Wegmans Central Affinage System

Team
Cheddar Luck Next Time
Chris Cowdery Corvan
Daniel Joseph
Sherman Lee
Miles Malerba
John Van Gundel
Bob Loblaw

Sponsor
Wegmans
Cathy Gaffney
Cheryl Polaski
Sue Thibodeau
Richard Vander Horst

Faculty Coach
Kenn Martínez
1. Project Overview
2. Basic Requirements
3. Constraints
4. Development Process
5. Project Schedule: Planned And Actual
6. System Design
   6.1 Architecture
      6.1.1 Utilized Patterns
         6.1.1.1 Layers
         6.1.1.2 Model-View-Controller
         6.1.1.3 Repository
         6.1.1.4 Inversion of Control
         6.1.1.5 Unit of Work
      6.1.2 Primary Representation
3. Use Cases
6.3 Challenges
   6.3.1 Database Technologies
      6.3.1.1 ASP.NET MVC 3 Consistency
7. Process and Product Metrics
8. Product State at Time of Delivery
   8.1 State of the Product
   8.2 Missing Features
   8.3 Unplanned Features
9. Project Reflection
10. References
1. Project Overview
In its current form, each Wegmans retail store is responsible for ordering, processing, storing and selling cheese. Each store maintains contracts with vendors around the world and is responsible for maintaining the correct amount of inventory in the store - especially in case where turnaround time from order placement to delivery takes 6-8 weeks. Moreover, each store has to have specialized staff for each store strictly for cheese and this training and headcount needed to maintain this level of quality. Wegmans did not find to be ideal.

To mitigate all these issues, Wegmans is beginning construction of a Central Affinage Facility. This facility would centrally manage, order and ship all cheeses for all Wegmans retail stores. By centralizing the process, Wegmans is able to cut down order turnaround time down to 2-3 days, require less staff, ensure consistent quality across all products and ship the ideal number of cheeses to individual stores.

The Central Affinage System (CAS) was requested by Wegmans as a software system to manage all operations of the facility. More specifically, the system would need to track every operation performed on cheeses from when they arrived at the facility to when they are shipped to stores. The system must manage quality control, recipe steps, room cleaning, cheese moving and all administrative operations required to create, update and delete the aforementioned tasks.

The delivered system would be accessible in full via a web application with some parts also accessible via a mobile application running on a Windows Mobile wireless barcode scanner. Although the system would not integrate with Wegmans’ internal systems directly due to scope and disclosure reasons, the system was designed in such a way those elements could be substituted and integrated when Wegmans saw fit.

The feature-complete system was delivered to the sponsor Friday, May 18, 2012, and the sponsor was very pleased with the end product. The system has three known bugs at the time of this writing, and their current status and information about the defects have been reported to the sponsor. Ultimately, the project was considered a great success by all parties and everyone was very pleased with the end outcome.

2. Basic Requirements
The Central Affinage System will be responsible for managing all cheeses that go in and out of the Central Affinage Facility. In addition to managing the cheeses, it will also track of information eternal to the facility, such as cleaning frequency of the cheese caves, the statuses of cheeses and where they currently are in their recipes, and temperature logging of every cave. Cheeses that are received at the Central Affinage Facility will be processed into the system and go through a series of quality control steps before actually being accepted into the system and undergo processing.
The basic workflow of the central affinage facility that the central affinage system must be able to log and keep track of is in the following diagram:

*Figure 2.1: Basic Workflow Diagram*

1. Vendor Delivery
2. Quality Control
3a. Accept Delivery
3b. Reject Delivery
4. Assign to Group
5. Move to Cave
6. Perform Affinage
7. Move to Order Fulfillment
8. Move to Shipping
9. Load Shipment
10. Ship to Store

Not only does the Central Affinage System have to keep track of the cheese that move through the system, but it is also responsible for the cheese affineur to be able to manage the different types of product types, cheeses, vendors, and groups that are used to track the cheese within the facility.

