Establishing a Virtual Manufacturing Environment for Military Robots

by

Ryan J. Andersen
B.S. Computer Engineering, University of Utah 2003

Submitted to the MIT Sloan School of Management and the Department of Electrical Engineering and Computer Science in Partial Fulfillment of the Requirements for the Degrees of

Master of Business Administration
and
Master of Science in Electrical Engineering and Computer Science

In conjunction with the Leaders for Manufacturing Program at the Massachusetts Institute of Technology June, 2007

Signature of Author _____
MIT Sloan School of Management
Department of Electrical Engineering and Computer Science
May 11, 2007

Certified by _____
Jonathan Byrnes, Thesis Advisor (Management)
Senior Lecturer, Engineering Systems Division

Certified by _____
Steven B. Leeb, Thesis Advisor (Engineering)
Professor of EECS & Mech Eng

Certified by _____
Don Rosenfeld, Thesis Reader (Management)
Senior Lecturer, MIT Sloan School of Management

Accepted by _____
Debbie Berechman, Executive Director, MBA Program
MIT Sloan School of Management

Accepted by _____
A.C. Smith, Chair, Department Committee on Graduate Theses
Department of Electrical Engineering and Computer Science

This page is intentionally left blank

Establishing a Virtual Manufacturing Environment for Military Robots

by
Ryan J. Andersen

Submitted to the MIT Sloan School of Management and the Department of Electrical Engineering and Computer Science on May 11, 2007 in Partial Fulfillment of the Requirements for the Degrees of

Master of Business Administration

and

Master of Science in Electrical Engineering and Computer Science

ABSTRACT

Recent advances in the robotics industry have given the military an opportunity to capitalize on industry's innovation. Not only has core robotics technology improved but robotics manufacturing technology has also made significant improvements. This has enabled the U.S. military to procure robotic counterparts at an astonishing rate. The military expects to have a significant contingent of robots in the battle field by 2015. Today, "automated or remote-controlled machines are sniffing out mines, defusing explosives and watching for signs of danger".[1]

The Army's Future Combat Systems program to revolutionize the military into a quicker, faster, and more lethal force is predicated on the rapid development of robotics technologies.[2] Four out of the twelve vehicle systems being developed under the FCS program are unmanned vehicles requiring advanced robotics technologies. With this rapid pace of development a need for sound manufacturing environments exist to ensure product quality, availability, cost, and continuing innovation. This thesis presents a method for establishing a manufacturing environment for defense robotics that is both viable for industry cooperation and meets the demands of ongoing military procurement.

This thesis presents a framework for supplier selection that includes: 1) military contracting requirements and contract manufacturing, 2) public policy review, 3) key skill identification, and 4) manufacturing environment modeling. The research contained in this thesis was conducted in cooperation with iRobot, a military robotics supplier.

Thesis Advisors

Jonathan Byrnes (Management)
Senior Lecturer, ESD

Steven Leeb (Engineering)
Professor of EECS & Mech Eng

Reader

Don Rosenfield (Management)
Senior Lecturer, MIT

---

[1] Herper, Matthew. 2005. Robots of War, Future Tech. Available from http://www.frobes.com, Feb 7, 2005

[2] [14]

This page is intentionally left blank

ACKNOWLEDGMENTS

I would like to thank the staff at iRobot for allowing me to explore the selection process in my own unique way. I know that they took a chance on my project and a thank you for doing so is appropriate. Special thanks go to Rick Robinson, my immediate supervisor at iRobot. Rick was the man behind most of the efforts. Jeff Amberson significantly contributed to my experience at iRobot. Our theoretical discussion caused thought provocation and challenged some of my assumptions. I must also thank John Devine for the input and effort he expended to make sure I understood the manufacturing issues in the supply base for iRobot. Thanks to upper management as well, specifically Bill Ames and Bob Bell, for their routine meetings and assistance in getting access to information.

My thesis advisors are of course in need to a special acknowledgment, Steve Leeb and Jonathan Byrnes. They have continually challenged my ideas and have provided encouragement throughout this thesis work.

I recognize Salim Hart, a Sloan MBA for his financial metrics knowledge, and Barrett Crane of HP, a Leaders for Manufacturing alumnus, who has acted as my mentor the last year. Barrett provided some critical critiques of my thought processes and gave pointers to where I could take my work.

I must also thank my wife, who has made the greatest sacrifice in making this thesis come to life. She has put her life on hold while I left my job at Northrop Grumman and we moved with our four kids from California across the U.S. to Massachusetts. I have to acknowledge my kids: Ethan, Julia, Jordan, and Erin for putting up with my being gone most of the time while attending the LFM program.

This page is intentionally left blank

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

This page is intentionally left blank

1 Introduction

The U.S. military has a goal to transform one-third of its operational ground combat vehicles to unmanned robotic vehicles by 2015 and one-third of its operational deep strike force aircraft for unmanned operation by 2010.[3] Over the next 10 years the U.S. military will exhibit a strong demand for robotic counterparts. This demand and the increasing reliability the military is placing on these counterparts requires manufacturing process and systems that can repeatedly deliver quality products in a timely manner. Congress has recently weighted in on the importance of the unmanned vehicle program:

> *...the Congress--Recognizes the significant and lifesaving contributions of Unmanned Ground Vehicles to current combat operations, and notes the need for increased funding for development and deployment of UGVs.*[4]

The government is also increasing its use of small businesses to perform contracting work. About 23% of all Defense spending goes to companies that are classified by the government as small.[5] Small businesses that do not have the manufacturing capabilities to produce product in the quantities the military requires may turn to contract manufacturers.

