An Equipment Sales Cost Calculation System

A Manuscript
Submitted to
the Department of Computer Science
and the Faculty of the
University of Wisconsin-La Crosse
La Crosse, Wisconsin

by
Craig K. Lenz
in Partial Fulfillment of the
Requirements for the Degree of
Master of Software Engineering
May, 2012
An Equipment Sales Cost Calculation System

By Craig K. Lenz

We recommend acceptance of this manuscript in partial fulfillment of this candidate’s requirements for the degree of Master of Software Engineering in Computer Science. The candidate has completed the oral examination requirement of the capstone project for the degree.

Dr. Martin Allen
Examination Committee Chairperson

Dr. Thomas Gendreau
Examination Committee Member

Dr. Kasi Periyasamy
Examination Committee Member
Abstract


The availability and quality of product costs prior to manufacturing within the Trane Commercial Systems business of Ingersoll Rand has always been inadequate and poor. Of course, not having accurate costs means that the gross margin cannot be accurately calculated. Gross margin is needed as a component of discounting decisions and also as a leading indicator of future profitability. The focus of this project was to create a system to solve that problem.

This manuscript describes the development of a complete cost calculation system designed to deliver accurate costs and margin prior to, during, and after the sale of Trane products. It also describes the software engineering principles, learned in the Master of Software Engineering courses at UW-L, that were used throughout the development cycle. Those principles were used to successfully achieve the project’s goals.
Acknowledgements

I want to express my sincere thanks to several people who have helped me with this project over the last year. First I want to thank Dr. Martin Allen for his time and guidance on this project. Dr. Allen met with me weekly during his busy schedule and really helped keep me on track in order to meet my goal of completing the project in one year. Dr. Kasi Periyasamy has been such a wise and helpful professor. I believe he taught 6 out of the 8 classes that I took for the graduate degree, and they were all very interesting, challenging, and useful in my work at Ingersoll Rand (IR). UW-L is very fortunate to have him leading the graduate program. I want to thank my manager Brad Goetz at IR for giving me time, guidance, and funding to see this project through to completion. Several colleagues at IR played a role in this large project too and I want to give my thanks to them also, Glenn Fernandes, Mike Jefferies, Sume Nagamanickam, Aarthi Vedhavalli Gandhi Nathan, and Sundararajan Angappan. Ingersoll Rand, in general, is very supportive, both financially and personally. Last I want to give thanks to my biggest supporters, Jackie, my wife, and my three kids, Abby, Micah, and Nathan. Jackie has always been very encouraging, understanding, and patient with all the time that I have devoted to this project and the Master of Software Engineering program. My young kids have also been real troopers with all the time that this has taken my attention away from them.
Table of Contents

Abstract ........................................................................................................................................ iii
Acknowledgements .................................................................................................................... iv
Table of Contents ......................................................................................................................... v
List of Figures ................................................................................................................................ vii
List of Tables ................................................................................................................................. viii
Glossary ......................................................................................................................................... ix
1. Background Information ........................................................................................................ 1
   1.1 State of Costing .................................................................................................................. 1
   1.2 Sales System Process ......................................................................................................... 5
2. Project Goal and Overall System Requirements ..................................................................... 8
3. Software Development Process ............................................................................................... 10
4. High level system architectural plan ....................................................................................... 17
5. Functional Requirements ......................................................................................................... 19
   5.1 Cost Cache File Builder .................................................................................................... 19
   5.2 Part Cost DLL .................................................................................................................. 20
   5.3 Cost Interface DLL .......................................................................................................... 21
   5.4 Unit Coster ....................................................................................................................... 22
   5.5 Cost Template Designer ................................................................................................. 23
   5.6 JCHQ and UPS ................................................................................................................. 24
   5.7 Cost Detail Review Tool (CDRT) ..................................................................................... 25
   5.8 Automated Cost Loading ................................................................................................. 26
   5.9 Miscellaneous Maintenance Screens ............................................................................... 27
6. Design and Programming ........................................................................................................ 28
   6.1 User Interface Screen Designs ......................................................................................... 28
   6.2 Architectural Design ......................................................................................................... 28
   6.3 Database Design .............................................................................................................. 29
   6.4 Programming Languages ............................................................................................... 32
   6.5 Data Setup & Analysis ..................................................................................................... 33
7. Validation and Testing .............................................................................................................. 35
   7.1 Code Reviews ................................................................................................................... 35
   7.2 Gray-box testing ............................................................................................................... 36
   7.3 Usability Testing .............................................................................................................. 37
   7.4 Regression Testing and Automation ............................................................................... 37
   7.5 Component Testing and System Testing .......................................................................... 38
8. Project Challenges .................................................................................................................... 40
9. Continuing Work ....................................................................................................................... 42
10. Conclusion ............................................................................................................................... 44
11. Bibliography ............................................................................................................................ 45
Appendix A: Cost Detail Review Tool Screens ........................................... 47
Appendix B: Cost Template Designer Screens........................................... 51
Appendix C: Job Center HQ – Price Rollup Screen..................................... 54
Appendix D: Ordering Nbr Assoc. Maintenance Screen............................... 55
Appendix E: Cost Rate Maintenance Screen............................................. 56
### List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>Sales System Diagram (prior to the completion of this project) ..................</td>
<td>7</td>
</tr>
<tr>
<td>Figure 2</td>
<td>General outline of software re-engineering process ..................................</td>
<td>11</td>
</tr>
<tr>
<td>Figure 3</td>
<td>The Waterfall Software Process ...................................................................</td>
<td>12</td>
</tr>
<tr>
<td>Figure 4</td>
<td>The Prototyping Model ...............................................................................</td>
<td>13</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Functional Requirement for MonitorSelectionQueue function ........................</td>
<td>14</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Architectural Diagram of the Costing System ..........................................</td>
<td>18</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Partial class diagram .............................................................................</td>
<td>29</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Partial ER Diagram of sales system DB ..................................................</td>
<td>31</td>
</tr>
<tr>
<td>Figure 9</td>
<td>Model View Control Framework ...................................................................</td>
<td>32</td>
</tr>
</tbody>
</table>
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>Example price table for motor horsepower option</td>
<td>2</td>
</tr>
<tr>
<td>Table 2</td>
<td>Example Price Table for Sensor Accessory (Ordering Number Based)</td>
<td>2</td>
</tr>
<tr>
<td>Table 3</td>
<td>Example part rules for a motor part type</td>
<td>3</td>
</tr>
<tr>
<td>Table 4</td>
<td>Test Case of Various Costing Method Scenarios</td>
<td>36</td>
</tr>
<tr>
<td>Table 5</td>
<td>Test Case Scenarios Relating to Test Case in Table 4</td>
<td>37</td>
</tr>
</tbody>
</table>
Glossary

Backlog
A collection of all products not yet manufactured, but customer commitment to purchase those products has been received.

BOM
Abbreviation for bill of materials. It is a list of the raw materials, sub-assemblies, intermediate assemblies, sub-components, components, parts and the quantities of each needed to manufacture an end product.

Cincom
One of the manufacturing systems used in Trane. Made by Cincom Systems, Inc.

Cognizant
Consulting company contracted by IR to fulfill IT consulting service needs.

COJO or Coordinated Jobs
Team of people who perform job coordination (or discounting) functions with IR.

Configuration or Configured Product
Same definition as selection. It is a variation of a product after the sales associate has picked all the options desired. A selection belongs to a Job.

Cost Template
The cost template is the attributes, set per product family, used in calculation of costs of a selection from those product families. An application called the Cost Template Designer will be used to create and edit these templates.
Design Special
A design special is a special option that is not offered via a standard configuration. A sales associate can request these from the manufacturing location. The manufacturing location determines design costs and charges and provides the special pricing authorization (SPA) back to the sales associate.

DLL
Abbreviation for dynamic link library. It refers to Microsoft’s implementation of the concept of a shared library, which is a library that can be shared by multiple programs throughout the operating system.

ESTRNP
The production enterprise database. The FOE application transmits order data to this database when an order is committed. Several other applications, like KODIAK & MDP also run against this database in the order fulfillment process.

FOE
Abbreviation for Field Order Entry. This is an application used by Trane sales associates that allows them to create and edit orders.

