**Technical Report**

**for**

**Wegmans Deli Kiosk**

**Version 1.0**

**Prepared by the Deli-cious Developers**

**Jared Schutt, Tim Heard, Warren Shaw, Dan Larsen**

**Sponsored by Wegmans**

**Joe Pizzo, Joe Zanino, Richard Vanderhorst, Steve Schwartz, Tony Diaz**

**Faculty Coach**

**Professor Kenn Martinez**

**Rochester Institute of Technology**

**May 05, 2014**

### 

### 

### Table of Contents

1. Project Overview
2. Basic Requirements
3. Constraints
4. Development Process
5. Project Schedule: Planned and Actual
6. System Design
7. Process and Product Metrics
8. Product State at Time of Delivery
9. Project Reflection

### 

### Project Overview

#### **Purpose**

On a busy day in a Wegmans’ grocery store, customers may experience congestion and long lines around the deli area. In addition to causing difficulty with ordering from the deli, high-traffic areas around the deli are also affected. The goal of the Wegmans’ Deli Kiosk Project was to address these issues and increase efficiency in Wegmans’ delis. This was achieved by allowing customers to order deli products using kiosks placed in strategic locations around the store. A mobile solution, potentially integrated with the existing Wegmans’ mobile application, is also a future goal. Not only will this reduce congestion, but customers will also be able to order what they want quicker and more easily.

The project can also potentially be applied to other Wegmans’ departments, changing the way that customers shop and how products are delivered to them.

In order to deliver the right solution, we needed to understand the ramifications of both kiosk and mobile solutions, while also understanding the long-term goals of the project. Specific initial deliverables from our team were proposed for both kiosk and mobile platforms. After customer input and further planning, a direction was decided on for the project and development commenced. Ultimately, the team and sponsors decided to develop a Windows 8 / .NET solution. However, our solution was designed in such a way, that any client side technology, such as android, ios, or even web, could be used to interact with our system.

The team utilized software engineering practices such as requirements elicitation, re-usable system design, continuous sponsor communication, and the use of a carefully chosen process methodology. This allowed us to produce high quality software that wa on time and met customer needs.

#### **Scope**

At the time of this document the scope of the project is the following:

The project encompasses 3 main components:

1. A RESTful WCF Service running on Windows Server 2008 R2
2. An ASP.NET web application that connects to the aforementioned service (1)
3. A Windows 8 application that serves as a customer and employee endpoint. This application interacts with the aforementioned service (1)

The WCF Service currently hosts a wide variety of restful endpoints that can manipulate the state of the service. These include but are not limited to Kiosk registration, Ordering functionality, Item catalog functionality, and administrative report/editing functionality. See the Web Services doc for more information on these endpoints. It also interacts with Zebra printers that are registered. Currently these printers use an IP address as means for identification. The service also relies on a MS-SQL database instance, and .csv files. These two areas are used to store item catalogs, orders, and all other information required by respective endpoints. See the server setup information for more information on the database and csv files needed.

The ASP.NET application currently consumes certain restful endpoints and organizes their return values into data that can be manipulated in a administrative fashion. Currently there is a 1-1 relationship between a service and ASP.NET application. In the future this could be expanded to where one ASP.NET application could consume the endpoints for many WCF services - in effect creating a distributed administration environment.

The Windows 8 application currently has two modes - Customer and Employee. The customer mode loads item catalogs from the server, consumes splash screens, checks for item catalog and splash screen updates, and processes orders

The Employee side loads item catalogs from the server, consumes previous orders, checks for item catalog and order updates, and can modify items in the item catalog. It also can use a manager login to edit said items.

The first time the app is registered it gives the user the option to choose what type it is, a printer to register to the kiosk, and then registers the kiosk using the restful service. At this point the kiosk can no longer switch between modes, and the app must be re-installed if that is desired. The app will always start up in the mode it is set to.

Currently the system does not interact with any outside systems. Information is put in via .csv files, and splash screen images, and all information from that point forward is kept internal using the restful calls.

#### **Desired Results**

The desired results of the system are a functional kiosk server and client system that can successfully order deli products. These orders shall be logged and printed. The order process will facilitate customers ordering deli products, and employees filling said orders. We think that the product in its current state fulfills these desired results.

### Basic Requirements

#### **System Requirements**

|  |  |
| --- | --- |
| **Feature** | **Description** |
| Process Customer Orders | The system must allow customers to place orders which will then be processed (saved and potentially printed) and persisted by the system and presented in the deli together with an audible chime to allow for fulfillment of that order. |
| System Configuration and Administration | System must be highly configurable through a remote administration UI. |
| System Reporting | Properly authenticated administrators will have the ability to view reports on the sales history and usage information for the system. |
| Remote System Administration | Administration and configuration of the system can be performed by a properly authenticated administrator through a remote web console. |
| Support Future Mobile Solution | The system design will support the future implementation of a mobile deli ordering solution. |
| Enable Use in Other Departments | The system workflow design will be generic enough to support usage of the system in other prepared foods departments. |
| Live Updates | System must ensure any updates to data such as the item catalog and splash screen images will propagate throughout the entire system in a reasonable amount of time. |
| Receipt Printing | When a customer places an order a receipt will be printed at the corresponding customer kiosk printer and a matching order ticket will be printed in the deli. |