The Cheese Affineur and the employees of the Central Affinage Facility will be the main users of the Central Affinage System. They will interact with the system through either a web application on a computer or laptop, or a mobile handheld 9090 device. The web application and mobile handheld device are both able to do the same actions within the system. Some actions that can be done both on the handheld and web application are accepting a delivery of cheese, logging that a cheese is undergoing steps in the recipe, and when a ship is being cheesed from the
facility.

3. Constraints
The Central Affinage System was limited by time, where we only had two quarters to fully elicit the requirements to release and hand off to the customer in that time. Though we planned our schedule to fully accommodate all aspects of the software development process to release a fully functional application, more time would have been good to completely implement all the features. Meeting with the sponsors was also only limited to once a week on Thursdays from 4:30 to 6 PM normally. These meetings did not occur during the breaks that RIT had which was the Christmas break that lasted for three weeks and Spring break that was only one week. The sponsors could easily be contacted though through e-mail at any time.

Technology wise Wegmans wanted an C#, ASP.NET web application with the back-end database being implemented with Oracle 11G. In order to connect and map the front-end to the database we used Fluent nHibernate. There was also the mobile application that was required to be developed on a Motorola MC9090 device of which we only had one, limiting development to only one developer. Not only did we just have one 9090 to develop on, it also required the usage of an older version of Visual Studio, which was our primary IDE. The web application was developed in Visual Studio 2010, whereas the mobile was limited to being developed in Visual Studio 2008. The rest of the technologies used in the Central Affinage System were researched by the team since the sponsors gave us the freedom to use whatever we wanted as long as it was still a C# ASP.NET application that worked with an Oracle back-end.

4. Development Process
Our team followed the PMI process; this is a waterfall-like process that is split into five main phases: initiate, plan, develop, implement, and evaluate. The initiate phase consists of documenting project vision and scope, defining high-level milestones, and ensuring access to appropriate documents. The plan phase is when requirements gathering and system design occurs. The develop phase consists of implementing the system as well as unit tests. The implement phase is when the system will be deployed and acceptance tested. And finally, the evaluate phase is when the team will reflect on the project in order to gain insight for the next project.

In the initial project description Wegmans had indicated that they wanted us to use this methodology, however, in later discussions they informed us that it was just a suggestion, not necessarily a requirement. The team decided to stick with PMI, but at the sponsor's request we did make some changes to better suit it for this project. The sponsors wanted to be able to give us frequent feedback about our progress, but this was difficult with a waterfall-like process model. To allow the sponsors more opportunities to provide feedback we adapted the process by adding in iterations. These iterations made it a more iterative process and allowed the sponsors to see the results of our work and give feedback at the end of each iteration.

Our process included two weekly meetings on Tuesday and Thursday, with additional meetings...
planned as necessary. The Thursday meeting was designed to be a conversation with the sponsors. The weekly sponsor meeting included gathering requirements and feedback, updating the sponsors on our progress, and answering any questions either side may have had. Additional communication with the sponsors was carried out through email. The team set up three mailing lists: one for team member only, one for team members and the faculty coach, and one for the team members, faculty coach, and sponsors. If the team or sponsors had any questions prior to the weekly meeting, we could use the mailing lists to communicate. In order to simplify communications the team designated Chris as the main point of contact with the sponsors. This made it easy for the sponsors because they knew who to contact and made it easy for the team because rather than everyone having side conversations we could go through Chris, who would keep the whole team in the loop.

The team identified roles based on the team member's strengths and preferences. Fortunately our team was diverse enough that this worked out well for us. Chris was the team leader, UI designer, and man in charge of configuration management. Dan was the team's mobile developer. John was responsible for the architecture and design. Miles and Sherman shared the role of primary web developer, and Miles was responsible for testing. The role divisions worked out very well on our team. Each member became an expert in their area and was able to help the others when they had questions. Everyone enjoyed their role and there were no conflicts.