Gross Margin (or Margin)
The difference between revenue and cost before accounting for certain other costs. Generally, it is calculated as the selling price of an item, less the cost of goods sold.

HVAC
Abbreviation for Heating Ventilating and Air Conditioning. HVAC refers to the technology of indoor and automotive environmental comfort.
IR
Abbreviation for Ingersoll Rand. IR is the parent company of Trane. IR is the sponsor of this project.

Job
A job is a collection of configured products for a specific customer’s needs. A job also contains many attributes such as location of building, customer name, etc.

Job Center
An application used by Trane sales associates that allows them to manage equipment jobs, including configuring and pricing products.

Job Center HQ
A special version of the Job Center application used by the coordinated jobs department for viewing jobs and managing the discounts given on jobs in the quoting phase of the project. JCHQ is another abbreviation of this application’s name.

MDP
Abbreviation for manufacturing data preparation. This is an application that serves as a linkage between the front-end sales systems and back-end mfg systems performing some order fulfillment functions. Part pick and validation rules are set up in the system for use in validating selections and determining the BOM.

MDP background process
Applications that are outside of the MDP application, but triggered by changes made within MDP. These background processes perform operations like selection validation and part picking.
Oracle BI or OBI
Abbreviation for Oracle Business Intelligence. OBI is a complete, open, and architecturally unified business intelligence solution for the enterprise that delivers capabilities for reporting, ad hoc query and analysis.

Order
An entity which ties the configured products to be purchased together along with other information such as estimated ship date, ship address, etc.

Price Table
A product family can have several price tables, one for each category. A price table contains rules which relate to selected options. Each rule can have a list $, net $, and a cost associated with it.

Part Pick Rule
Within the MDP application, a part pick rule can be configured for a product. The part pick rule relates selected options with a particular part, which is one element of the BOM.

Re-engineering
The modification of a software system that takes place after it has been reverse engineered, generally to add new functionality, or to correct errors.

Selection
A variation of a product after the sales associate has picked all the options desired. A selection belongs to a Job.
**SOTRNP**
The production coordinated jobs database. The JCHQ application runs against this database in the coordinated jobs process.

**Testing**
A type of validation applied to source code.

**TOPSS™**
Abbreviation for Trane Official Product Selection System. It is an application used by Trane sales associates that allows them to select and predict performance of Trane products operating under various conditions.

**UPDS**
Abbreviation for unitary product distribution system. It is used in the unitary products division of Trane. It handles all the manufacturing system functions for that business. It is an internally developed application.

**UPS**
Abbreviation for unitary pricing system. This is an application used for coordination of the unitary light commercial products.

**Validation**
A process that confirms that the product (or partial product) meets the expectations.

**Verification**
A process that confirms a development process or activity or task to be correct.
1. Background Information

The Trane Commercial Systems business of Ingersoll Rand produces and sells a large range of products for the HVAC market. Most of these products are engineered to order. This means that the product requested is built specifically for the job based on the customer’s requirements. With this type of built to order business, it is not easy to know the cost of the equipment prior to manufacturing. For some models of products, there are literally trillions of ways that a product could be configured. Because of the huge number of combinations, it is not practical to build or maintain costs for all potential variations of a product. Prior to the completion of this project, costs were not easily known at the time that list prices were set. Additionally, costs were not known at the time that discounts were requested for a job or at the time that the equipment was ordered. It was not until the equipment was manufactured that the cost calculation was done within each location’s manufacturing system.

1.1 State of Costing

Two methods have been used for pre-sales cost calculation of configured selections. One method employs a “cost control unit” process, which involves re-entering the configuration of the product in a separate system from the sales application. The re-entered configuration is then costed, the process of calculating the cost, in an overnight background process called the MDP background process. This cost control unit process is error prone due to the re-entering of the configuration, and it is time consuming because someone must manually re-enter the configuration. It is also slow due to the need to wait until the next day to find the result.
The second method that has been used is to determine a cost for each option level price rule. Pricing is maintained at the option level with configuration rules.

<table>
<thead>
<tr>
<th>BASEUNIT/MTHP</th>
<th>BASEUNIT/UNVT</th>
<th>List Price</th>
<th>Net Price</th>
<th>Std Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not Set</td>
<td>Any</td>
<td>$0.000000</td>
<td>$0.000000</td>
<td></td>
</tr>
<tr>
<td>1/3 Horsepower</td>
<td>2-Speed 115/60/1</td>
<td>$382.758300</td>
<td>$87.163200</td>
<td></td>
</tr>
<tr>
<td>1/3 Horsepower</td>
<td>115/60/1 OR 208/60/1 OR 230/60/1</td>
<td>$261.579500</td>
<td>$60.466500</td>
<td></td>
</tr>
<tr>
<td>1/3 Horsepower</td>
<td>277/60/1</td>
<td>$284.745000</td>
<td>$65.796400</td>
<td></td>
</tr>
<tr>
<td>1/3 Horsepower</td>
<td>220/60/1 OR 240/50/1</td>
<td>$322.678300</td>
<td>$74.561700</td>
<td></td>
</tr>
<tr>
<td>1/3 Horsepower</td>
<td>190/50/3 OR 380/50/3 OR 415/50/3</td>
<td>$364.355900</td>
<td>$84.192200</td>
<td></td>
</tr>
<tr>
<td>1/2 Horsepower</td>
<td>2-Speed 115/60/1</td>
<td>$483.737200</td>
<td>$121.430500</td>
<td></td>
</tr>
<tr>
<td>1/2 Horsepower</td>
<td>115/60/1 OR 208/60/1 OR 230/60/1</td>
<td>$280.887300</td>
<td>$64.906500</td>
<td></td>
</tr>
<tr>
<td>1/2 Horsepower</td>
<td>277/60/1</td>
<td>$322.678300</td>
<td>$74.561700</td>
<td></td>
</tr>
<tr>
<td>1/2 Horsepower</td>
<td>208/60/3 OR 230/60/3 OR 460/60/3</td>
<td>$364.355900</td>
<td>$84.192200</td>
<td></td>
</tr>
<tr>
<td>1/2 Horsepower</td>
<td>220/50/1 OR 240/50/1</td>
<td>$435.759900</td>
<td>$100.650000</td>
<td></td>
</tr>
<tr>
<td>1/2 Horsepower</td>
<td>190/50/3 OR 380/50/3 OR 415/50/3</td>
<td>$395.112700</td>
<td>$91.299200</td>
<td></td>
</tr>
<tr>
<td>3/4 Horsepower</td>
<td>2-Speed 115/60/1</td>
<td>$702.858400</td>
<td>$162.410400</td>
<td></td>
</tr>
<tr>
<td>3/4 Horsepower</td>
<td>115/60/1 OR 208/60/1 OR 230/60/1</td>
<td>$397.786300</td>
<td>$91.917000</td>
<td></td>
</tr>
<tr>
<td>3/4 Horsepower</td>
<td>277/60/1</td>
<td>$391.011800</td>
<td>$90.351600</td>
<td></td>
</tr>
<tr>
<td>3/4 Horsepower</td>
<td>208/60/3 OR 230/60/3 OR 460/60/3</td>
<td>$395.112700</td>
<td>$91.299200</td>
<td></td>
</tr>
<tr>
<td>3/4 Horsepower</td>
<td>575/60/3</td>
<td>$401.798900</td>
<td>$92.844200</td>
<td></td>
</tr>
<tr>
<td>3/4 Horsepower</td>
<td>220/60/1 OR 240/50/1</td>
<td>$493.224400</td>
<td>$113.970000</td>
<td></td>
</tr>
<tr>
<td>3/4 Horsepower</td>
<td>190/50/3 OR 380/50/3 OR 415/50/3</td>
<td>$439.821500</td>
<td>$101.630100</td>
<td></td>
</tr>
<tr>
<td>1 Horsepower</td>
<td>2-Speed 115/60/1</td>
<td>$980.516000</td>
<td>$226.569100</td>
<td></td>
</tr>
<tr>
<td>1 Horsepower</td>
<td>115/60/1 OR 208/60/1 OR 230/60/1</td>
<td>$399.054300</td>
<td>$92.210000</td>
<td></td>
</tr>
<tr>
<td>1 Horsepower</td>
<td>277/60/1</td>
<td>$418.782000</td>
<td>$96.768500</td>
<td></td>
</tr>
<tr>
<td>1 Horsepower</td>
<td>208/60/3 OR 230/60/3 OR 460/60/3</td>
<td>$439.821500</td>
<td>$101.630100</td>
<td></td>
</tr>
<tr>
<td>1 Horsepower</td>
<td>575/60/3</td>
<td>$399.974100</td>
<td>$92.421900</td>
<td></td>
</tr>
<tr>
<td>1 Horsepower</td>
<td>220/60/1 OR 240/50/1</td>
<td>$667.510400</td>
<td>$154.224500</td>
<td></td>
</tr>
<tr>
<td>1 Horsepower</td>
<td>190/50/3 OR 380/50/3 OR 415/50/3</td>
<td>$476.595500</td>
<td>$110.127500</td>
<td></td>
</tr>
</tbody>
</table>