#### **App Requirements (for both application modes)**

|  |  |
| --- | --- |
| **Feature** | **Description** |
| Application Initialization | The first time the application is run after being installed the user will be directed through the appropriate steps for initializing the application, including selecting the appropriate application mode. Once initialization has been completed successfully the app will launch into the appropriate mode and any time the app is closed and launched again it will automatically start into the same mode. |
| Server Communication | The app will send requests to and handle responses from the system server to provide updates to the server and also received regular updates regarding any changes to key system information such as the item catalog. |

#### **Customer App Requirements**

|  |  |
| --- | --- |
| **Feature** | **Description** |
| Place Order | Select items, specify quantity (in pounds or slices and type of slice), and submit the order for fulfillment. |
| Browse Items | Browse through the available category of items based on item category. |
| Search Items | Search for specific items by item name. |
| Edit/ Cancel Orders | Edit the items already in the current order (the “cart”) by either changing the quantity or removing that item. Also allow users to cancel their order entirely. |
| View Item Details | Allow the user to view the details for any item in the catalog including its price and a brief description. |
| Display Splash Screen | If the app is idle (i.e. has not customer interaction) for a sufficient amount of time, the system should ask the user if they want more time and, if no response is received, should display a splash screen consisting of a series of images retrieved from the server. |
| Dynamic Item Catalog Updates | Item catalog is kept up to date via polling the server for an updated time. If the catalog has been updated, the application will reload the catalog. |
| Dynamic Splash Screen updates | Splash screen sequence and images are kept up to date via polling the server for an updated time. If the sequence or content of splash screens have change, the application will reload the splash screens. |

#### **Employee**

|  |  |
| --- | --- |
| **Feature** | **Description** |
| View Active Orders | The employee UI will allow deli workers to easily view the active orders and mark orders as being complete. |
| View Order History | Employees will be able to view a list of all past orders within a certain time period. |
| View Order Details | The employee UI will allow users to view the complete details of any order. |
| Modify Items | Properly authenticated administrators will be able to view and make changes to any item in the item catalog, including changing the price, description, and marking the item out of stock. |
| Manager Log in | Managers will be able to log in with a PIN specified in the Admin ASP.NET service |
| Dynamic Order Updating | New orders will be pulled on a set interval from the server. When a new order is received a chime will play. |
| Dynamic Item Updating | In order to facilitate proper order contents, as well as the ability to edit items, the item catalog must be loaded and consistently updated with new changes if applicable. |

#### **Management Web App Requirements**

|  |  |
| --- | --- |
| **Feature** | **Description** |
| View Printers | The Management Web App will allow users to view all of the printers currently in the system. |
| View Kiosks | The Management Web App will allow users to view all of the kiosks currently in the system. |
| View Employees | The Management Web App will allow users to view all of the employees currently in the system. |
| View Splash Screens | The Management Web App will allow users to view all of the splash screens currently in the system. |
| View Coupons | The Management Web App will allow users to view all of the coupons currently in the system. |
| Generate Reports | Users will be able to generate weekly reports pertaining to the number of orders per kiosk, number of orders per item, and number of order per hour. |
| Add/Modify/Remove Printers | The Management Web App will allow users to add, remove, and modify printers in the system. |
| Add/Modify/Remove Employees | The Management Web App will allow users to add, remove, and modify employees in the system. |
| Edit/Remove Kiosks | The Management Web App will allow users to remove and modify existing kiosks in the system. |
| Change Kiosk Printer | Users will be able to set the printer that is associated with any kiosk that is in the system. |
| Add/Modify/Remove Coupons | The Management Web App will allow users to add, remove, and modify coupons in the system. |
| Activate/Deactivate Coupons | Users will be able to activate and deactivate coupons in the system. The active coupon will print on customer receipt. |
| Add/Remove Splash Screens | The Management Web App will allow users to add and remove splash screens in the system. |
| Modify Splash Screen Sequence | The Management Web App will allow users to modify the existing sequence of splash screens in the system. |
| Activate/Deactivate Splash Screens | Users will be able to activate and deactivate existing splash screens in the system. |
| Splash Screen Preview | Users will be able to click and view each splash screen that has been added to the system. |

#### **Inputs**

##### App Inputs

|  |  |
| --- | --- |
| **Input** | **Description** |
| Server Web Service Responses | After making a RESTful request to the server application, the response is sent back and consumed by the employee or customer app that made the request. |
| User Input | Customers or Employees will interact with the windows 8 device via a touch screen interface. |

##### Server Inputs

|  |  |
| --- | --- |
| **Input** | **Description** |
| Catalog Files (csv) | In order to generate the item catalog, three csv files are consumed by the server application, parsed, and processed into the deli item catalog. Changes to these files are automatically picked up by the system. |
| Client Web Service Calls | The server application exposes a RESTful API, which an external application can use in order to interact with the server and the system data. |
| SQL Server | The data for the system is contained in a Microsoft SQL Server database. This data is queried by the server application and that data is fed back into the server application for further processing and to pass it back to the client app or management web app. |

##### Management Web App Inputs

|  |  |
| --- | --- |
| **Input** | **Description** |
| User Input | A remote user or administrator of our system, can access the management app, through the use of a web browser. |
| Server Application Web Service Responses | The user makes RESTful api requests, via the web interface to the server application. The response of these requests are consumed by the management web app and displayed on the page to the user. |