5. Project Schedule: Planned And Actual

During the initiate phase of the project the team first compiled a list of deliverables that we would need for both Wegmans and the Software Engineering Department and when they were due (Table 5.1). Based on this list, we came up with a high level overview of the project schedule. This schedule outlined when we planned to finish each phase of the PMI process, how many iterations would be in each phase, and which deliverables were do at the end of which iterations.

Table 5.1: List of Deliverables

<table>
<thead>
<tr>
<th>Deliverable</th>
<th>Due</th>
<th>For</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time/Effort Tracking</td>
<td>Updated</td>
<td>SE Dept</td>
</tr>
<tr>
<td>Project Synopsis</td>
<td>12/2/2011</td>
<td>SE Dept, Wegmans</td>
</tr>
<tr>
<td>Project Website</td>
<td>12/4/2011</td>
<td>SE Dept, Wegmans</td>
</tr>
<tr>
<td>Project Plan, Schedule, and Process Methodology</td>
<td>12/16/2011</td>
<td>SE Dept, Wegmans</td>
</tr>
<tr>
<td>Definition</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Technical Requirements Document</td>
<td>1/19/2012</td>
<td>Wegmans</td>
</tr>
<tr>
<td>Interim Peer Evaluations</td>
<td>1/20/2012</td>
<td>SE Dept</td>
</tr>
<tr>
<td>Architecture Diagram</td>
<td>2/2/2012</td>
<td>Wegmans</td>
</tr>
<tr>
<td>High Level Design Document</td>
<td>2/2/2012</td>
<td>Wegmans</td>
</tr>
<tr>
<td>Interim Project Report</td>
<td>2/14/2012</td>
<td>SE Dept</td>
</tr>
</tbody>
</table>
During the plan phase, we further refined the schedule by assigning specific features and pieces of functionality to different iterations within the development phase. The plan we came up with called for the first three iterations to be used for "bootstrapping" - essentially getting ourselves acclimated to the technology and getting all of the different technologies to play well together. The fourth iteration was for authentication; this is when we implemented users and roles as well as the ability to authenticate against LDAP. Iterations five and six were scheduled for development of the core cheese processing functionality. Iteration seven would be the implementation of the reporting features. And finally, iteration eight would be a polish iteration where we would fix any remaining bugs and make sure the UI looked perfect.

Wegmans requested that we make more detailed plans than just the main areas of focus, so we also created charts that showed when we planned to do each feature. These charts included feature dependencies and optimized the number of tasks we were working on at any one time, so there was always something for each team member to do.

There were two points where we had to change our schedule (Figure 5.2). The first was during iteration five of the development phase. At this point the team realized that the cheese processing operations we had selected for that iteration were going to take a little longer than we initially anticipated. After talking to the sponsors the team agreed the best course of action was to add a week to iteration five, thus extending our development phase by one week. The second was at the end of iteration six. The team presented the results from the iteration, but based on the feedback from the sponsors we realized that we had not fully understood the requirements. The changes needed to make the system conform to the new requirements were non-trivial and required a large refactoring. The refactoring put the team about two weeks behind where we intended to be. After a long meeting with the sponsors where we hashed out the remaining scope, the team decided that the best thing to do was merge the reporting and polish iterations into a single longer iteration that encompassed both. At the same time we extended the development phase by two weeks in order to allow time to make
the changes. This was not ideal for the team as it meant we had to many of the activities for the implementation and evaluation phases while still working on the development phase. However it was necessary in order to get the system to a point where we felt we had satisfied the customers needs.