Table 1 - Example price table for motor horsepower option

<table>
<thead>
<tr>
<th>ACCY/SENS</th>
<th>Ordering Number</th>
<th>List Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without zone sensor</td>
<td></td>
<td>$0.000000</td>
</tr>
<tr>
<td>Room sensor w/override button</td>
<td>BAYSENS573A</td>
<td>$100.000000</td>
</tr>
<tr>
<td>Room sensor w/temp adj. and override</td>
<td>BAYSENS574A</td>
<td>$117.000000</td>
</tr>
<tr>
<td>Single set pt manual changeover sensor</td>
<td>BAYSENS606A</td>
<td>$96.000000</td>
</tr>
<tr>
<td>Dual setpoint changeover sensor</td>
<td>BAYSENS608A</td>
<td>$156.000000</td>
</tr>
<tr>
<td>Dual setpoint w/system lights sensor</td>
<td>BAYSENS610A</td>
<td>$328.000000</td>
</tr>
<tr>
<td>Programmable sensor w/night setback</td>
<td>BAYSENS619A</td>
<td>$511.000000</td>
</tr>
<tr>
<td>Digital display zone sensor</td>
<td>BAYSENS635A</td>
<td>$190.000000</td>
</tr>
<tr>
<td>Digital Display Wireless Zone Sensor</td>
<td>BAYSENS550A</td>
<td>$352.000000</td>
</tr>
</tbody>
</table>

Table 2 - Example Price Table for Sensor Accessory (Ordering Number Based)
Table 3 - Example part rules for a motor part type

Table 1 and Table 2 show examples of what a price table might look like. Each row of the price table is called a price rule. Table 1 is a price table on the motor horsepower (MTHP) category. This price table has 2 factors, motor horsepower (MTHP) and unit voltage (UNVT). Based on the options within those two categories, the price is set. What this means, for example, is that if the sales person specified this product with a $1/2$ horsepower motor running on 208 volts, 60
hertz, single phase power, then the list price is $280.89. Table 2 is an example of a price table that directly relates the price rules to a unique ordering number. This price table only has one factor, which is the type of sensor (SENS) picked. In this example, if the “Dual setpoint changeover sensor” was picked with this product, ordering number “BAYSENS608A” would be chosen with a list price of $156. In Table 3, an example of a set of part pick rules is shown. In this example, the part pick rules are all for a motor part relating to the same product for the price table in Table 1. In this case there is exists a relationship between the price table in Table 1 and the part pick rules in Table 3, since both have the same factors. This is normally not the case. This makes relating the parts and associated cost very difficult and labor intensive. Additionally, because the costs are difficult to capture, the cost data can get stale and therefore becomes invalid. To capture the cost at the option level, many cost control orders are created to try to determine the costs for the options. This is not an exact science and takes quite a bit of analysis to figure out how to extract the option cost out of the total cost. It should also be noted that the cost is determined based on parts, like screws, and the prices are based on options, like unit size. As a result, making the price and cost align is pretty difficult.

Other factors contributing to the difficulty in obtaining costs are related to differences in how the products are manufactured and set up for sales, and differences in the manufacturing systems. Although the products manufactured by Trane are for the same industry, HVAC, that does not mean the products are similar. Trane manufactures everything from small variable air volume (VAV) units all the way to large Centravac™ Chillers. The most complex products are probably the modular climate changer products. There are also electronic controls products and other aftermarket products, like rebuild kits. The wide variation in products, supported by the system, adds to the complexity in costing, because each different type of product has different costing needs. The manufacturing locations spread out throughout the country also have different manufacturing
systems and different means of handling cost and margin. For example, the unitary light commercial products out of Clarksville and Lynn Haven have an entirely different kind of manufacturing systems than the rest of the locations. Those locations also calculate their margin a bit differently than other locations. They include outbound freight cost and warranty reserve costs in their pre-sales margins, whereas other locations do not.

1.2 Sales System Process

Before the sales process starts, the product’s sales setup must occur. Product setup includes determining what options are to be offered and what the pricing for the options will be. On the manufacturing system end, BOM rules are created and costs are assigned for the parts. During the sales process, the product options are chosen within the Job Center or TOPSS application. Once all the options are picked, a list price can be determined for the configured unit, also referred to as a selection. Sales associates have the ability within the sales system to discount the list price to a certain level. Beyond that level, the sales associate must escalate the job for discounting, also referred to as coordination. The coordination team can authorize and assign discounts, or they can escalate for an upper management decision. Following the customer’s decision to accept the offer, the order enters a committed status and is transmitted by the FOE application back to the enterprise database. This also puts the order in a queue for manufacturing based on the entered ship date. When the manufacturing location is ready to build the unit, the order is run through the MDP application to complete the BOM, resulting in a serial number assignment, among other things. The Cincom or UPDS manufacturing system picks up the BOM information from MDP and carries the unit through its system. Following shipment, the financial system picks up the information, including the sales price and cost, and determines gross margin.
In terms of applications and databases, the diagram in Figure 1 shows the various systems involved in the sales process. The initial product and pricing data setup mentioned above is performed by the data maintenance applications running against the Oracle enterprise databases (ESTRN). A monthly replication of the product and pricing data from ESTRN to a Sybase master occurs. The Sybase master data is then pushed into each of the Sybase sales office databases. The data associated with the sales associate’s configuration is stored in the sales office Sybase database. When the sales associate requests a coordination to occur for discounting, the sales associate’s job data is transferred to SOTRN, the Oracle coordinated jobs database. A special headquarter version of Job Center, called Job Center HQ is used to complete the coordination. For jobs with only unitary light commercial products, another application, called UPS, is also used for coordination. Authorization of discounts is then passed back to the enterprise database so that if and when the order is committed, a validation of discounts can be done. Once the job is accepted by the customer, an order is created in FOE and the committed order data is then passed to the ESTRN database. From that point the order is processed by the Kodiak, MDP and Cincom applications or Kodiak and UPDS applications (for unitary light commercial products). For the Cincom portion, some data is fed into the Cincom database. The financial system runs on its own database, FSTRN, and pulls in pricing and cost from the ESTRN and Cincom databases.
Figure 1 - Sales System Diagram (prior to the completion of this project)
2. Project Goal and Overall System Requirements

The goal of this project is to provide a complete unit cost for all products at any point after the configuration of a unit is complete. This cost will be determined based on the configuration and will not require re-entry of the configuration, as has been the case for cost control unit process. It should also be calculated in real-time or near real-time, on demand. The cost calculation should be initiated at the time of discounting based on the configuration stored in the coordinated jobs database, when a sales associate submits the job for coordination. These costs will feed an application called Job Center HQ that allows headquarters coordination staff to see the complete job, the cost, list price, sales price and margin. The cost calculation should also be initiated later in the cycle at the time the order is committed based on the configuration stored in the enterprise database. Both costs for job coordination and costs for committed orders will feed a pricing analytics tool called PROs. Additionally, the costs will flow into a data warehouse for backlog reporting (i.e. reports on equipment not yet manufactured) via Oracle BI.

A secondary goal is to be able to preload ordering number costing into the enterprise database and associated price tables so that the costs are available at the time that the list prices are set for those items. This is an important goal because it establishes the relationship early on and thus avoids running into missing ordering number, and therefore missing cost, when a configured unit is in need of costing. Most of the time an ordering number will represent a complete accessory item which is sold with a configured product. Other times an ordering number can represent a complete “built to stock” type product.