#### **Outputs**

##### App Outputs

|  |  |
| --- | --- |
| **Output** | **Description** |
| Server Application Web Service Requests | The windows 8 employee and customer app makes RESTful service requests to the server application, in order to interact with the system. |

##### Server Outputs

|  |  |
| --- | --- |
| **Output** | **Description** |
| Server Application Web Service Responses (json) | When a RESTful service endpoint is accessed by an external application, the response of that service call is sent back to the external application via a json object. |
| Receipt Printing | When an order is submitted by the customer or an employee reprints an existing order, receipts are printed on both a customer facing zebra printer and an employee facing zebra printer. |

##### Management Web App Outputs

|  |  |
| --- | --- |
| **Output** | **Description** |
| Server Application Web Service Requests | The management web app makes RESTful service requests to the server application, in order to interact with the system and attempt to make configuration changes. |

#### **Human Operator Characteristics**

#### 

|  |  |
| --- | --- |
| **User** |  |
| Customer | The primary goal and purpose of this system is to decrease congestion, optimize the ordering process at Wegmans’ delis, and increase overall customer satisfaction. Therefore, the customer application UI and the system as a whole must be designed for easy and efficient use by a broad range of Wegmans’ customers. The customer application must also be highly available, responsive, and up-to-date (with regards to having accurate item catalog information). |
| Deli Employee | The primary interest of this type of user with regards to this system is being able to fulfill customer orders quickly and efficiently. Therefore the system should be easy to set up and maintain and require minimal interaction from this user to minimize the overhead associated with its use in the deli. |
| Deli Manager/Administrator | These users will interested in using the system to increase the overall operational efficiency of their deli. The design impact of this user group therefore includes the aforementioned impact of the customer and deli employee user interests. In addition to these considerations, the system must allow deli managers to perform basic administrative tasks related to the day-to-day operation of the deli quickly and easily. |
| System Administrator | This class of users will be responsible for performing administering the system as a whole. Such users will be interacting with the system at a store and/or (possibly) enterprise (multiple store) level. Therefore configuration and management of the system as a whole must be supported through a single, easy to use interface and all updates made to the system must propagate throughout the system in a reasonable amount of time. The system must also allow these users to view relevant system reports. |
| Developers | Responsible for the maintenance of and any future extensions/modifications to the system. The system should be designed and implemented in way which supports these tasks. |

### Constraints

#### **Technology Constraints**

At the beginning of this project, one major technology constraint was the use of a zebra printer. This was a technology that the customer has used in the past, and wanted us to integrate it into our solution. The Zebra technology is used to print receipts and has a great number of features available. However, this was a very difficult technology to learn and integrate into our solution. It does not have an official .NET library so we were forced to write custom code in order to interact with the Zebra printer and as such there wasn’t much documentation online to help with this integration.

Another constraint that was placed on our implementation was to run the server application on Windows Server 2008 R2. This is the common server technology that our sponsors were familiar with and since they want to be able to extend our application in the future, we needed to integrate with this technology.

Lastly, we knew that we were going to use a Microsoft SQL Server, in order to store our data. However, we didn’t want to interact with the data directly, we wanted to use an ORM to do this. Originally, we had planned on using the Entity framework to do this, but our sponsors wanted us to use an ORM that they were familiar with. As a result, it was suggested that we use NHibernate.

#### **Design Constraints**

From a design perspective there are few constraints.

The server must be able to load a catalog of some pre-determined format to be consumed by deli kiosks. This catalog is currently the summation of some .csv files on the server. The catalog design must be abstract enough to handle potentially many types of workflows ranging from Deli, to Subs, etc. This is done using a modified composite pattern that accounts for many types and certain items having specific attributes that augment them.

The deli kiosks must have an active network connection to the server as they directly consume server resources.

### Development Process

#### **Process Choice**

The team researched and discussed many choices for our process methodology. One such methodology was the traditional waterfall model, which was the methodology recommended by the RIT Software Engineering department. Initially, this appeared to be a logical choice for this project as it would give the project a set series of steps to follow with concrete deliverables at each step. However, the team did not like how rigid and inflexible this model is. The waterfall methodology also requires one phase to be fully completed before the following phase can begin (requirements, design, implementation, and then delivery). This means that project requirements have to be fully defined prior to moving into the architectural design phase and there is no opportunity to move back to a previous phase if problems with the requirements or design are discovered later on. Therefore, the team decided against using the traditional waterfall model, in part because the requirements were very unclear at the start of the project.

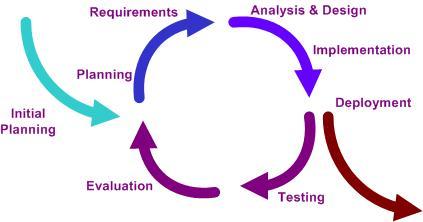
When the issue of choosing a process methodology was presented to the sponsors they mentioned that they use Scrum, a common agile methodology, for their technical projects. After researching this methodology the team decided that Scrum would not have been an appropriate choice either. While popular and very effective in the right circumstances, many of the core parts of that methodology would not have been practical or effective for this project, including the daily “stand-up” meetings for giving short status reports which every member of the team is required to attend. Given that the project team was comprised of students with busy and very different schedules, such daily meetings would not have been possible. In addition to this, the team did feel that they had enough experience with the Scrum methodology to use it effectively.

