A manufacturer of industrial coffee roasters wanted a complete overhaul of their software platform. After receiving a seven figure quote from GE, they found Justin Kuo, founder of Sparkle Technologies, who performed the task at a fraction of the cost.

Case Study: Legacy Systems Migration

Fresh Roast Systems

Justin Kuo
Founder, Sparkle Technologies

Table of Contents

Case Study: Legacy Systems Migration – Fresh Roast Systems

This case study outlines an example of legacy systems migration, performed by Justin Kuo, founder of Sparkle Technologies.

Background

Fresh Roast Systems (FRS) is a company in Palo Alto, California, which designs and manufactures high tech industrial coffee roasters. Their customers, owners of coffee shops, lease these machines to roast coffee in their cafes. Many years ago, FRS went bankrupt and the assets were taken over by the main investor. The investor received the roasters, some spare parts, and a thousand-page printout of the software code that ran the roaster, but all the employees associated with the bankrupt company were gone, taking with them their proprietary knowledge. Bankruptcy proceedings had also taken several years and by the time FRS was ready to resume operations, the roaster software and electronics were over 15 years old.

Embedded on the coffee roaster is a touch screen computer. This computer hosts the main software used by customers to control the roaster. Since the software was old and outdated, FRS wanted to rewrite the software and contacted General Electric, but they quoted a seven-figure price for their in-house CIMPLICITY programmers to perform the job (the original software was written and owned by GE with their CIMPLICITY language). Thus, Justin Kuo was brought in to do the job instead.

Figure 1: Picture of the coffee roaster

Project Scope

The roaster is a fairly complex machine. It costs $40,000 each to manufacture, including $10,000 of electronics. It is controlled by the roasting software which has three main functions:

1) Provides a user interface for the customer to use
2) Reads sensor inputs (temperature readings, digital inputs, and analog inputs) and controls sensor outputs (digital and analog outputs)
3) Coordinates the sensor inputs and outputs to perform tasks (roast coffee, mix coffee, move the carousel, etc.)

Hardware controlled by the software include high powered blowers to move air, a catalytic converter to catalyze the exhaust, heaters to heat the air, a laser reflectometer to read the color of coffee beans during the roasting process, a carousel to store the beans, scales to weigh the beans, motors to turn the roasting drum, open and close various doors, and much more.