These system requirements were developed based on the company’s goals as expressed by Rick Aldridge, who realized all these gaps in the current system. All of the requirements were to be satisfied within the context of the existing
system, as described in section 1. A means of setting up a costing template for each product was needed, as was functionality for taking a configured job and producing a cost. Modification to the Job Center HQ application was needed to pull in the new costs and calculate and display the margin. A means of displaying the cost details was needed for investigation of costs. Functionality for automated cost loading was also required. All of these business requirements have been factored into the system’s architectural design, as described in section 4, and functional requirements, described in section 5.
3. **Software Development Process**

When taking on any large project, it always seems to work best to break the overall project down into parts. With this project, the first step that the author took was to break the overall project into three categories. At the core of the project there needed to be the functionality to be able to cost a selection at any point in the sales process. So the costing functionality was one of the three main categories. Prior to costing a selection, the costing functionality needed to have some information to drive it. The maintenance system, providing attributes onto the products, was the second category of the project. The third category of the project was the functionality to display the costs and use those cost for margin calculations.

From these three categories a high level architectural plan was developed to describe the components needed within each of the three categories. Within the costing functionality category, the main components needed were: 1) the Unit Coster, which was to run 24x7 costing selections as needed, 2) the part costing DLL and cost interface DLL were two other key components needed to cost selections based on the type of costing needed, and 3) the cost cache file builder was needed to routinely build cache files for the part costing DLL to use.

Within the second category of the project was the cost template designer application which allowed users to set the cost attributes for each product. Additionally, cost rate maintenance and ordering number cost relationships maintenance were needed to provide rate data and ordering number relationships.

Within the third category of the project, the JCHQ and UPS application changes were required to view costs and calculate margin. The cost detail review tool was another component in this category, designed for display of the cost breakdown for a selection. The third component within the third category is the use of the cost data in pricing analytics, via PROs, and in reporting, via Oracle BI.
After the high-level architectural design was completed, each one of the components of the system was developed one at a time. Although the components were interrelated, a sequenced development order of components was determined, so that each could be developed in a somewhat independent manner. This helped break the large effort down into manageable sized chunks. It also allowed for small successes and useable pieces, on the road to developing the entire system [3].

It took a few different software engineering models to develop each of the components. The initial development was focused on the re-engineering of the existing part pick functionality, known as the MDP background process, which was written in Unix C to run on the database server. This effort to build the Part Cost DLL did truly follow a traditional re-engineering process. The goals of this effort were: 1) to get the costing to run on a Microsoft Windows server, 2) to run against the VPFC/VPC/SI data as opposed to FCAT/FCODE data, 3) to run discounted from the database for the part pick rule processing, and 4) to include an association from parts to their cost so that total unit cost could be calculated.

![Diagram](image)

Figure 2 - General outline of software re-engineering process (Based on [5], page 219, exhibit 15-4)
The diagram in Figure 2 shows a general outline of the software re-engineering process [5]. Re-engineering starts with an existing system. Understanding of the existing functionalities and structure and transforming them into new structure are the key parts. In this case, the MDP background process was the existing system that had to be understood. This code is massive and literally hundreds of functions were involved in the re-engineering process. There was no documentation for these functions outside of the comments embedded in the source code, therefore the source code served as the main basis for understanding the existing requirements. Another aid in understanding the functions was the data which has been set up for the part pick and validation rules for each product. There was documentation on the various rule formats and that did serve as a beneficial resource. The last source of knowledge on the existing system was the output data. Having the input and output data for the existing functionality also served as a great means to test the resulting Part Cost DLL and verify that its results were equal to the existing MDP background process.

![Figure 3 - The Waterfall Software Process](Based on [2], page 2, figure 0.3)
For many of the components of this system, the traditional waterfall model was used. The waterfall model works great for applications or functionality which do not have a user interface. In this system, the waterfall model, as shown in Figure 3 [2], was used for: 1) the Cost Cache File Builder, 2) the Cost Interface DLL, 3) the Unit Coster, and 4) the Automated Cost Loading functionality. Each of these components was developed independently in its own waterfall process and linked into the system as it was completed. As described in section 5, the functional requirements, developed by the author, were completed and reviewed with the sponsors. The design, as described in section 6, was also completed by the author. For most of the components, the programming was completed by an intern or Cognizant consultant. The code was reviewed by the author during the development and after completion prior to testing. Testing, as detailed in section 7, was also mainly completed by the author. In one case an automated test suite was used, managed by a separate Cognizant testing team. Following deployment, as with many new systems, some maintenance work was also required to work out a few unplanned issues.

Figure 4 - The Prototyping Model
(Based on [17], page 53, figure 2.5)
The prototyping model [17], shown in Figure 4, was used for several components of the system where a graphical user interface was required. Those components included: 1) the cost template designer, 2) JCHQ & UPS, 3) the Cost detail review tool, and 4) the cost rate maintenance and ordering number cost relationships maintenance screens. The prototyping model was ideal for these components because they required a fair amount of interaction with the users who had a hard time articulating the requirements. Once the users could see and interact with the prototype, then the specific requirements of how they would interact with the UI, and what data they needed to see, came to light.

Within the waterfall or prototyping models, functional requirements were written for the components of the system [6,8-13] by the author. The IEEE Standard 830-1998 for writing software requirement specifications [4] was followed for these documents. The requirements for each of these components will be described further in section 5, but listed in Figure 5 is an example of a requirement written for the MonitorSelectionQueue function of the Unit Coster.
This is a functional requirement written for the monitoring of the selection queue and the response taken if a selection exists.

The design phase of the processes mainly followed the UML design standards. The best quote to represent the place of UML in the development process comes from *The Unified Modeling Language User Guide*: “Things that are best expressed graphically are done so graphically in UML, whereas things that are best expressed textually are done in the programming language” [1]. This included some use case diagrams, class diagram, and ER diagrams in this project. The author took the functional requirements and developed those UML diagrams.

Because of the lack of one clear functional expert in this area, and uncertainty from business sponsors, some of the requirements were revisited and revised after the development or testing had started. There was nothing that required an overhaul, it was just minor enhancements and fine tuning of the requirements that occurred to exactly meet the needs that arose. For example it was assumed that an ordering number would only appear in one price table, and therefore only appear once within a selection. Although it doesn’t occur very often, some cases to the contrary were discovered after running tens of thousands of configurations through the costing functionality. A correction to the functionality had to be made to account for duplicate ordering numbers in a selection.

After the design was completed, either an intern or Cognizant consultant took the design and requirements and used them to do the programming needed. Throughout the coding and after the completion of the coding, the author completed code reviews to make sure that good programming practices were followed and that the design was interpreted correctly. Additionally the code reviews served as a basis for gray box testing.

Testing and deployment were the final pieces. These were also completed by the author. Testing involved white box, black box, and gray box testing. Unit testing of the functions occurred to ensure their accuracy. Gray box testing involved testing the components based on knowledge of how the code was put
together. Black box testing was performed on some components using an automated test suite in some cases and manual testing in others. Black box testing was performed by a separate Cognizant development team using the requirements as a basis for what needed to be tested. Additionally some components were put through a semi-automated test by the author to test the resulting costs against known cost results from the manufacturing systems.
4. High level system architectural plan

Given the fact that the existing Trane sales and manufacturing system needed to stay intact, the architecture of the costing system was expected to revolve around that existing system without disturbing it too much. Additionally, in order to accomplish the goals of this project, it seemed like breaking the needs down into components of the system would help break the daunting task into several smaller projects which could be accomplished. Creating a componentized system would also add to future reusability.

From the existing sales and manufacturing system, shown in Figure 2, the high level architectural plan was formed which would shape the direction of the project. The result of this planning stage was a diagram of how things would fit together inside of the existing Trane sales and manufacturing system, and a document describing the purpose of each of the components [7]. That resulting diagram is shown in Figure 6.