Figure 5.2: Original And Revised Project Schedule

|       | W1  | W2  | W3  | HB1 | HB2 | HB3 | W4  | W5  | W6  | W7  | W8  | W9  | W10 | WF  | SB  | S1  | S2  | S3  | S4  | S5  | S6  | S7  | S8  | S9  | S10 | SF |
|-------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| Initiate | Iteration 1 |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| Plan     | Iteration 1 | Iteration 2 | Iteration 3 | Iteration 4 |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| Develop  | Iteration 1 | Iteration 2 | Iteration 3 | Iteration 4 | Iteration 5 | Iteration 6 | Iteration 7 | Iteration 8 |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|          | Bootstrapping |     |     |     | Auth |     | Cheese Processing | Reporting | Polish |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|          | Iteration 1 | Iteration 2 |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|          |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |

6. System Design

6.1 Architecture

6.1.1 Utilized Patterns

6.1.1.1 Layers
A layered architecture was the primary pattern used for the application. By using a layered approach, we were able to divide our application into four key functional areas – composition root, presentation, business, and persistence. Due to the uncertainty regarding what technologies Wegmans would keep in the system after hand-off, we needed an architecture that allowed for a high amount of modifiability with a relatively low cost. In particular, we were unsure about what database technology Wegmans would actually be using in production. By isolating all of our persistence logic and object-relational-mapping tools in their owner layer, we enabled Wegmans to later replace those technologies without having to heavily modify the application.

6.1.1.2 Model-View-Controller
The use of the model-view-controller pattern was implied due to our use of ASP.NET MVC 3. This pattern allowed us to keep our views and models isolated from our control flow logic, which prevented our presentation layer from becoming bloated with unnecessary concerns. We were very effective in implementing this pattern, and managed to avoid the use of any god-controllers
within our application.

6.1.1.3 Repository
We used the repository pattern to facilitate CRUD operations between business entities and our underlying data store. This was achieved using the built-in mapping classes provided by Fluent nHibernate combined with the expressive querying abilities of LINQ.

6.1.1.4 Inversion of Control
Using an inversion of control container, we were able to abstract our component dependencies as interfaces, which in turn allowed us to modify implementation without the need to change each dependent, so long as our interface contracts were maintained. Dependency injection was enabled by the out-of-box capabilities of our container, which allowed for convention-based automatic mapping of interfaces to the classes that implemented them.

6.1.1.5 Unit of Work
The unit of work pattern allowed us to ensure that atomic operations were consistently performed on our underlying database by handling committing and rollback of all transactions on a per-HTTP request basis. Utilizing our inversion of control container, we were able to build this support directly into the life-cycle of the existing ASP.NET MVC 3 HTTP session, greatly reducing implementation complexity.

6.1.2 Primary Representation
Figure 6.1: System Architecture
6.2 Use Cases

*Figure 6.2: System Use Cases*
Login excluded, the application is divided into four primary sections. Each section is contained in a menu in the navigation. Each section has subsections (also accessible via a sub-menu), these subsections are denoted using dashed lines where menu elements are denoted using a solid black line and “Menu” text bisecting the arrow.

The Cheese Handling section (Yellow) is concerned with operations on a container of cheese. These operations can also be accomplished using the handheld 9090 device, but they are also accessible on the web as well in this section.

The Reports section (Blue) is concerned with producing HTML and PDF reports of both the cheese history (for health reasons) and cheese tracking (for recipe analytics and history). Reports for each of the items in the Dashboard are available in this section.

The Workbench section (Green) is concerned with creating, reading/viewing, updating and deleting the core entities of the system. These entities are typically involved in the “Cheese Handling” section of the system. For example, in this section you can create a cheese type, then create QA rules for that type of cheese, wash types for the new type of cheese, and also recipes for the new cheese. At a bare minimum, each cheese type must have at least one recipe (the rest of the entities are optional). Moreover, because many of these processes are interconnected, the Workbench features a “Create Cheese” wizard where the system will walk
the user through the creation of a cheese type and its associated QA steps, wash types and recipes (wherever applicable).

The Administration section (Red) is concerned with adding/removing users to CAS and also setting roles for the users in the system. Only users that have the admin role can view this section.