Figure 2: The inside of the coffee roaster

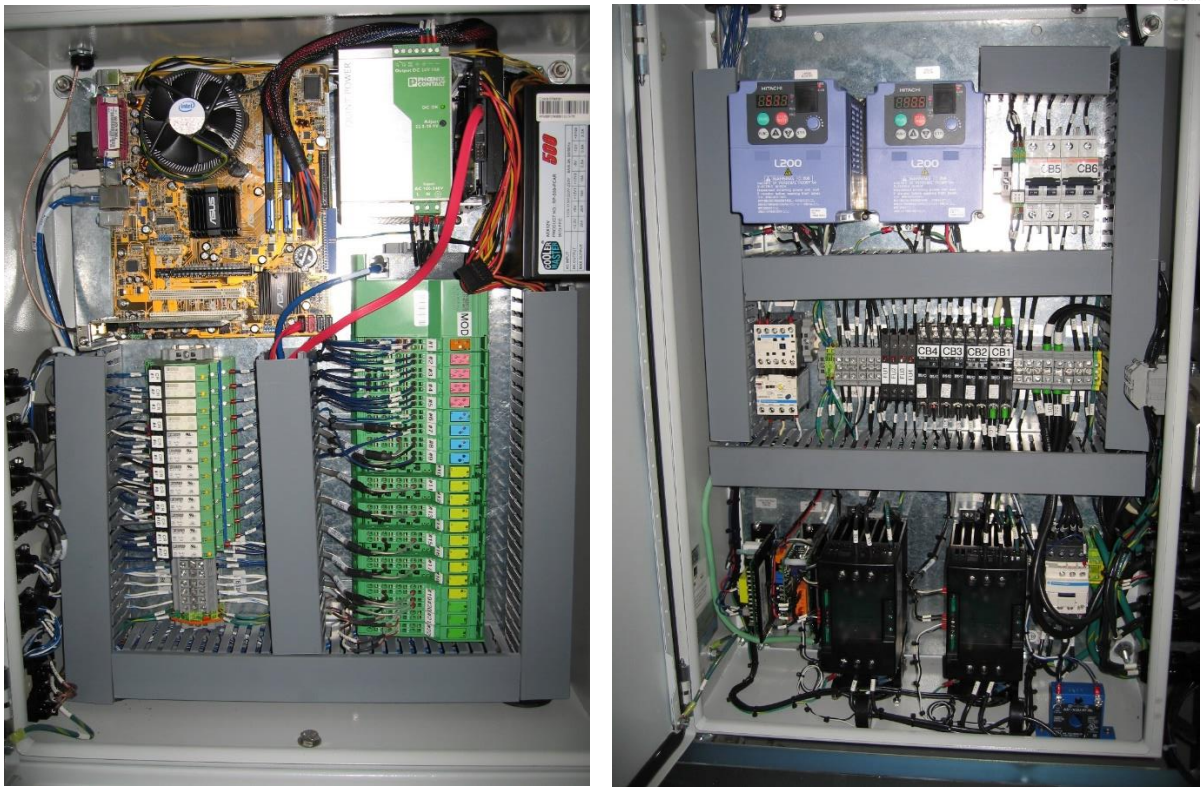


Figure 3: The electronics panel of the coffee roaster, controlled by the roaster software

This project essentially consisted of three phases:

1) Understanding the existing system
2) Understanding the desired features of the new system
3) Implementing the new system

Note that this project was done in an agile way so while it is broken down in three separate phases here, in reality, there was a large amount of cycling back and forth.

Understanding the Existing System

The original software had very few features, and the system / software architecture reflected it.

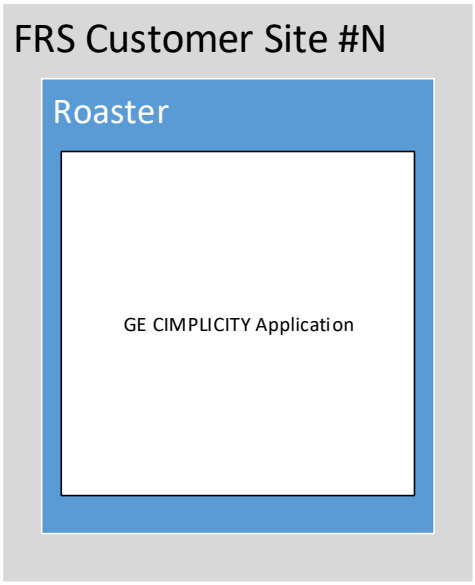


Figure 5: Existing System Architecture

As shown in the diagram above, the roaster was located at the customer site and there was a GE CIMPLICITY Application that ran the roaster. The application ran on a consumer version of Windows, was extremely basic, and crashed often.

While the original software was simple in terms of features for the user, it was challenging to understand how it controlled and coordinated all the hardware. The original code was implemented and owned by GE. All FRS had was a thousand-page printout of the code. Additionally, the "code" was done in ladder logic, which made interpreting how it worked difficult.

