chapters 6 - 8 of the book Head First Object-Oriented Analysis and Design by McLaughlin, Pollice, and West. These chapters build on the previous chapters by explaining how those Object-Oriented Analysis and Design (OOAD) concepts can be applied to larger, real-world problems to make them more manageable to design.

The first step offered by the text to make a large problem less intimidating to approach is to break the project into smaller, logical pieces that can be developed using the OOAD concepts provided in the previous chapters. To determine these logical pieces, techniques such as conversations with the customer and commonality and variability analysis can be used to determine the features of the system. Once the features are determined, the most architecturally significant features are determined using the three Q's of architecture and are then designed first to reduce risk. After the architectural design of the system is completed, design principles are applied to ensure that the system is not rigid, fragile, or immutable.

# Problem Statement

With the opening of SparkMacon, Macon's own Maker Space, there is now a need for a web service that will allow makers of all trades and skill sets to advertise their products and promote their business. This service will allow makers to create a profile that contains: a biography describing their craft, previously completed or sold projects, products currently for sale, user reviews of the maker's products, and a place for users to request unlisted or new products. The people behind SparkMacon are devoted to building a strong maker community in the Macon area and want a system that helps to push their makers further. To build this community that SparkMacon aspires to, the web service will need features that promote community involvement: featured makers (i.e. Maker of the Week), community forums, the ability to like or follow products and users, etc.

# Discussion With Customer

# Commonality and Variability Analysis of System

In order to ensure that this web service meets all of the customer's expectations, our team performed a commonality and variability analysis on the system. This means that we compared the proposed systems to preexisting web services and described what these systems offered or did that the customer either wanted their system to be like or not be like. In addition to ensuring that the designed system meets the customer's expectations, this commonality and variability analysis will aid our team in determining the major features of the system which will allow us to determine the most architecturally significant components.

Based on the discussion with the customer, the core functionality of the web service should be much like Etsy, an e-commerce marketplace for creators of unique items to sell their goods. The customer stressed that they wanted posted items to be actually made by the users that posted the listings, so the web service should not be like Amazon or Ebay. Since the customer is interested in strengthening the Macon community, the system should not be international or even open to anyone like Etsy, Amazon, and Ebay are. The area-based product listing system used by web services such as Craigslist is ideal for building this strengthened community of Macon makers.

# Features List

After talking to the customer and doing a commonality and variability analysis of the system, our team compiled a list of features that the system will support. Listed below are the features that were determined as well as the requirements that make up each feature:

* User Accounts
  + User biography
  + List of products available
  + List of previously sold products
  + Product Requests
  + User Rating/ Review
* Product Listings
  + Product description
  + Product Category
  + Price
  + Product Rating/ Review
* Home Page
  + Maker of the Week
  + Featured Products
  + News
  + About
  + Link to Spark Macon
* Community Involvement
  + Community forum
  + Event coordination/ planning
  + Liking or following products or users

The User Account feature will be responsible for storing all of a user's information: biography, products, user rating, and requests. A user will be able to create an account at any time and begin selling and marketing their products. Once an account is created, the user will be able to login to the system and edit/ update their account information as well as create postings for products.

The Product Listing will contain all of the information for a user's product: product description, category, price, and product reviews. A user with a registered account will be able to post listings for their products. The use of the description and category will allow other users to search for products that interest them, such as robotics or art.

The Home Page will be the main access point of the web service. It will be used to display important pieces of information, such as the featured Maker of the Week, upcoming events, and news about the makerspace. This page will also contain information about what SparkMacon's mission is as well as provide links to their website.

There will be several components that will aim to build community involvement, including user forums, announcing local makerspace events, and the ability to like or follow other users' accounts and products.

# Domain Analysis

# Use Case Diagram

The users should be able to interact with the system in many different ways. These interactions include:

* Creating and editing accounts
* Creating and editing product listings
* Purchasing products
* Contributing on forums
* Navigating to SparkMacon's website
* Rating users and products.

The account interactions will be handled through registration and log in processes. A new user will have to register with the system in order to have a username and password that they will be able to log in with. Once they are registered, they will be given the ability to edit their account information, such as their biography and available products.