As described in section 3, there were three categories of needs that were taken into account: 1) maintenance components to maintain data supplied to the costing functionality, 2) the costing components, and 3) components to view or use the cost data. The design of the costing system takes into account these categories of components. Within the maintenance components, there is: 1) the cost template designer application, which allowed users to set the cost attributes for each product, and 2) the cost rate maintenance and ordering number cost relationships maintenance components, which were needed to provide the rate data and ordering number relationships. The costing components needed were: 1) the Unit Coster application, which was to run 24x7, costing selections as needed, 2) the part costing DLL for costing products set up for part costing, 3) the cost interface DLL, which was to cost products having all other costing methods, and 4) the cost cache file builder, which was needed to routinely build cache files for the part costing DLL. Within the category of components to view or use the cost data, the
following were needed: 1) changes to the JCHQ and UPS applications to view costs and calculate margin for the coordinated jobs process, 2) design of the cost detail review tool for display of the cost breakdown for a selection, and 3) a means of feeding the cost data to the PROS pricing analytics tool, and to the data warehouse for use in Oracle BI reports.

Figure 6 - Architectural Diagram of the Costing System
5. Functional Requirements

Functional requirements were identified for each component of the system. Software requirement specifications were written for each component based on the IEEE Standard 830-1998 for writing software requirement specifications [4]. A brief summary of the functional requirements for each component is explained in a subsection below.

5.1 Cost Cache File Builder

The functional requirements for the cache file builder were developed in tandem with the functional requirements for the Part Cost DLL. The cost cache file is a component needed to build cache files on a nightly basis to hold all of the part pick rules, product data, and part costs needed by the Part Cost DLL to cost a selection without going to the database. The requirement to use cache files was not a business decision but a design decision based on the fact that the cache files could be formatted and presented in a way that would make the Part Cost DLL more efficient. The Part Cost DLL will need to cost a selection as quickly as possible and using cache files allows us to speed up the process. A second reason for using a cache file was that the manufacturing system databases were known to have frequent outages; putting the part costs into cache files eliminated the possibility of the database outage interrupting the costing operation.

The functional requirements for this component [11] included retrieving data from the database in a specific arrangement and storing it in a file format. This component was required to be built with a high level of robustness to deal with database outages, mentioned above. If repeated database connectivity became an issue while the cache file builder was trying to retrieve its data, the fallback operation was to copy the previous day’s files for that manufacturing system.
database. E-mail notification of success or fallback are required to be sent following the completion of it process every night. The cost cache file builder component was not required to have UI. It is initiated nightly on a Windows Server from a task scheduler. It builds its files prior to midnight and puts them in a folder with the next day’s date. The Part Cost DLL always uses the folder for that date for processing the costing requests, so at midnight the Part Cost DLL switches to the new day’s cache files.

5.2 Part Cost DLL

The existing MDP part pick functionality was the basis for the Part Cost DLL. It had all the rule processing functionality needed, but it was written in Unix C procedural style, designed to run on the database server against the enterprise, designed to work on FCATs and FCODs as opposed to VPCs and SIs, and not designed to calculate costs, but only pick parts (i.e. build a BOM). The cornerstone to the success of this project was going to be whether this code could be reengineered into a C#.NET object oriented version. The MDP part pick code was very large so this was not an easy task. The first order of business was to analyze all the functions of the code and understand what each did. The main requirements were: 1) converting the C code to C#.NET, which involved modifying C string formatting to C# style, 2) converting the procedural code to an object oriented version, 3) removing and replacing the FCAT/FCOD functionality with VPFC/VPC/SI functionality, 4) using the cache files as opposed to the database calls to retrieve its data, and 5) adding functionality to look up costs for each part and tally the results. The Part Cost DLL was designed with no user interface, but it does need an application interface so that it can be called from other applications. In this system, the cost interface DLL is the component that invokes the part cost DLL. The cost interface DLL passes selection configuration data to the part cost DLL, which returns the cost and part list.
5.3 Cost Interface DLL

The purpose of the Cost Interface DLL is to handle all types of costing requests. It consisted of the following four methods: 1) Part Costing, 2) Price Table Costing, 3) Ordering Number Costing, and 4) Model Number Costing. The part costing method, which uses the Part Cost DLL described in section 5.2, is the preferred method for costing build-to-order products. This method is designed to cost products dynamically based on the part pick rules and the part costs. Another method, price table costing, has been around for some time. The cost values are entered into the same rule-oriented price tables as the list and net price values. The difficulty with this method is in keeping the costs up to date and accurate. This method is needed for products which are sold in the Trane sales system, but are manufactured by another company, although the method can easily be set up for any product. Ordering number costing is for use in build-to-stock products and accessory items. These items have a pre-assigned ordering number, which can be directly referenced in the manufacturing system to find the cost of that item. Model number costing is the last method. The model number can be built from the specified product configuration. That model number can then be directly referenced in the manufacturing system to find the cost of that model.

The Cost Interface DLL is passed a selection ID. The DLL retrieves the configuration of the selection and cost template for the product from the database and then processes that selection using one of the four methods. The resulting cost and cost details for the selection are then passed back to the calling application. The Cost Interface DLL was designed with no user interface, but it does need an application interface so that it can be called from other applications. In this system it is called by the Unit Coster application. For testing it has been set up to be called from the automated test apparatus.
5.4 Unit Coster

The Unit Coster is an application that runs 24x7, checking for selections to appear in its queue [32]. This application runs on a Windows server within the Windows task scheduler. The application has a modest user interface to display activity and allow for the application to be exited; however, because the application will run on a server and be invoked through a special application account, no one other than the developer will ever see the user interface. The most important requirement of this application is robustness. It needs to log errors and continue functioning no matter what happens. The application monitors a queue for selections. The queue is actually just a database table set up to hold records containing a selection ID, date/time created, number of processing tries and date/time of the last attempt. When one or more selections are present in the queue, the application begins costing the first selection in the queue by invoking the Cost Interface DLL. When the application begins costing a selection, it modifies the queue record’s last attempted date/time to indicate that it has recently been tried. It also increments the attempt counter. If the costing attempt succeeds, the application removes the record from the queue. It then adds a record to a costing feedback table to indicate date/time of processing and total process time. It also saves the cost and cost details information to a master and child table. If the costing request fails, the application moves onto the next record in the queue. It ignores all records with a process attempt date/time within the last hour by employing a where clause which filters out all records which have failed in the last hour. When a preset number of attempts have been tried the application logs the failure in the feedback table and then removes the record from the queue.
5.5 Cost Template Designer

The cost template designer is a key application in the system because it provides the ability to maintain the cost attributes for each product [9]. These attributes dictate how the Unit Coster will process a costing request for a configuration of that product. The simple attributes include: unit costing method, cost type (standard or current), and manufacturing system source database. There is also a means to override the main costing method for specific product categories. For example, a product may use the part costing method, but several accessory categories may use ordering number costing.

A second main requirement for the cost template designer is to provide a means to initiate an automated cost load request for a product. Included in this is the requirement to break down the request by the product’s category. Essentially, the request process consists of saving the request to a database table with the e-mail address of the person making that request, and storing a breakdown of the product’s categories into a related table. The request is then picked up by the automated cost loading background process and processed.

A third main requirement of this application is to provide a means of editing rates for warranty and outbound freight cost for a product. The application also provides a means to override the base product’s rate for any product codes. For example, the rate for the main product may not apply to certain accessories included with that product. Those accessory items will have a different product code and therefore the application must allow the main rate to be overridden.
5.6 JCHQ and UPS

The JCHQ (Job Center HQ) application has existed for many years. It is used by a team COJO team. It is also used by some other product support groups in the manufacturing locations. The application provides a means to view all the selections on a job and set discounting factors to provide discounts requested by sales associates. The requirements for the changes to this application involve adding new columns into the *price rollup* screen [10]. This screen allows the users to view several different price types and rates. The new columns required are: 1) Standard Manufacturing Cost, 2) Warranty Cost, 3) Freight Cost, 4) Entered Standard Margin %, 5) Authorized Standard Margin %, 6) Entered Freight and Warranty Adjusted Margin %, 7) Authorized Freight and Warranty Adjusted Margin %, 8) Entered Contribution Margin %, and 9) Authorized Contribution Margin %. The columns are needed to allow JCHQ users to see the costs and margins and use those values in discounting decisions. JCHQ uses the cost data that the Unit Coster has saved to the coordinated jobs database. Formulas for the margins are based on feedback from personnel in finance, to determine precisely what was needed.