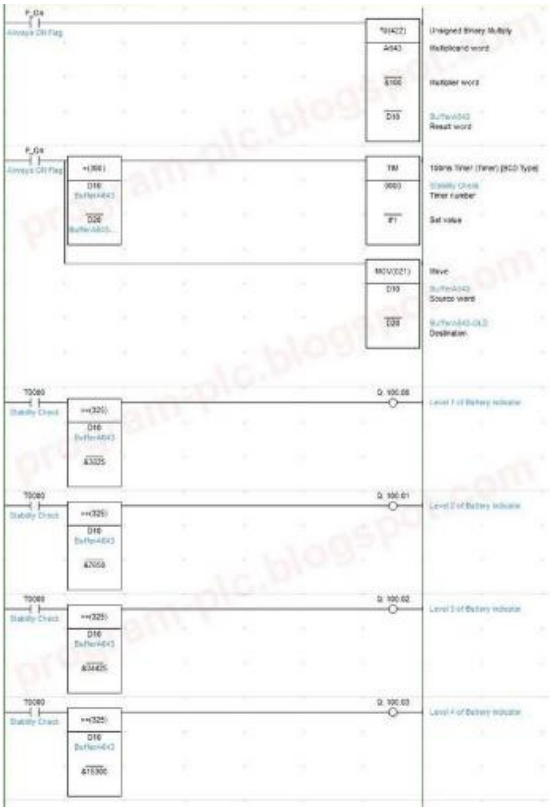


Figure 4: One page of ladder logic printout

Using a roaster with the original software, a voltage meter to monitor hardware inputs and outputs, and the ladder logic code printout, Justin was able to reverse engineer and understand all the parts of the roaster that could be controlled, all the sensor readings available to the software, and ultimately, how the original software controlled the roasting process.

For example, a simple command such as moving the carousel to the next bin consisted of the following tasks:

1) Apply a “high” voltage on the carousel motor analog output (to move the carousel).
2) Poll the digital input bin sensor until it is reads “1,” meaning a metal pole which represents the start of a bin is read by the sensor (to know the bin is almost in position).
3) Reduce the applied voltage on the carousel motor analog output (to move the carousel slower).
4) Poll the digital input bin sensor until it reads “0,” denoting it has passed the metal pole (to know the bin is in position).
5) Remove the applied voltage on the carousel motor analog output (to stop the carousel).

Understanding the Desired Features of the New System

FRS had a list of features they wanted for the new system such as:

- Allow the roaster to make blends (mix different raw beans together).
- Allow the user to draw a roasting profile (specify what temperature to roast the beans at different points in time).
- A modern user interface for the software.

However, the employees were non-technical, had only joined FRS recently, and did not really understand what was possible for the new system. This is one area where a good software consultant really separates him/herself from a poor software consultant.

Because Justin now had a deep understanding of the system, he was able to discuss with FRS employees and FRS customers their current pain points and burdens, brainstorm with them, and suggest improvements that FRS employees did not know were possible.

For example, FRS employees were spending large amounts of time on the phone with their customers to troubleshoot and fix roaster breakdowns. Justin realized that the roaster hardware could be modified to connect the computer to the Internet. This would allow for:

- Automatic upgrading of the roasting software without visiting the customer.
- Remote monitoring and diagnostics of the roaster.
- Detection of when roasters were about to fail.
- Remote control of the roaster.
- Automatic billing (FRS leased out roasters and billed based on usage).

By understanding the needs of FRS and their customers, brainstorming with FRS employees, the new software could now be made orders of magnitude better than the old software.

Implementing the New System

The original system only consisted of an embedded software that ran on the roaster. The new system included many more pieces to reflect the increase in capabilities. Implementation started on the roaster software itself and ended with the more ancillary pieces. While this breaks up the process into three distinct stages (Understanding the Existing System, Understanding the Desired Features of the New System, and Implementing the New System), the actual process was much more agile and iterative with much bouncing between the stages.

High Level System Diagram

Below is a high level system diagram.