Only registered users will be able to create and edit product listings. Once a user is registered, they will be able to post as many products as they have to sell. Once a product is listed, only the user who posted it will be able to make edits to the listing. These edits include the product's description and price.

Registered users will also be able to contribute to the community by rating other users and their products as well as commenting on forums. If a user is interested in learning more about the SparkMacon makerspace or learning about upcoming events, they will be able to click a link that navigates to the SparkMacon site regardless of if the user is registered.

Figure 1 below visualizes the users' interactions with the system in a UML Use Case Diagram.

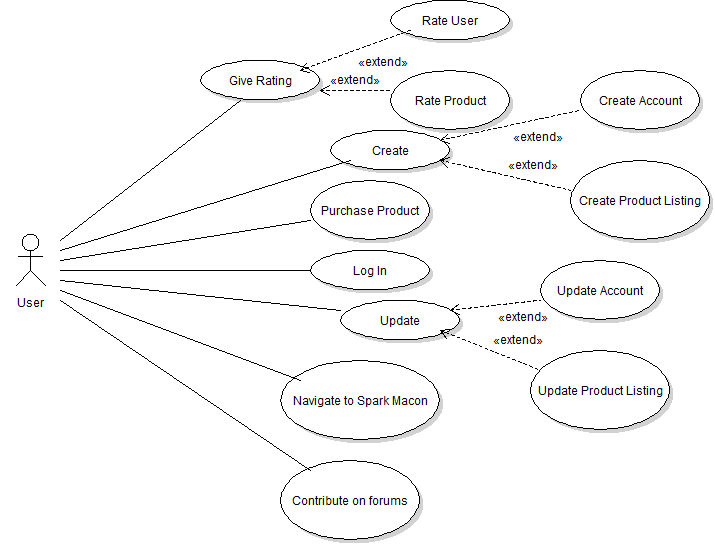


Figure : MakerStore Use Case Diagram

# Design Pattern Selection

The designed system will be built on the ASP.NET MVC Framework. The use of this framework utilizes the MVC (Model-View-Controller) design pattern which encourages designs to be loosely coupled through a separation of concerns. This separation is achieved by using the three main components: Models which handle the data and logic of the system, Views which display the data in a way that makes sense to the user, and Controllers which are responsible for dealing with user inputs by updating the models and view. Below in Figure 2 is a diagram that visualizes the interactions between these components.



Figure : MVC Interaction Diagram

The ASP.NET framework builds on the MVC design pattern by relying on a "convention over configuration" approach which reduces the amount of code required for a project. This approach requires certain design decisions, such that items are placed in the correct locations (i.e. Controllers must be in the Controller folder) and that they are named appropriately (i.e. Controllers will be named ending with "Controller"). Not only does this convention based approach reduce code requirements, it also aids in overall readability of the system.

Due to the loose coupling offered by the MVC design pattern, our team chose ASP.NET MVC as the framework for this system because it allowed for code reuse and parallel development. Since the models, views, and controllers are separated, they can easily be implemented into another application. An example of this would be if the customer requested a mobile application since the model and controller classes could be reused. The only major change in this example is that new views would have to be created specifically for the mobile platform. This separation also allows components to be designed and built separately. One developer can build the store user interface in the views while another developer designs the underlying business logic in the models.

In addition to all of the benefits of MVC's separation of concerns listed above, our team decided to develop with the ASP.NET MVC Framework for its use of .NET languages (C# and Visual Basic) which we are familiar with as well as the use of its easy to learn view engine, Razor. C#, our .NET language of choice, is extremely powerful and easy to develop with due to its number of available libraries and the use of Microsoft's Intellisense, which can auto-fill code for the developer such as variable and method names. The Razor view engine is a combination of HTML and either C# or Visual Basic, which makes it very natural for .NET developers to write and learn. The use of C# in the markup allows for very powerful dynamic web pages. Razor also allows the use of layouts which enables developers to have a single Razor file act as a template for all other views. These layouts reduce duplicate code by encapsulating common view elements into the layout. The final advantage of Razor that will aid in development is the ability to use Intellisense to quickly write the markup and code contained in the Razor files.