6.3 Challenges

6.3.1 Database Technologies

The initial proposal from Wegmans called for the use of Microsoft SQL Server. In accordance with this, and after initial confirmation from Wegmans, the team began researching ORM technologies and started prototyping the database structure. Shortly after deciding on an ORM (PetaPoco), Wegmans pointed out that they did not use SQL Server locally, and therefore would need to rewrite our persistence layer after the project was finished.

To avoid handing off an unusable out-of-box product to Wegmans, we decided to drop PetaPoco and SQL Server in lieu of Fluent nHibernate and Oracle. This required a rewrite of the work we had done so far, which was further complicated by a group-wide lack of experience working with both new technologies. As a result of this change, we lost roughly two weeks of development time as we became familiar with our new technology stack.

6.3.1 ASP.NET MVC 3 Consistency

As a result of our team only having two members that were familiar with ASP.NET MVC 3, there were several inconsistencies that occurred in our code-base, such as:

1. No use of strongly-typed view models
2. No use of built-in helpers
3. No validation on form fields
4. Inconsistent coding styles for C#, HTML, and JavaScript
5. Use of in-line CSS and JavaScript

In order to address and resolve these issues, a number of refactorings were made. Every view was updated to use view models, which in turn allowed built-in helpers and validators to be quickly implemented and added, respectively. There was also a concerted effort by all team members to keep JavaScript and HTML consistent, which helped alleviate differences in coding styles. The majority of in-line CSS and JavaScript was transferred to external files. This came with complications, as several AJAX routes were broken in the process, resulting in further refactoring.

The above refactorings were largely performed manually. However, via the use of the ReShapper productivity tool, we were able to automate a number of refactorings - most importantly C# consistency and changes made to Razor views. ReShapper allowed us to set up a central coding standard and reformat all of our code automatically, resulting in a more uniform coding style. It also provided more advanced refactorings for controller renames, which automatically addressed broken routes in our Razor views that would have otherwise needed to be manually identified.
7. Process and Product Metrics

Although our team did not place as much emphasis on metrics as other teams have in the past, we did use a handful of process, productivity and quality metrics to gauge the progress of the system. The metrics were as follows:

- **Defects per KLOC (Quality)**
  Defects per 1000 lines of code (KLOC) helped us gauge the relative stability of our build. As our entire suite of unit tests were run upon each check-in, we usually had pretty good visibility on what areas of our system contained defects. Using defect density we were able to better partition our efforts and focus on subsystems with a high number of defects.

  Defects per KLOC was calculated on a per-checkin basis. Upon every checkin, the Continuous Integration server would automatically calculate the defect density and inform us of areas which needed the most amount of attention. Although helpful to document, the team as a whole already had an decent grasp on what needed doing, and moreover, when defects did roll in, the team focused on feature completeness instead of fixing defects. The team feels that this was more beneficial to us ultimately as Wegmans was very much a “I know it when I'll see it” client, and providing visual representations of the features was essential as a result. In other words, if Wegmans didn’t like a certain functionality, it didn’t make sense for us to waste time on quality if the sponsor wanted to overhaul a feature in progress. This sentiment was also captured in the sponsor (Rich)’s quote “We’d much rather have a successful demo with functionality over full coverage on testing." However, after a feature was functionally complete, time was then taken to ensure high quality - part of which entailed ensuring the feature contained low defects per KLOC.

- **Digressions per Meeting (Productivity)**
  During every meeting we would informally maintain a tally of how many digressions (in conversation) occurred during a meeting. Although the team got along extremely well, this also meant that the probability of us getting derailed on topics not related to the project was much higher. In the short term, keeping track of digressions per meeting helped us recognize when a digression happened so we could focus more on the topic at hand (when it was recognized). In the longer term, it helped us analyze factors that contributed to losses in productivity so they could be mitigated in the future. Digressions per meeting was calculated on a per-meeting basis. This metric is also affectionately referred to as “Reichs."