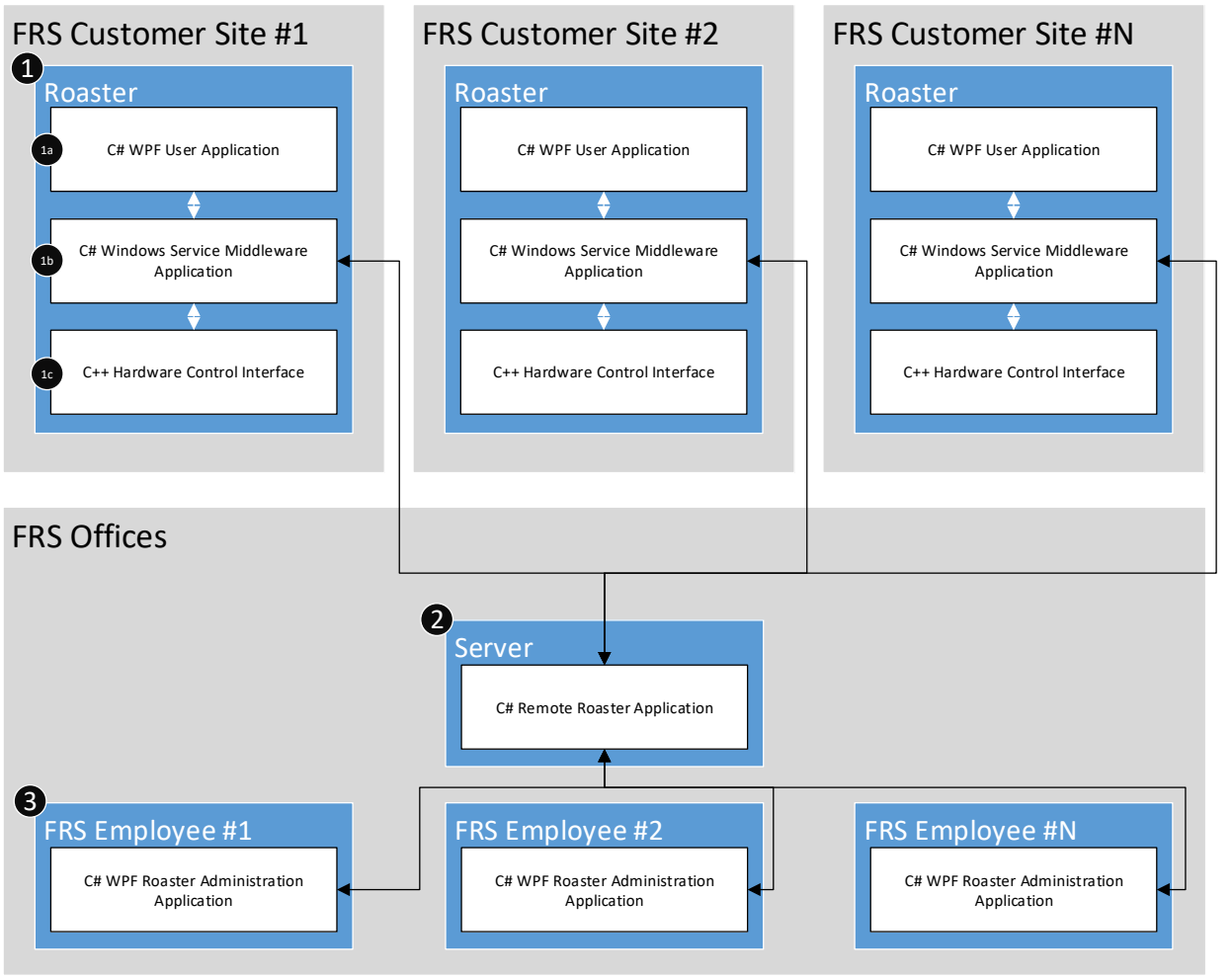


Figure 6: New System Architecture

There were several components to the new system, labelled in the diagram:

1) This is the coffee roaster itself. On the roaster, there is a computer than runs the Windows Embedded Operating System. Then, there are three custom software applications that work together to run the roaster:
    a. The C# WPF User Application is the application the roaster customer uses to control the roaster. This is all the roaster customer knows and the application is started upon powering up the roaster.
    b. The C# Windows Service Middleware Application is the application that controls the roaster. It takes commands from the WPF User Application (such as start roasting coffee), stores all the data for the machine (such as amount roasted, roast profiles, etc.), and sends back information to the WPF User Application to be displayed on the screen. This application also communicates with the C# Remote Roaster Application in the FRS Offices to allow remote diagnostics and control of the roaster.

c. The C++ Hardware Control Interface is the application that communicates with the low level hardware on the roaster. This reads the thermocouples readings, bin sensors, etc., and sets voltages on the analog and digital outputs.