# Determining the Architectural Significance of Components

To select which feature to begin with, our team looked at each feature and asked the Three Q's of architecture:

* Is it the essence of the system?
* What does the feature mean?
* How do you implement the feature?

A feature is deemed architecturally significant if it is either the essence of the system, the meaning of the feature is unclear to the developers, the developers are unsure of how to implement the feature, or any combination of the previous three. Neither of the Three Q's carries more significance than the other and thus asking these questions only gives us an unweighted list of the most architecturally significant features.

The features that have been determined to be the essence of the system are: the user accounts, the product listings, and the home page. User accounts are the core of the system. Without users, new content cannot be generated and Macon area makers will not be able to display their crafts. Just like a makerspace without makers is just an empty building, a makerstore without makers is just a blank website. Adding to this point, a store without items is just an empty building. This means that the product listings are also essential to the system. However, the system can still operate to a certain extent with only user accounts. Makers will still be able to describe their craft and promote themselves without the ability to post listing for their products. Finally, the home page acts as the main access point of the system. It will contain a description of the site, links to all of the other features, and a link to SparkMacon's website. This access point is especially important for users who are not familiar with what the system's goals are or the makerspace that is supporting it.

The product listing was a feature that our development team needed clarification on to get a better understanding of. Some of the components of this feature that our team is unsure of are: product categories, listing formatting, and whether the customer would want similar products to be listed. Since product categories should not be something that a user can add or remove, there needs to be a set list of available categories that a product can be. Our team will need to be given this list in order to develop the database that will store product information. Once the categories are determined, our team will need to know what format the product information should be presented in. This includes the layout of the web page as well as what information should be presented to users (such as description, posting user, or price). When performing the commonality and variability analysis of the system, our team noticed that almost every online store service implemented some form of a related products function. We will need to discuss this idea with the client to see if it would be an addition to the product feature that they would be interested in adopting.

The user account was a feature that our development team was unsure of how to implement. Since our team does not have experience with handling user authentication and protection, it is a feature that we will need to thoroughly research, experiment with, and test. User security is very important because user accounts will contain personal information, such as credit card information. Aside from the security aspects, the process for creating an account must be intuitive and not overly intrusive. Our team will research the most effective ways to facilitate the process of creating and managing accounts and account information.

After analyzing these Three Q's, our team determined that the most architecturally significant features of the system were: the user account , the product listing, and the home page. The community involvement functionalities were not deemed architecturally significant for the following reasons:

* These functionalities were regarded as nice-to-haves and thus non-essential to the system
* The meaning of the functionalities (such as the forums) were clear and obvious
* The implementation was well within our team's capabilities

Determining these architecturally significant components narrowed down the list of features to help our team find an appropriate starting place that would help to reduce the overall risk of the development.

# Risk Analysis and Reduction

With the architecturally significant features determined, our team needed to determine the order in which we developed each feature in such a way that will reduce risk. Since it was considered to be the main access point of the system, we decided to develop the home page first. Having this feature developed will allow to have a place to build off of for the remaining features. This main access point will also act as the "glue" that holds all of the remaining features together, so having this feature in place first will reduce the risk of subsequent features not merging together properly.

After the home page is set up, our team will work on the user account functionality. This will include the registration, log in, and account information functions. This feature was selected to be the next task because it enables users of the system to be able to fully interact with all of the components inherent in the system. It is also is required to have a user account system set up before setting up the product listing system because a user account will be required to post a product listing. Once the account system is developed, it will need to be integrated into the home page in such a way that a user can easily find and access it's functionalities. This feature was also selected to be the second task because of the research and experimentation required by our team to ensure a secure system. Keeping this task as close to the top of the priority list as possible will enable us to better estimate time requirements for this system which will allow us to stay on time and on budget.

Once the home page and user accounts have been developed, the product listings can be created. These listings must be available to view from both the home page and user account pages. Listings must also be creatable only by registered users and must only be editable by the registered user who created the product listing. This feature will require our team to further discuss with the client about the layout and format of the listings, so we will want to keep this task as a priority to mitigate any risk of delays or discrepancies due to miscommunications between our team and the customer.