A third approach that was researched was the spiral methodology. This process was one that the team researched and almost selected, largely because of its heavy focus on risk assessment and management. This aspect was appealing at the time because the project requirements were still unclear at that point and likely to change and therefore, being able to assess and mitigate risks, including changes to requirements, was going to be vital to the success of the project. Additionally, this model produces working software much earlier than a traditional waterfall. However, the team decided against this process model due to the fact that none of the team members had any experience using this model.

Ultimately, the team decided to adopt an incremental build model, which is based on what is essentially an iterative waterfall methodology. This served as a good compromise between all three methodologies, providing the team more agile approach than the traditional waterfall methodology that incorporated the forward-looking risk management component of the spiral methodology as well as the incremental development and delivery of working prototypes while also avoiding the added administrative overhead and complexities of the Scrum and spiral methodologies. After presenting the methodology to the sponsors and explaining the rationale behind the choice, they agreed that it would be an appropriate choice for this project.

## 

## Process Description



The incremental build model utilizes an iterative development process, building off of previous progress. During each stage, more functionality and features were added until the product was feature complete and ready for final delivery.

Each stage started with a short planning phase for the upcoming iteration. This included reconsidering and re-evaluating the project risks as well as addressing any changes to the project requirements and/or design which needed to be made based on what had been learned during the previous iteration. The list of features to be implemented during that stage of development were also created during this time, with each feature being broken down into a set of smaller tasks which were assigned to specific team members and given an estimate of how much time that task was expected to require (very similar to the sprint planning portion of Scrum). These tasks were then entered into the Team Foundation Service (TFS) account for the project which allowed the team’s progress for the iteration to be easily monitored.

Once this initial portion of the stage was complete, the team moved into the implementation segment during which the planned features and tasks for that iteration were completed. This segment accounted for the majority of the time in each iteration, with the exception of the first iteration, which was focused primarily initial requirements elicitation and high-level system design.

This was then followed by short period reserved for testing the application with a focus on the newly implemented features. This allowed the team to ensure that each incremental prototype as well as the product as a whole was of a high quality. These tests included unit tests of specific, low-level features and functionality; acceptance testing with the sponsors to ensure the product being produced matched their needs; and usability testing to allow for continued improvement of the user interaction portion of the application. The results of these tests were then incorporated into the next iteration during the next planning phase.

Finally, at the end of each iteration, a working prototype was delivered to the sponsors at the end of each iteration and following the final iteration the completed product was provided to the sponsors, along with all of the accompanying documentation and information.

#### **Communication with the Sponsor**

Throughout this process the team maintained continuous communication with the project sponsors. This was accomplished through a variety of methods, including weekly, in person meetings with the sponsors. Prior to each meeting, the team sent an email containing the meeting agenda which was followed by a second email following each meeting. This email included a short summary of the meeting (in case any of the sponsors were not able to attend), a list of the action items from the meeting, and the team’s weekly status report (which was also reviewed at the start of each meeting). Additional emails were also sent as needed to coordinate the completion of sponsor action items or special events (such as the final presentation).

In addition to this, the sponsors also had a high degree of insight into the team’s activities and the overall project status through their access to both the team’s website, which contained the complete, up-to-date project documentation as well as the weekly team metrics charts, and the TFS account for the project which contained detailed information on all planned, active, and completed tasks for not only the current iteration, but also any other iteration (past or future). The sponsors were also provided with full access to the active project code base, allowing them to pull down the latest changes and then build, and run the solution, which they used for internal usability testing.

#### **Team Roles**

Dan Larsen - Team Coordinator/Leader; Dan provided oversight and management of the project as a whole, ensuring, among other things, that the weekly status reports were completed on time and that the project was progressing along the planned timeline.

Jared Schutt - Website Manager; Jared created and maintained the project website. He made sure that up-to-date copies of all project documentation were available through the website and he also created charts of the team’s core metrics (estimation accuracy and individual weekly project hours) which were automatically updated on the website as team members update their individual information.

Timothy Heard - Sponsor Communicator; Tim was responsible for creating and sending out the weekly meeting agenda at least 48 hours before each meeting. He also sent the meeting summary emails following each sponsor meeting and ensured any additional emails were sent to the sponsors as needed.

Warren Shaw - Meeting Scribe; Warren took detailed notes at every meeting and provided a summary of the main discussion points and action items from each meeting to ensure that all issues and tasks brought up during each meeting were properly addressed.

### Project Schedule: Planned and Actual

#### **Planning Our Schedule**

Our schedule was planned based on the number of iterations we wanted to complete as well as some specific departmental deliverables. In the end we decided on 6 iterations, with increasing functionality at each iteration. The first 2 iterations were more of a documentation iteration, with the end “product” being a well defined requirements and architecture document, respectively. Iterations 3, 4 and 5 were product iterations with increasing functionality coming out of each prototype. Iteration 6 dealt with mostly bug fixes, and wrapping up documentation, etc for the sponsors and department.