Another requirement is the ability to launch the Cost Detail Review Tool from JCHQ’s price rollup screen. This has been implemented via an onscreen link that calls a URL to bring up the browser. Part of the URL includes the Job ID of the job currently displayed on the price rollup screen. This enables the Cost Detail Review Tool to go directly to a display of the selections on that job, allowing the user to quickly get to the details of the cost for a selection.

UPS (Unitary Pricing System) is also an existing application, used to display price and cost on configured selections. UPS is only used for coordinating jobs consisting solely of unitary light commercial products. The requirements for this change only involves build-to-order products. Most of the products handled by UPS are build-to-stock products. UPS’s cost for those items is accurate, but for
the configurable products it is not. The required change involves bringing in cost for those units as calculated by the Unit Coster. No other changes to the existing system are required.

5.7 Cost Detail Review Tool (CDRT)

The need for the CDRT (Cost Detail Review Tool) came out of the need to provide product support personnel and cost accountants a means to view more than just the total cost. CDRT provides a means to drill down into the costs of a selection to see every detail of the cost. It is required to be a web tool, in order to provide an easy way to access the data from within the IR network [8].

The application provides several means of searching for a selection, and thus a selection’s cost. It supports searching by coordinated job, by committed order, by product code, and by product family. Based on the search filter, the search results are displayed in a grid. The grid provides several columns of information, and also a link to drill down into the costs for that selection. The selection details will be displayed differently based on the costing method. For the part costing method, it will display all the parts and their costs, along with a separate table of list pricing per option. For ordering number costing, the display is similar to the part costing display except that instead of displaying parts, it will display the ordering numbers and their costs. For the price table costing method, the display of the cost is combined with the list prices, since this information all comes from the same source and is directly related.
5.8 Automated Cost Loading

The automated cost loading application is designed to run on a server, so it is not intended to be seen by the users. According to the functional requirements document [6], automated cost loading requests will be made through the cost template designer and stored in two tables. The automated cost loading application will routinely scan for new requests in those tables. When it finds one, it will start the cost loading process. This process involves linking the ordering numbers from the price tables into the ordering number cost tables, as described in the next paragraph. The costs are then loaded back into the price tables so that the user will have the cost shown side-by-side with the list prices that they have set. This provides a means to verify accuracy of the list prices based on reductions or increases in the costs of those items.

A stored procedure is needed to collect the standard cost values for all ordering numbers used in the Trane products on a routine weekly basis. This stored procedure will cycle through the current ordering numbers; adding costs for any new ones and updating costs for any modified ones. The difficulty with this is matching the ordering number from the sales system to the manufacturing system. Some differences involve whether the ordering number is with or without dashes, with or without spaces, and some even have characters appended at the end of their string. The big advantage of running this procedure prior to the ordering number ever being selected in a configured unit that needs to be costed is that mismatches can be caught right away. Notification of mismatches will be sent to a designated list of users, including the author. Those items can then be manually corrected within the ordering number cost relationships maintenance screen described in section 5.9. Once the relationship is established, the costing will continue without intervention in the future. As an example, in the enterprise database, the ordering number is called “0233-0540-02-00” whereas in the Macon Cincom database the unit is called “0233054002000”. The stored procedure is
designed with some ability to handle differences, like stripping out the dashes, but sometimes the differences cannot be handled.

5.9 Miscellaneous Maintenance Screens

Two final requirements are the addition of a pair of maintenance screens to the Data Maintenance Center (DMC) application, in order to provide a way to maintain some product-related data associated with the cost system. The first screen is for overall maintenance of the rate for the cost add-on, like warranty and outbound freight cost. Section 5.5 outlined the ability to maintain the rates for warranty and outbound freight within one product. This new maintenance screen allows the user to edit data in the same database table, but also provides the ability to view or edit the rates for several products at the same time. It is designed so that the user can quickly select several products and do an update on all their rates at the same time.

The second screen is for maintenance of the ordering number cost relationships used in the stored procedure described in section 5.8. There are only two attributes involved in the relationship between the sales and manufacturing system that need to be edited when the relationship cannot be determined. The source manufacturing system database is the first attribute. This can be any one of the ten Cincom databases, or the UPDS database. The other attribute is the ordering number used in the manufacturing system database to identify the item. The number will resemble the one used in the enterprise database, but may use other or no separators between characters. Once those two attributes are modified, the next time the stored procedure, described in section 5.8, is run again, it will be able to starting getting costs for that item from the source database.
6. Design and Programming

From the functional requirements a design was established for each of the components. The design was on an object-oriented approach. UML diagrams such as use case diagrams and class diagrams were used. In addition to UML diagrams, ER diagrams were created for the database design. From there, the programming of the applications began using several programming languages and tools. In this section the author will discuss the design and programming techniques used for this project.

6.1 User Interface Screen Designs

When working on any of the components that required a user interface (UI), the design work usually starts with the UI. This project also followed the same technique. This included how the application was going to flow from one screen to another, what information be shown, and what actions would be occurring on each screen. The UI often dictated how the functional requirements would be developed, and so this process often occurred in conjunction with development of the functional requirements. Taking these designs and developing the working prototype of the UI was the next step. This allowed users to see the flow and look of the application without investing a large amount of time. Revision could easily be made at that point and new prototypes could be reviewed, all prior to completing the functional requirements.

6.2 Architectural Design

Based on groupings of functional requirements, classes were created for each
group. Based on the classes, all functional requirements were aligned with a corresponding class and turned into methods and attributes. Figure 7 shows part of the class diagram for the Part Cost DLL. In the upper right you can see the main class, CostCalculator, and the main interface functions to the Part Cost DLL, such as those to “CreateAndValidateSelection()”, “CreatePartListTable()”, and “ValidateSelection()” also appear. The associations to the other classes, like the “Validation” class are also shown in figure 7.

![Figure 7 - Partial class diagram](image)

(complete diagram appendix F)

6.3 Database Design

The existing database design used in the sales system dictated much of the design for this costing project. Several new and modified database schema were required to implement all of the functionalities required. New tables, views,
triggers, roles, DB links, functions, and procedures were added to the ESTRNP and SOTRNP databases. Some of the tables were added with foreign keys to the sales system tables to maintain referential integrity. The following is a list of just the tables that were added along with their purpose:

1. The PROD_FAMILY_COST_SETTING table was added to hold the main attributes for each product family that are needed in the costing operation.

2. The UNIT_COST_PROCEDURE table was added to hold overridden methods within a category of a product family.

3. The BU_COMP_COST table was added to hold the cost values (freight, warranty, direct material and direct labor, variable overhead, fixed overhead) for each ordering number.

4. The COST_ADD_RATE table was added to hold the freight and warranty rates at the product family or product code level. These rates are used to determine the cost by multiplying the rate against the manufacturing cost.

5. The PRICE_TABLE_COSTING_REQUESTS table was added to hold requests for automated cost loading for product families.

6. The PRICE_TABLE_REQUESTS table contains child records of the record in PRICE_TABLE_COSTING_REQUESTS table and is designed to hold the categories chosen in a costing request.

7. The COST_CALCULATION_FEEDBACK table was added to hold additional information collected during the unit costing operation. Errors, warnings, and process time is some of the information stored in this table.

8. The SELECTION_QUEUE_FOR_COSTING table was added to hold requests for the unit costing of selections and the date/time added. The Unit Coster processes the selections in this queue table based on order.

9. The SEL_PC_COST table was added to hold the total cost values for a selection.
10. The SEL_PC_COST_DETAIL table was added to hold the cost values for the parts of the total selection’s cost. It is a child of SEL_PC_COST.

Figure 8 - Partial ER Diagram of sales system DB
Figure 8 shows a portion of the sales system ER diagram with the new tables highlighted in fuchsia.

The reason behind storing the part rules and costs in cache files is due to the fact that the manufacturing system databases have many outages. Running off the cache files provides a way to work around the database outages and also provides a performance improvement.

6.4 Programming Languages

Several programming languages were used for this costing system. C#.NET was used for the Unit Coster, Cost Interface DLL, Part Cost DLL, Cost Template Designer and the Cost Cache File Builder. The main reason for using C#.NET for these components was the author’s familiarity with that language/compiler. That language is the primary one used in the author’s department for Windows client applications. Another reason was the ability to build the components in a modular fashions for reusability.