  The number of Reichs was captured in a Google document for posterity and reflection, however no formal analysis of the metric was conducted.
Estimation Accuracy (Process / Productivity)

Wegmans was very interested in receiving estimates of workload from the team, and as a result, it was very important that we produce high accuracy estimates for the sponsor. By tracking the actual vs. estimated time for a task, we were better able to come up with estimates which benefited both us and the sponsor. Having more accurate estimates helps us better forecast our workload and gives the sponsor higher confidence and visibility on the project’s progress. Over time the estimation accuracy became more accurate and took on the form of a dampened sinusoidal function.

Although estimation accuracy was captured on a per-task basis, estimation accuracy was reviewed on a per-week basis. The metric was captured in a Google document updated nightly by members. Capturing estimation accuracy was without question one of the most useful metrics we captured as a team as so much of our project’s success was centered around accurate predictions of work remaining - especially towards the end.

The charts of our predictions are available below:
8. Product State at Time of Delivery

8.1 State of the Product
The feature-complete system was delivered to the sponsor on Friday, May 18, 2012, and the sponsor was very pleased with the end product. The system has three known bugs at the time of this writing and their current status and information about the defects have been reported to the sponsor. Ultimately, the project was considered a great success by all parties and everyone was very pleased with the end outcome.
8.2 Missing Features

Although the system we delivered to the sponsors was fully feature complete to our agreements, there remained some functionality which the sponsor had expressed some interest in. The features they would be implementing (or finishing) internally are as follows:

- **Extended Reporting** - More reports than just “Cheese Inventory Report”, namely:
  - 30 day shrink report
  - Work in progress report
  - Labor hours used report
  - Rejected cheeses report
  - Total inbound / outbound report

- **Increased Dashboard functionality**
  - Seamless (Ajax) updating
  - Calendar tasks
  - Quick Number Widgets did not have:
    - Recipes currently in process
    - Recipes finished
  - Charting
    - Only implemented the current number of cheeses

- **Form validation** - Some of the system’s logic (especially in the user interface) proved to be too time consuming to validate in the remaining time we had left for the system. Although the system will not accept invalid inputs, having bad inputs highlighted upon entry (and providing a rationale for why the input was considered bad) would be ideal. Most of these changes would be reflected in custom JavaScript where unobtrusive validation could not be performed, but some would exist server side as well.

- **Simplified date/time pickers** - Instead of parsing a pre-populated text field with a DateTime object as a string, offer date pickers (a la jQuery UI) that provide a cleaner interface for entering dates.

- **Non-serial recipes** - Having the ability to have concurrent steps in a recipe. This was currently mitigated by having a recipe step list (in text) all steps that need to be performed as one step, but having this represented in the system model would be ideal.

8.3 Unplanned Features

Several released features of our system started as unplanned nice-to-haves. Having a centralized deployment system for the mobile application made sense. Instead of having to deploy the application to each mobile through cradling, the web application hosts a bundled copy of the application which can be easily downloaded and installed completely wirelessly. Quick numbers also started as an unplanned feature but when demonstrated to the sponsor,
they agreed that it made sense to include in the dashboard.

9. Project Reflection
The project both the sponsor and team consider to be an overwhelming success. The primary problems encountered in this project were pertinent to requirements elicitation and scope. During the project the needs of the system weren’t explicitly known which led to some thrashing and rescheduling, however, the system ultimately was delivered on time with full feature completeness.

In hindsight, we would have been very well served as a team if we provided low-fidelity mockups of our features right away to the sponsor before development began in earnest. By eliciting feedback from the sponsor before development began, we’d make sure that our development efforts were focused in the right direction. Moreover, stating assumptions early and often would help make sure that we were implementing the right functionality (e.g. avoiding the not individually identifiable cases misunderstanding). Lastly, having more granular task tracking may have helped clear up ambiguities of who was assigned to specific issues. Although this happened infrequently, having that extra amount of clarity would have helped clear up misunderstandings when they did occur.

Overall the team learned a great deal, gave it their all, and produced a high quality product that surprised and delighted the sponsor. The project was not without its learning opportunities, however, all things considered all parties were very pleased with its performance.

10. References