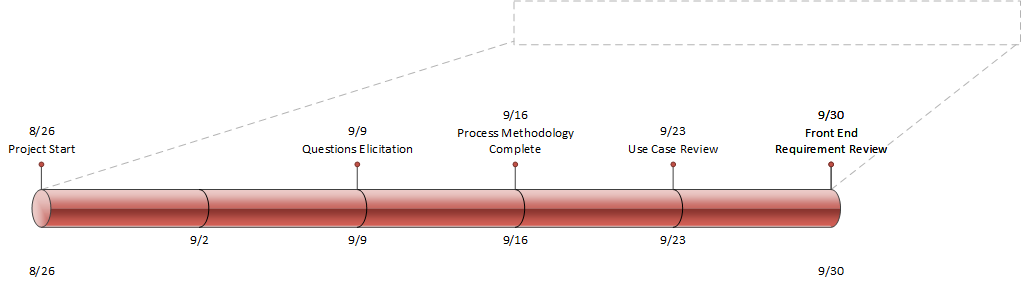
#### **Important Milestones**

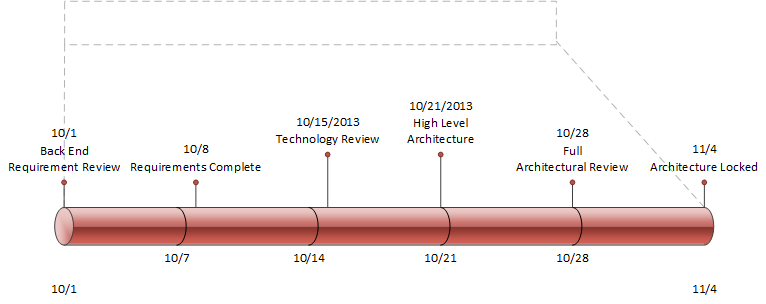
* Requirements
* Architecture/system design
* Prototype 1, 2 and 3
* Document delivery/ Wrap up

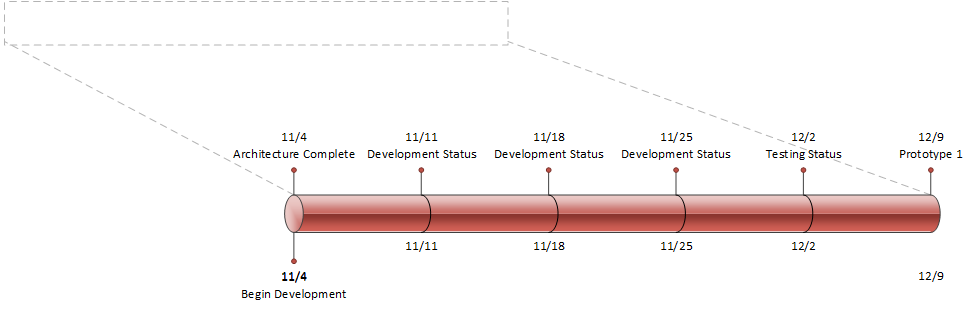
#### 

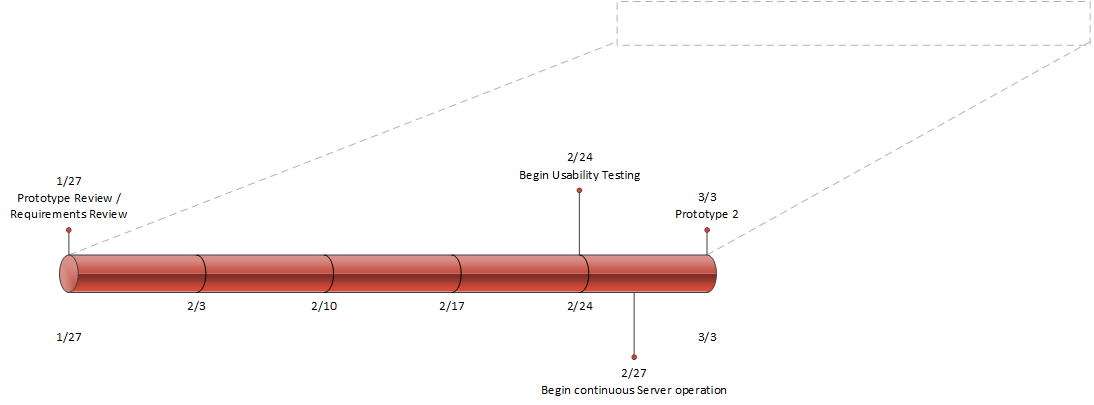
#### **Planned vs. Actual Schedule**

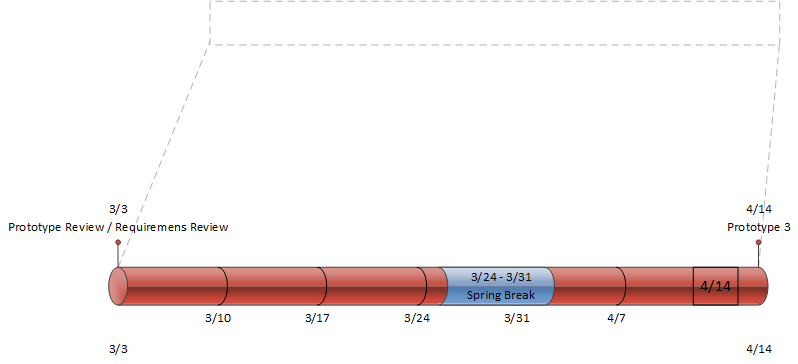
Our planned schedule was as follows. Each timeline is an iteration:

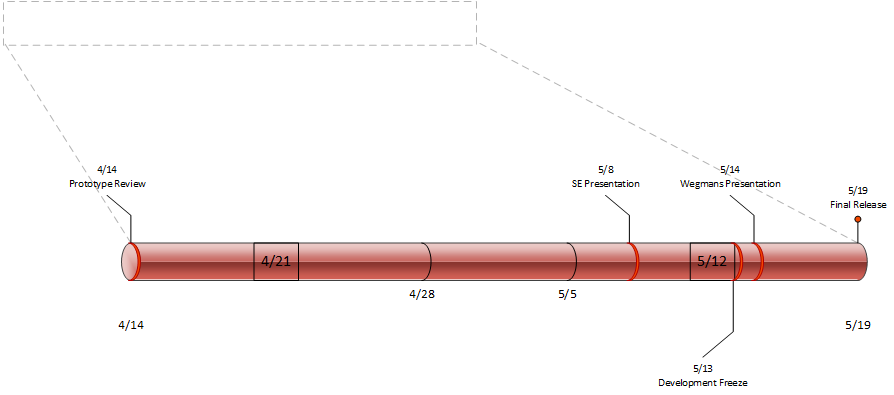








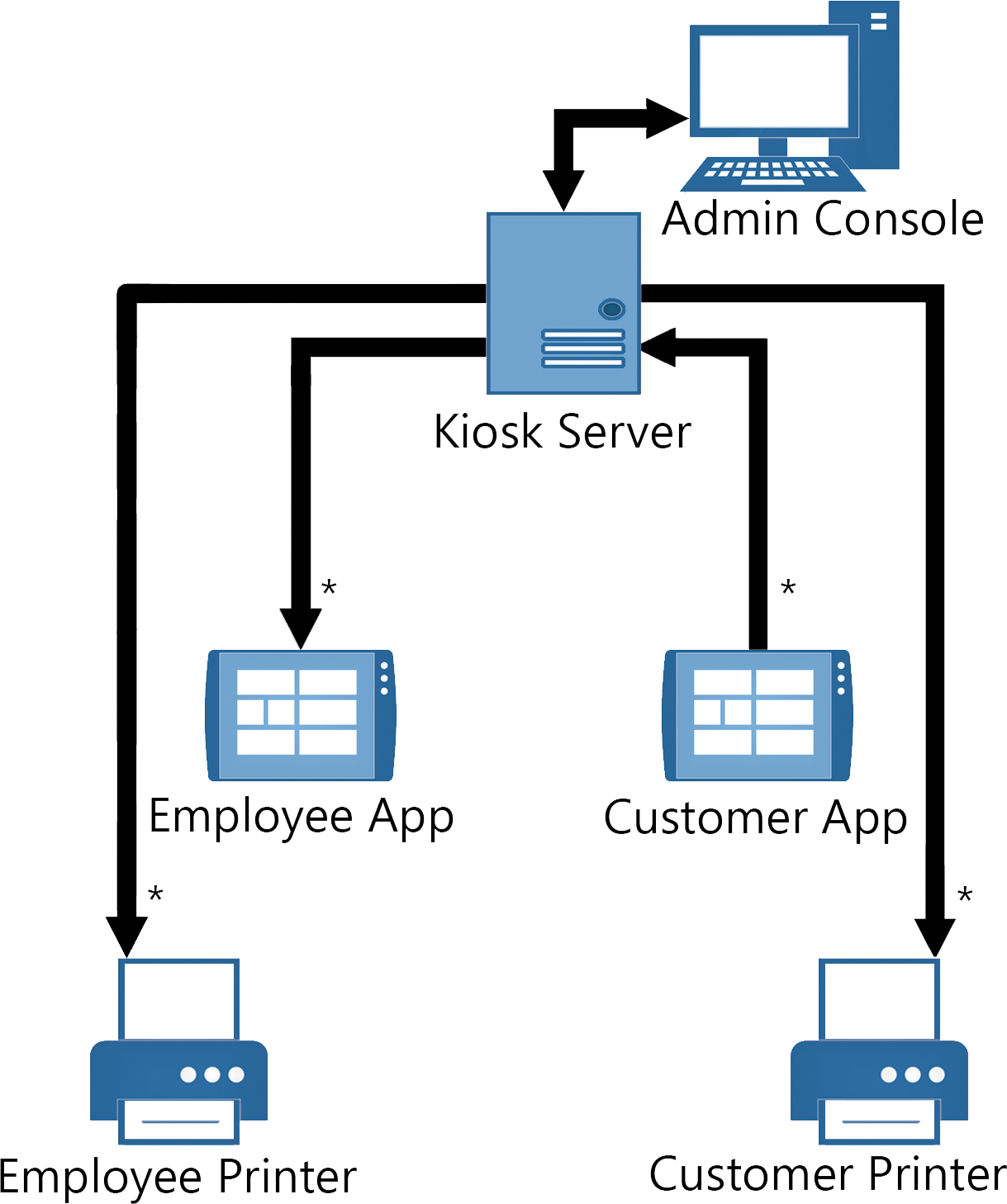




In reality we slipped our requirements and architecture deadlines by 1 week each. This pushed into our first prototype time. The rest of the schedule was on point and we had no further slips.