![Diagram](image)

Figure 9 - Model View Control Framework
(Based on [15], page 345, figure 13.1)

For the Cost Detail Review Tool, ASP.NET was used. The application was built with the MVC (model view control) framework, as shown in Figure 8, and JQuery. The model view controller framework separates the model, view, and
Controller into 3 separate modules [15]. The model is responsible for managing the data. The view is responsible for the display of the data provided. Lastly, the controller is responsible for converting user input into calls to invoke the model and view. The reason for using ASP.NET with JQuery is that this language is what others in the group were experimenting with for other new application development. This approach provides a tiered, light weight infrastructure for deploying the functionality over the web. The need for an easily accessible, zero footprint framework for the delivery of the information was needed to reach all of the people using the system, without requiring setup or training support.

For part of the automated cost loading, stored procedures were developed to automate the loading of the cost data into the Enterprise database. Stored procedures were written in PLSQL. One stored procedure was developed to update the list of ordering numbers. The other stored procedure was developed to compare and update the costs of the ordering numbers when different.

6.5 Data Setup & Analysis

Analysis of the setup of the products, their pricing, and their cost data was a large part of this project. Figuring out how to cost the product required an understanding of the connections between the data from the product and pricing setup all the way through to the cost calculation. No one person was able to supply all the information necessary to build a complete set of requirements for the project. Instead, that information had to be pulled together from pieces held by several different groups. The databases used also served as a good source to determine how things worked. The pricing data setup for the products served as one of the best sources of knowledge. Some product’s pricing was set up with the ordering numbers in the pricing rules. For some products every single price rule had an ordering number. These products were set up with the ordering price
method. For the products which had MDP part pick rules setup, the part costing method was used. Furthermore, for some parts of the pricing, ordering numbers were used. In the discovery phase, this analysis pointed out a need to have a mixture of costing methods. For products manufactured by external vendors, fortunately, those costs were maintained well within the pricing rules. These products required the standard price table costing method. Later, during the development of the Unit Coster application, the need for the model number costing method arose. This became apparent after talking with others and reviewing how the data was set up for those products.

In testing the cost calculated by the Unit Coster for a few products costed with the part costing method, it became apparent that costs did not match the unit costs in Cincom. Upon further discovery, it was found that these products had other costs added on at the time of manufacturing. For these products, it was decided to continue using the standard price table costing method for now and in a future phase of this project determine a way to cost these products dynamically.
7. Validation and Testing

Validation of the requirements with the business partners was completed to ensure that the system was developed in accordance with the business’s needs. The validation of the overall system requirements occurred before the high level system architecture was designed. There was also a validation step after the completion of each of the component’s requirement documents.

Several testing methods were used in this project. The methods were based on the situations and also varied according the development process followed. Code reviews, gray-box testing, black-box testing, usability testing, regression & automated testing all played a role in the successful testing of the components of this costing system.

7.1 Code Reviews

Because most of the programming was completed by someone other than the author, code reviews were a key component of the testing effort. All the code developed by others went through a code review by the author. Several things can be gained from good code reviews: 1) making sure the code followed the design, 2) making sure the code was understandable or commented so that it could be easily maintained in the future, 3) checking that the code followed the team’s coding standards and guidelines, 4) checking for correct use of object-oriented methods and variables, including correct visibility, and 5) gaining knowledge of the code to be used in gray-box testing, including boundary conditions. Additionally, once code is developed, duplicate code, used in different areas, may become apparent. Revising duplicate code into common methods is often required.

Code reviews were especially important in this project because often inexperienced interns or offshore consultants were used, and their coding
practices were not as good as those of an experienced programmer. Following an initial code review, corrections were made by the programmer and the code was then re-reviewed. In some cases it took several iteration of this process to get things correct.

### 7.2 Gray-box testing

Gray-box testing was used extensively by the author. Gray-box testing is a combination of white-box testing and black-box testing [16]. As with black-box testing, the requirements for the components were tested. Based on the code reviews, the author had a better idea of code that processed the functions, so some testing was completed with that knowledge in mind. In this gray-box testing, all functional requirements were tested for accuracy. Test cases were created for each functional requirement and were executed. Any problems found were sent back to the programmer for correction. Shown in Table 4 is one test case in which 12 scenarios, shown in Table 5 were run. These were different scenarios meant to test the functionality of the Unit Coster. The scenarios include the common situations, and the error handling capability of error situations.

<table>
<thead>
<tr>
<th>Test Case ID</th>
<th>Test Case Description</th>
<th>Prerequisite</th>
<th>Step#</th>
<th>Test Step</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCTC001</td>
<td>Verify the cost calculation for a selection from a product family that matches the specified scenario.</td>
<td>1. Use the Cost Template Designer to make sure the product family of the selection choices for testing has been setup scenario specified. 2. The queue table should be empty to start. 3. Determine the expected cost of the selection being used.</td>
<td>1</td>
<td>Start the Unit Coster running against the test database</td>
<td>no errors</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>Insert the selection into the cost calculation queue with SQL - insert into selection_queue_for_costing (selection_id) values (&lt;&lt;selection_id&gt;&gt;);</td>
<td>no errors</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td>Verify that the record is in the queue - select * from selection_queue_for_costing order by</td>
<td>Record exists</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>After waiting a minute, recheck the</td>
<td>Record should be gone</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5</td>
<td>Verify the records in the sel_pc_cost table - select prod_code, sel_pc_cost_cost_per_exh, sel_pc_cost_detail.* from sel_pc_cost, sel_pc_cost_detail where sel_pc_cost.selection_cost_id = sel_pc_cost_detail.selection_cost_id and selection_id = &lt;&lt;selection_id&gt;&gt;</td>
<td>Costs should match expected results</td>
</tr>
</tbody>
</table>

Table 4 - Test Case of Various Costing Method Scenarios

36
Table 5 - Test Case Scenarios Relating to Test Case in Table 4

| Scenario 1 | Part Set Costing (no overrides) |
| Scenario 2 | Part Set Costing (with ordering number overrides) |
| Scenario 3 | Part Set Costing (with list to cost override) |
| Scenario 4 | Part Set Costing (with net to cost override) |
| Scenario 5 | Ordering Number Costing (no overrides) |
| Scenario 6 | Ordering Number (with list to cost override) |
| Scenario 7 | Ordering Number (with net to cost override) |
| Scenario 8 | Price Table Costing (no overrides) |
| Scenario 9 | Price Table Costing (with ordering number overrides) |
| Scenario 10 | Model Number Costing (with ordering number overrides) |
| Scenario 11 | Ordering Number Costing with missing ordering number cost |
| Scenario 12 | Part Set Costing with missing product family cache file |

### 7.3 Usability Testing

For the components of the system that required a user interface, a usability test was performed with the actual users to get feedback as to how easy the system was to use and how intuitive it was to understand. Usability is a measure of appropriateness, functionality, and effectiveness of the interaction with the user is [16]. Usability testing was performed on these components: 1) the cost template designer, 2) JCHQ & UPS, 3) the Cost detail review tool, and 4) the cost rate maintenance and ordering number cost relationships maintenance screens. This is an important step because whether it is in a prototype development, or final release of an application, users should not complain about usability. Several good things were learned in the usability testing of these components. Those suggestions were incorporated in the final versions to meet the user’s needs.

### 7.4 Regression Testing and Automation

The JCHQ application had an automated regression test developed well before this project started. The regression test was setup to ensure that enhancements to
the application do not have a negative effect in some unforeseen part of the application. The regression test is maintained by Cognizant testing consultants. The regression test is mostly automated so that it can be easily re-run with every new release of the software. The JCHQ application is version of the Job Center application which is used by hundreds of sales associates every day, so it is very important to reduce the likelihood of bugs in the application. In addition to running the regression test, Cognizant also performed black-box testing of the JCHQ application based on the requirements. They built their own test cases and then had them reviewed by the author.