2) This is a server running in the FRS Offices which hosts a C# Remote Roaster Application. This application communicates with the roasters in the field. It gathers all roaster data in real-time, monitors the roasters to ensure they are operating properly, and sends remote control commands to the roasters.
3) This is a C# WPF Application used by FRS Employees to monitor and diagnose issues with roasters, generate customer invoices, and remotely control the roaster. It communicates with the roasters through the C# Remote Roaster Application on the FRS Server.

Software Design

The software was designed to be reliable and maintainable. This meant using best software architecture practices, separating the different logical pieces into their own components, writing the software to be unit testable, and writing unit tests. Here, we will show the main components of the C# WPF User Application and the C# Windows Service Middleware Application.

C# WPF User Application

This is the application the customer uses to roast coffee and interact with the roaster. It is a C# MVVM Windows application and has the three typical MVVM layers (the Model, View, and View Model).

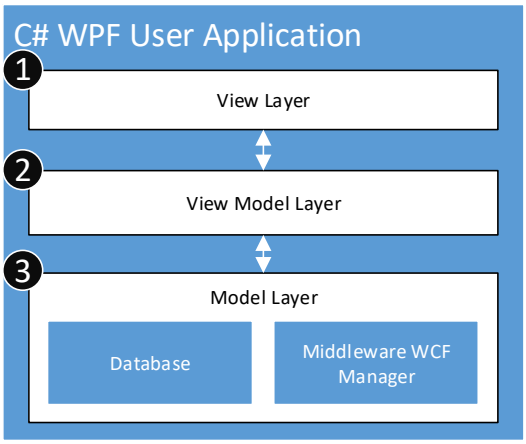
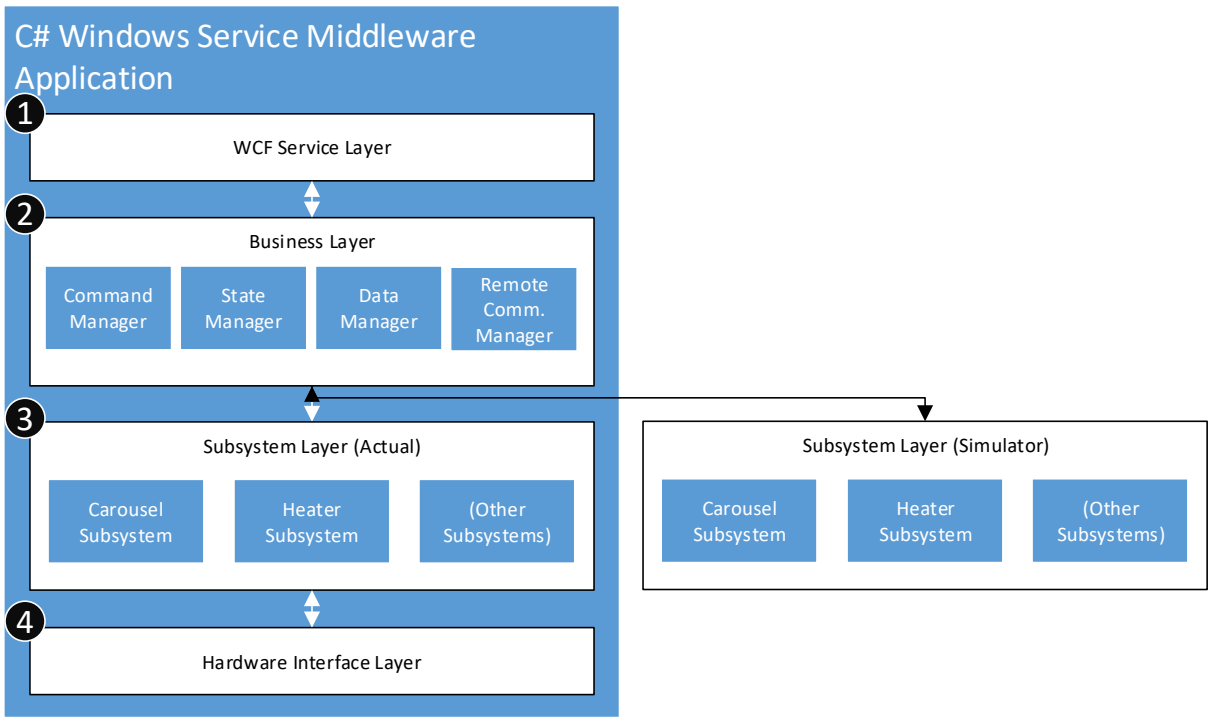