### System Design

#### **System Overview**



#### **Architecturally Significant Subsystems**

Client Side

*Common*

* Initialization
* UI/ Presentation
* Deli Item Management
* Server Gateway

*Customer Kiosk Only*

* Splash Screen Management
* Cart (order management/ creation)

*Deli Worker/ Admin Kiosk Only*

* Administration
* Order (History) Management

Server Side - Warren

* Reporting - This subsystem deals with pulling and aggregating records from the database, and then packaging them together in a meaningful structure, and then sending them off to be displayed
* RESTFUL Gateway - This is the top level system that handles routing all incoming requests to the right subsystems. It also has the logic for processing and constructing orders.
* Kiosk Data Management - This subsystem contains the logic for parsing and using a datasource of some sort and then taking that data and creating a meaningful item hierarchy that can then be sent off client side to be displayed.
* Printing - This subsystem has all of the logic and classes needed for managing different printers, and formating ZPL. Contains custom ZPL classes that handle the specific knowledge and logic needed for each ZPL command. Has interfaces to implement any future ZPL commands and any new printers that would be added to the system
* Data Access - This subsystem handles all interactions with the database and the Data access object layer where all of the information from the database is stored into new objects that can then be used by the system

#### **Design Rationale**

The team took many different design considerations into mind when developing the project. One major requirement from the sponsor was making sure the system was designed in a way that would allow for future departments being added. This was addressed by using the composite pattern to handle creating the different item hierarchy structures. By using the composite pattern, the team allowed the developer to create a tree N levels deep and also each root node having a connection to another node, allowing for a horizontal linking scheme as well to address workflows like subs where creation follows a more horizontal path.

The team also took into consideration the design of the system itself from a top-down level approach. Because of the exposed initialization RESTFUL calls, it is possible for this program to be distributed across many different stores, and have a higher up process hook in and manage everything. Because of the nature of a RESTFUL service, this program could also be integrated with mobile solutions because the data returned is device agnostic and could be used by whatever future app gets created.

Another way the team practiced good design patterns was with the introduction of a DAO layer or data access object layer in which all interactions with the database is handled. This keeps the user and developer away from writing any raw sql queries and instead lets the NHibernate framework deal with it.

### Process and Product Metrics

#### **Metrics Tracked**

* Task Slippage: the number of tasks needed to be done vs the number that were actually done.
* Estimation accuracy: overall estimation of tasks versus actual time
* Effort by activity: how much time was spent completing each task

#### **Results**

* Task slippage: There were a total of 6 Tasks that slipped during our development. These tasks were mostly development items that were moved. They were:
  + Requirements Document - Slip Time: 1 Week.
  + Architecture Document - Slip Time: 1 Week.
  + Modifying Items - Slip Time: 1 Iteration
  + Admin System UI & Login - Slip Time: 1 Iteration
  + Order Re-printing - Slip Time: 1 Iteration
  + Catalog Custom Attributes Rework - Slip Time : 1 Iteration
* Estimation accuracy

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Total Estimated | Actual Total | Result |
| Dan | 329.5 | 351.5 | Under Estimated 22 Hours |
| Jared | 360 | 345.75 | Over Estimated 14.25 Hours |
| Tim | 264 | 194 | Over Estimated 70.75 Hours |
| Warren | 272.5 | 294.6 | Under Estimated 22.1 Hours |

* Effort by activity
  + Average = 2.5 Hours
  + Min = 10 minutes
  + Max = 20 hours

#### **Interpretation**

Task slippage was overall not bad. We only slipped actual deadlines by a week at most, and did not slip later on in development. We also only put off lower priority items, so not much was effected.

Estimations were -reasonably- accurate and useful. Overall our biggest slip in estimations was 70 hours, which was about ⅓ of our actual time.The amount of effort put in (by week) was as expected. The spikes corresponded to the completion of major tasks and dips normally corresponded to a team member having to take care of other tasks/issues outside of senior project (interviews, a wrecked car, etc.)

WHen it comes to effort by activity, our average task time was good, but there were some outlying tasks (the 20 hour one for example) that maybe should have been split up more. That was a one off task though, and if we account for that the next highest task was 13, and then 10 hours - Still large but not nearly as bad.

### Product State at Time of Delivery

#### **Completed Features**

#### All the system features committed to and agreed upon by the development team and the project sponsors have been fully implemented at this point. See the ‘Basic Requirements’ section for further details.

#### 

#### **Incomplete Features (i.e. Future Goals)**

#### Although all the system features for this project have been fully implemented, there are some additional features which could be implemented in the future which are listed below.