An automated test apparatus was also built to test the part cost DLL. This was required to allow for the maximum amount of test data to be run without user intervention. The test apparatus was built to run real manufacturing orders through the part cost DLL, obtaining the calculated cost and also pulling in the cost calculated by the existing MDP background process. This helped ensure that the new DLL, which was re-engineered from the MDP background process, was performing exactly like the source system.

In a similar manner, a semi-automated tool was created to compare the cost of committed orders, for all costing methods, to the manufacturing cost once the units were manufactured. The manual part of this test was to analyze each one that did not fully match. A good reason, such as a design special, was found for each case in order to prove that the Unit Coster was correctly costing units for all costing methods.

7.5 Component Testing and System Testing

As each component was developed, it was tested with the gray-box testing. For the DLLs, prior to the completion of the entire system, testing apparatuses were created to simulate the calling of the functions. For instance, before the Unit Coster application was developed, a test apparatus was created to simulate the
calling of the cost interface DLL in order to test that component. Prior to
developing the Cost Template Designer, the database tables were set up and could
be updated manually to allow the Unit Coster to run.

Once all the components in the system were developed, the overall unit cost
accuracy was checked. Running the Unit Coster, with all the components in
place, was done to collect test data, which were then compared to known results.
Additionally, via the Cost Detail Review tool, users were asked to help check the
results of the system. They could use the Cost Detail Review tool to verify the
costs of real jobs and orders running through the sales system, well before the cost
data was actually used in any discounting decisions.
8. Project Challenges

The challenges on this project were numerous. Fortunately there were no show-stopper challenges, but there were quite a few obstacles that took additional time and attention to overcome. One of the challenges faced was the turnover of consultants. As mentioned earlier in the paper, Cognizant is the main consulting company that provides IR with development consultants. During the year-long project, the author dealt with the loss of two very good consultants. The first, who was extremely knowledgeable about the system, left the project early due to an expired Visa. The second consult left at the end of 2011 due to a pending marriage in India. Turnover of consultants takes a great deal of effort to get them back up to the level of knowledge need to work efficiently.

Another challenge was the lack of strong business leadership. Good projects require a strong investment of time and interest from the business. It also takes someone at a high enough level to get participation from all parties. Early in the project, there was a good business leader, Rick Aldridge, who helped get the project kicked off. Rick left the company and it wasn’t until later in the project that a business leader came forward, Dan Wendl, VP of Sales, who was at a level with enough authority to drive organization alignment. Dan also has a wealth of Trane sales experience which aided in his leadership effectiveness in this area. During the in-between period, there was a great deal of conflicting voices with no single group who could enforce the project’s direction.

Changing requirements were not a big problem, but as with most projects, changes are a fact of life. Completing committed order costing was not part of the original plan, but was added in February 2012. Fortunately this effort was accomplished without too much additional work, as it was something that was already planned to be completed eventually. Another change was the inclusion of the warranty and freight cost calculation and the maintenance screens for the warranty and freight rate maintenance.
The varying manufacturing systems within Trane were also a substantial challenge. Each location and each product had differences which had to be accounted for. It took time to understand all the differences and to find the expertise at each location to help explain the systems. Examples of this were the three manufacturing systems, several sets of cost data (current, standard, and planned), and four costing methods. Even within the manufacturing locations that ran the same version of Cincom, the systems were set up to work differently. For example within the same Cincom database table, the Rushville locations stores a single item cost whereas other location store the cost of the entire quantity. Understanding and overcoming differences like this required several additional hours of work.

A positive challenge was learning a new programming language, ASP.NET, and some modern web development techniques. It took a bit of learning [14] and also collaboration with other groups at IR who were also experimenting with ASP.NET, but overall the experience was a good one. The skills and techniques developed can be used on other future projects.


9. Continuing Work

Maintaining and adapting the pre-sales costing system will continue into the future. Now that Trane has this ability, the system’s data will be incorporated further into the operations of the company. The cost data and margins will likely be incorporated into more reports. There is also a potential to use the data to transform the way the company manages the discounting decisions.

One of the most glaring needs that have not been addressed yet is the need for dynamic costing of the Lexington performance climate changer product. Lexington has several customized components to their product’s cost that will be a challenge to overcome, but the benefit is saving of about 6000 man-hours of manual work every time it needs to be re-costed, according to the airside product marketing leader. (Usually the product’s pre-sales costs are updated every couple of years.)

The same need applies to the La Crosse Centrifugal chiller products. This need is not as great because La Crosse has a process for updating their pre-sales costs that is simpler than Lexington’s process, but it still does require many hours of manual work to maintain. The costs for the La Crosse products that are outside of the MDP part pick process should be easier to account for than the Lexington product because La Crosse does not have the custom systems that Lexington does. This will still be a project that will take a good deal of dedicated effort.

The Fort Smith custom products are also not costed by this system. These products are totally customizable, but they are much lower-volume than the Lexington or La Crosse products, so this is a low-priority need right now. The only downside of not having a cost for all products at the time of discounting is for jobs with several products; they will have an incomplete total cost if just one of the products is not costed. The product manager for these products had a good suggestion recently. He suggested allowing a way for the product support team to
enter in a cost manually. Because this is manual process it is still not ideal, but it would provide a way to get a total cost on jobs with these custom products.

Another coming need is to integrate with Ingersoll Rand’s future enterprise-wide back office system. The company is investing heavily in this effort and the Cincom and UPDS systems are planned to be replaced with an Oracle EBiz system in each of the manufacturing plants. Although the future system is not yet designed, this costing system has the potential to easily integrate with that system.

Now that the costing is underway, the costs have begun to be pulled into the PROS system and used for pricing analytics. Bringing the costs into the Oracle BI system for other reporting needs will be another area of future work.
10. Conclusion

The costing system, designed to work within the Trane sales system, has automated some of the manual processes in the company. The system can calculate the cost of nearly all products in a near real-time speed. Components of the system were developed with solid software engineering principles, which should make future maintenance much easier. The process also provided the business with a system which matched their expectations. Combining several software engineering models to fit the needs of the components also proved to be a real success.

This system was a really good learning experience for how to successfully create software using several different software engineering models. There were some real challenges and not everything went exactly according to the plan, but overall it was a real success. The system should serve as the cornerstone for future enhancements and transformational changes to the sales organization and to the discounting process.
11. Bibliography


Appendix A: Cost Detail Review Tool Screens

Cost Detail Review Tool – Input by Specific Order or Job
Cost Detail Review Tool – Input by Product Family
Cost Detail Review Tool – Input by Product Code
Cost Detail Review Tool – Cost Detail Page
Appendix B: Cost Template Designer Screens

Costing Template Designer – Open Template dialog

Costing Template Designer – Costing Request screen
Costing Template Designer - Category Override screen
Costing Template Designer – Freight and Warranty Rate Maintenance screen

Costing Template Designer – Freight and Warranty Rate Maintenance - Add Product Codes dialog
## Appendix C: Job Center HQ – Price Rollup Screen

<table>
<thead>
<tr>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
<th>Column 4</th>
<th>Column 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>Data</td>
<td>Data</td>
<td>Data</td>
<td>Data</td>
</tr>
<tr>
<td>Description</td>
<td>Description</td>
<td>Description</td>
<td>Description</td>
<td>Description</td>
</tr>
<tr>
<td>Date</td>
<td>Date</td>
<td>Date</td>
<td>Date</td>
<td>Date</td>
</tr>
<tr>
<td>Location</td>
<td>Location</td>
<td>Location</td>
<td>Location</td>
<td>Location</td>
</tr>
<tr>
<td>Status</td>
<td>Status</td>
<td>Status</td>
<td>Status</td>
<td>Status</td>
</tr>
<tr>
<td>Notes</td>
<td>Notes</td>
<td>Notes</td>
<td>Notes</td>
<td>Notes</td>
</tr>
</tbody>
</table>

### Price Rollup Details

- **Product Code**: ABC123
- **Product Name**: Job Center HQ
- **Price**: $100.00
- **Units Available**: 100
- **Days Available**: 30

### Additional Information

- **Contact Person**: Jane Doe
- **Email**: jdoe@jobcenterhq.com
- **Phone**: 123-456-7890
Appendix D: Ordering Nbr Assoc. Maintenance Screen
Appendix E: Cost Rate Maintenance Screen
Appendix F: Full Class Diagram for Part Cost DLL