Figure 7: Layers of the User Application

1) The view layer consists of all the user interface code. The application runs on a Windows Embedded machine and the machine boots straight into the User Application. This application has a completely custom user interface (all buttons and icons were custom) and most customers are not aware it runs on a Windows machine.
2) The View Model code is the “glue” between the user interface code and the lower level model layer code.
3) The model layer consists of a database to store user preferences and a WCF Manager to communicate with the C# Windows Service Middleware Application. This is what the WPF User Application uses to send commands from the user to the coffee roaster and receive status updates from the coffee roaster.

C# Windows Service Middleware Application

The C# Windows Service Middleware Application is the heart of the coffee roaster software. This is the software that takes commands (such as "roast coffee", "spin the carousel", etc.) from the WPF User Application or from the Remote Roaster Application at the FRS Offices and actually performs the tasks, while providing status updates and real-time information back to the interested parties. While the software had too many components to document here, below is a diagram of the most important pieces.



There are four main layers in this software:

1) The WCF Service Layer is what takes commands from the User Application and the Remote Roaster Application over the Internet. It takes any commands received and sends them to the Command Manager in the Business Layer. This layer also provides real-time roaster updates to any subscribers.
2) The Business Layer is where the high level logic functions are implemented. We will not go into detail of the different components, but one of the main functions is to translate high level commands into more granular subsystem commands and coordinate the execution of the subsystem commands. For example, a command could be "weigh 5 pounds of coffee from bin #3. This is translated into
    a. Send the tare scale command to the scale subsystem
    b. Send the move carousel command to the carousel subsystem
    c. Send the open bin command to the carousel subsystem
    d. Poll the scale subsystem until the scale reads 5 pounds
    e. Send the close bin command to the carousel subsystem

3) The Subsystem Layer is where more granular commands are translated into actual analog and digital output commands. For example, opening or closing of a drum door could involve several analog and digital output commands. Note that there are two implementations of the subsystem layer, one for use with the actual hardware and one for a simulator so development of the roaster software could be done without the actual coffee roaster.
4) The Hardware Interface Layer is what actually communicates with the hardware. This is where the analog and digital output settings are sent to the Phoenix Contact hardware modules and the inputs are read in.

Final Results

The new software system was implemented and ran on cutting edge technology, had an attractive user interface for the customer, was much more stable, and had many more features than before. It could mix coffee from different carousel bins, roast coffee according to customers own roast profiles, be controlled remotely, provide real-time status to FRS employees in their offices, and much more.

While running the old software, employees spent a total of approximately 60 hours per week on the phone and at customer sites troubleshooting roaster breakdowns, the new software cut the time down to approximately 4 hours per week. As a result, FRS employees were able to spend time on better things and customers were much happier due to the reduction in roaster downtime. This was largely a result of:

1) Remote diagnostics – allowed FRS to see what was wrong with the roaster without talking to the customer
2) Preventative maintenance – new software detected when things were about to go wrong with the roaster
3) Automatic updating of the software – if issues were found, software could be pushed to all roasters before they experienced the same issue

Finally, the new software was owned by FRS. Since the original software was owned by GE, any changes had to be made by them which was extremely costly. Now FRS could make changes at a fraction of the price. Additionally, since they could make the modifications themselves, updates to the software could also be made at a much faster pace.

Conclusion

At Fresh Roast Systems, Justin took an old outdated software, reverse engineered it, worked with FRS employees and customers to figure out requirements for the new software, and implemented a new cutting edge software system that was orders of magnitude better than the original in almost every way imaginable. A good software consultant should not only help you by implementing new software, but by way of understanding your requirements, should be able to suggest many improvements, as was the case at FRS. Legacy systems migration can be difficult, however, if done right, can be a huge benefit to the business.