|  |  |
| --- | --- |
| **Feature** | **Description/Reason** |
| Employee Tracking | Monitor and report on employee performance with regards to the number of orders each employee completes and how long they take to complete each order. In addition to tracking the performance of individual employees this would also allow metrics to be generated for overall deli and store efficiency, changes to that efficiency over time, and even order time estimation based on the historical average time a particular deli takes to complete orders (taking into account the number of active orders already in the system for that deli). This would require having each employee indicate when they started an order using their unique employee ID and then also confirming when they have completed that order. At present, the system allows employees to acknowledge that an order has been completed; however, this is performed anonymously (no concept of which employee completed the order) and this is not reported to the server. |
| Shopper’s Club Integration | Allow customers to optionally scan or enter their shopper’s club card number when placing an order. This would allow the system to record what each specific customer ordered and would allow for additional features such as remembering favorite/common orders, making order and/or item suggestions based on their order history, and special offers. This information could also be used to send customers notifications via a text message and/or email when their order is complete. The system does not currently have any built in support for any of these features. |
| Printing Adds on Receipts | In addition to coupons, advertisements for different products and items could also be printed on the customer receipts. This is not currently supported by the system, but the existing functionality for printing images (the Wegmans’ logo) and printing coupons (which are managed by the central server) would provide a good basis for extending the system to include this functionality. |
| Mobile Application Integration | Extending the system to allow customers to place orders from their mobile device. Although this has not been implemented at this time, the server has been designed to support such applications and the server interface is completely agnostic/independent of the type of device which is using it, so extending the system to include this feature should only require implementing the mobile client application(s). |
| Multiple Department Workflows | Extending the system to include workflows for other prepared food departments beyond the current deli workflow (such as subs, pizzas, cheese, etc.). The system design explicitly supports such modifications through the item catalog/workflow structure. However, the customer application interface is currently designed specifically for use with the deli workflow and therefore the UI would have to be redesigned and modified to support any new workflows which are implemented. |

#### **Unplanned Completed Features**

|  |  |
| --- | --- |
| **Feature** | **Description** |
| Dynamic/Animated Splash Screen | Display a series of images with transitions between them for the splash screen (instead of displaying a single, static image). |
| Wegmans’ Logo on Receipts | Print the Wegmans’ logo on the top of all receipts and order tickets printed. |
| Coupons | Allow properly authenticated system administrators to create, edit, and activate different coupons which will then be printed onto the customer receipts for qualifying orders in the form of a description and a coupon barcode. |
| System Administration Web Console | Enable system administrators to configure and monitor the system through a remote web console. |

### Project Reflection

#### **What Went Well**

There were many things that went well for us on this project. Our team was very cohesive and worked very well together. All of our team members were very dedicated and we didn’t have to worry about any of our members not doing something that they had committed to. as a result, we were able to make all of the major deliverables and milestones that we originally agreed to and more. We all feel that this was one of the best teams we have ever been a part of and it was a major reason for our success.

Another thing that went well for our team was our project process. Although we didn’t strictly follow our chosen process to the letter, it still worked very well for us. We were able to make a very realistic schedule and as we mentioned above, we were able to hit all of our major milestones. In order to do this, we utilized TFS and its ability to assign features to different releases, tasks for those features, and then assign and track the progress of those tasks, all on the web interface. This process in general, worked very well for our team.

Additionally, our communication with our sponsor was another aspect of our project that went particularly well. We had the privilege to be working with sponsors that had some technical background, so they were very understanding of our suggested timelines and whenever we had issues with what they had originally wanted. They were also available to meet with us on a weekly basis and were very responsive to our communications between meetings. Whenever we sent them emails, they were very prompt with their responses which helped us to stay on track with our schedule and gave them the necessary visibility on our project that enabled us to be as successful as we were.

Lastly, we were able to meet the expectations, not only of our sponsors, but as a team. After giving our onsite presentation, the sponsors and their colleagues let us know that they were very pleased and impressed with the work that we had done. One of their colleagues went as far as to get our LinkedIn information and tried to offer us jobs. Additionally, as a team, we were all very satisfied with how the project turned out. Ultimately, we were able to develop a dual windows 8 app, a server application, integrate with two zebra printers, and developed a administrative web app. It was a lot of work, but we really enjoyed it and were pleased with the outcome.

#### **What Went Wrong**

Our project enjoyed alot of success. As such, there wasn’t much that went wrong. The only thing that we would like to point out was that our requirements elicitation could have went smoother than it did. It seemed to drag on for a few weeks longer than we would have liked. We feel that it could have been shorter and more efficient. Instead of taking so much time to figure out which technology to use, we could have had this choice made before the project started. Originally, when we came together as a team, before we had a meeting with our sponsors, we had suggested to each other that it would make sense to use a full .NET stack with a windows 8 app. Instead of just going with that choice from the start, we went through a technology study for a few weeks and still ended up choosing the full .NET stack with the windows 8 app. We feel we could have avoided this extra churn if we had just asserted ourselves and selected this technology from the start.

Additionally, during this requirements elicitation it seemed that there was a lot of back and forth between the sponsors and our team. We were essentially starting from scratch when we began requirements elicitation and it could have been more efficient if we had an upfront schedule for the amount of time we would use for requirements elicitation and stuck to it.

#### **What Would We Change**

One thing that team would have changed about this senior project would have been the time available for doing the project. Going into the project, we assumed that senior project would be something that would consume all of our time and would basically be like a “job” for the duration of the project. However, it seemed more like a regular SE project that just went on for two terms. Something that we would suggest for future projects and would have liked for our project, would have been to have the first semester be all about gather requirements and developing a design for the system. This is nothing new, as many of the teams, ours included, did this for the first semester. However, we would suggest that the second semester be treated like a co-op. All of the senior software engineering students only be allowed to take senior project and at most, one other class during that second semester. Then the teams would have the time available to treat the senior project as a real world project.

We feel that this would greatly prepare the students for the real world, by giving a project that would work on, forty hours a week, with their team, just like they would in a real world setting. This would allow teams to get much more work done on the project and would ultimately mean that these projects would production ready, by the end of the second term.