Following these steps will ensure that risk is kept to a minimum in both design and implementation. The ordering of the tasks will be as follows: developing and designing the home page, developing and implementing the user accounts, and developing and implementing the product listings. For the reasons listed above, this task ordering will mitigate much of the risk inherent in the development and design of this system and help to ensure that our team remains on time and on budget. This risk mitigation will also allow our team to determine any potential bottlenecks or complications that we may run into in the development timeline which will allow us to plan in advance for how to deal with them.

# Additional Discussion With Customer

# Design of Core Features and Use of Design Principles

While developing this system, our team will implement the use of design principles such as the DRY (Don't Repeat Yourself) Principle to ensure that our system is maintainable and flexible. The main principles that will be implemented are:

* The Open-Closed Principle (OCP)
* The Don't Repeat Yourself Principle (DRY)
* The Single Responsibility Principle (SRP)
* The Liskov Substitution Principle (LSP)

The Open-Closed Principle is about the changeability of the system. The definition of the Open-Closed Principle in the words of McLaughlin, Pollice, and West is that "classes should be open for extension, and closed for modification". Simply, this means that the components of the system should be designed in such as way that the system is flexible without needing to be changed. Example implementations of this principle include inheritance of an abstract class and the use of private methods.

The Don't Repeat Yourself Principle is about avoiding repeated code. The definition provided by McLaughlin, Pollice, and West is that "avoid duplicate code by abstracting out things that are common and placing those things in a single location". In other words, the DRY Principle is about ensuring that functionality is placed in a single, logical place so that changes to this functionality will not require changes in multiple places of the system. The most common implementation of this design principle is through the use of encapsulation. However, this principle is not limited to software design and can be used in gathering requirements to ensure that no two requirements address the same topic. The use of this principle ensures that a system is easily modifiable and flexible.

The Single Responsibility Principle is closely related to the Don't Repeat Yourself Principle in that it deals with keeping functionality in a single place. McLaughlin, Pollice, and West define this principle by saying that "every object in your system should have a single responsibility, and all the object's services should be focused on carrying out that single responsibility". This means that each class in a software system should be directly related to only one task. This makes each class only have one reason to change, which reduces the effects of a change to the system. Cohesion is an example implementation of this principle, therefore software that is highly cohesive (and thus loosely coupled) is following the SRP principle.

The Liskov Substitution Principle is about the appropriate uses of inheritance and knowing when not to use inheritance. The definition of the Liskov Substitution Principle given by McLaughlin, Pollice, and West is that "subtypes must be substitutable for their base types". This means that any class that inherits from another class should be able to use the base classes methods without causing any problems. Inheritance that doesn't follow the Liskov Substitution Principle becomes hard to understand which can cause issues with implementation of the subclasses. Some alternatives that can be used when inheritance is not appropriate are: delegation, composition, and aggregation. Delegation is where functionality from another class is used to accomplish a task as opposed to extending the used class. This is useful for when the needed functionality does not need to be changed to meet the designed goals. Composition is where your class is made up of other families of classes. This is useful for when the implementation of a class may change at runtime. A side effect of this alternative is that once the composing class is destroyed, all of the composite classes that it owns are also destroyed. The last alternative takes care of this side effect. Aggregation is like composition in that it uses other families of classes to have a dynamic implementation at runtime but the composite classes will still exist outside of the context of the composing class.

## Home Page

## User Accounts

## Product Listings

# Mock Scenarios of System Interaction

## Account Registration

## Editing Account Information

## Creating a Product Listing

## Editing a Product Listing

## Purchasing a Product

# Appendix A: References

* McLaughlin, Brett, Gary Pollice, and David West. *Head First Object-oriented Analysis and Design*. Sebastopol, CA: O'Reilly, 2007. Print.
* "Easy Intro to ASP.NET MVC." *Easy Intro to ASP.NET MVC*. N.p., n.d. Web. 21 Oct. 2014. <http://www.beansoftware.com/ASP.NET-Tutorials/Intro-ASP.NET-MVC.aspx>.