The last 10 years have brought substantial growth in the contract manufacturing industry.[6] This is an industry that consolidates the manufacturing of other companies' products to gain economies of scale, diversification, and to provide flexibility and speed. Today, many companies outsource manufacturing to contract manufacturers because the cost of doing so is less than what could be done internally.

Some of these contract manufactures are performing services for the Department of Defense. Using contract manufacturers to perform defense work however can increase the risk in meeting the Department of Defense's objectives. Products developed under

---

[3] H.R.5408.IH, Sec. 220(a), 106th Congress
[4] H. CON. RES. 113, 110th Congress, 1st Session
[5] Reardon, Elaine, "The Department of Defense and its Use of Small Businesses: An Economic and Industry Analysis", RAND 2005
[6] Flextronics, Solectron, Foxcon, SCI, and Celestica SEC filings

Department of Defense contracts can at times be of high complexity and stretch the forefront of technology. Separating the design of these products from their manufacture can delay product improvement cycles and mask quality problems. Thus, there is a need in this rapidly expanding area to have a process by which to evaluate and select contract manufacturers that will reduce risk while attaining the cost savings inherent in manufacturing consolidation.

This thesis presents a process by which defense contractors can select contract manufacturers for the manufacture of robotics. There are four main reasons why a defense contractor would find the process outlined in this thesis useful. The process stipulates: 1) simulation of the supply chain for cost minimization, 2) risk reduction and strategic positioning through macroeconomic and political investigation, 3) technology alignment, and 4) objective supplier selection, which is required in defense contracting as outlined in the Federal Acquisition Regulations.

There are numerous articles and papers written on supplier selection, but few cover it with the breadth found here. Presented in this thesis are: macroeconomic, political, and technical aspects for making the outsourcing placement decision. Placement of Defense contracts is specifically considered, but by removing the political aspects, the process could be used for general contract placement.

iRobot, a military robotics firm, sponsored the research for this thesis. iRobot is working on the Army's Future Combat System (FCS) program developing a small unmanned ground vehicle known as the SUGV. iRobot is in the process of selecting a lead contract manufacturer to perform the final assembly of the SUGV, which provided an intriguing research question: "what is the best way to select a lead contract manufacturer in this environment?"

This chapter has briefly introduced the applicability of this thesis. The remainder provides additional context for the thesis research and an overview of the thesis' structure. In section 1.1 an introduction to iRobot is provided. Section 1.2 provides an overview of the Department of Defense's Future Combat System program, iRobot's largest defense program. Section 1.3 provides a description of iRobot's existing

manufacturing environment. Section 1.4 introduces the Leaders for Manufacturing internship that led to this thesis. Section 1.5 describes the thesis goal. Section 1.6 provides an overview of the process flow presented in this thesis. And, section 1.7 provides an outline for the remaining chapters.

1.1 iRobot

The majority of the research for this thesis was conducted at iRobot, who hosted a six month Leaders for Manufacturing internship focused on Defense robotics supplier selection. iRobot is a robotics firm, headquartered in Burlington Massachusetts, specializing in mobile robots for home and military applications. iRobot was founded by three roboticists from MIT in 1990. The first products were mainly targeted for military applications (some of these products can now be found in museums). Over the years iRobot has grown into a multi-national company with revenues over $188 million ($167 million from products and $21 million from services, including military contracts[7]). On November 15, 2005, iRobot celebrated its IPO with proceeds of $70.4 million.[7] iRobot now has 371 full-time employees and a substantial number of interns and contractors.

iRobot is organized in two main divisions: Government and Industrial and Home Robots. The Government and Industrial division oversees all aspects of military commercial off the shelf (COTS) products and contracts to provide research and development. The main COTS product line, in the government and industrial division, is the Packbot mobile tactical robot shown in Figure 1-1.

---

[7] iRobot Press Release 18 May 2005, case 05-068 Available from http://www.irobot.com/sp.cfm?pageid=86&id=147&referrer=169

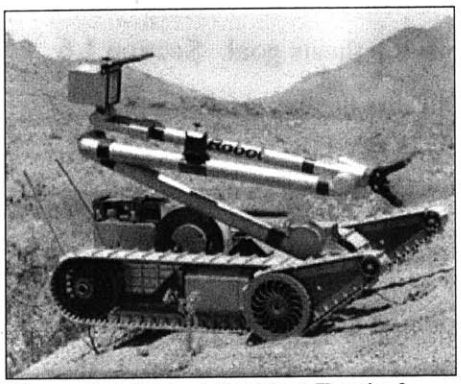


Figure 1-1 iRobot Packbot Tactical Mobile Robot

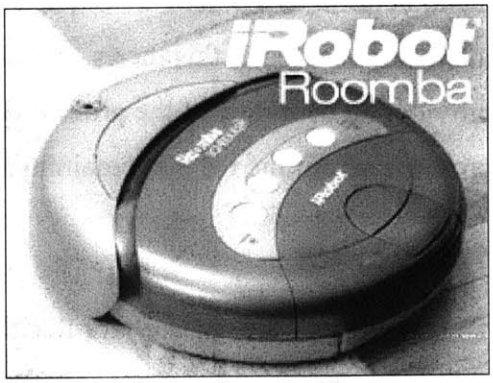


Figure 1-2 iRobot Roomba Floor Vacuuming Robot

The home robots division oversees all aspects of robots for general consumers. The lead product line on the consumer side is the Roomba™ vacuuming robot shown in Figure 1-2. The Roomba™ is an autonomous robot capable of vacuuming floors on a predetermined scheduled. Revenue and Gross Profit Margins between the two divisions are shown in Figure 1-3.

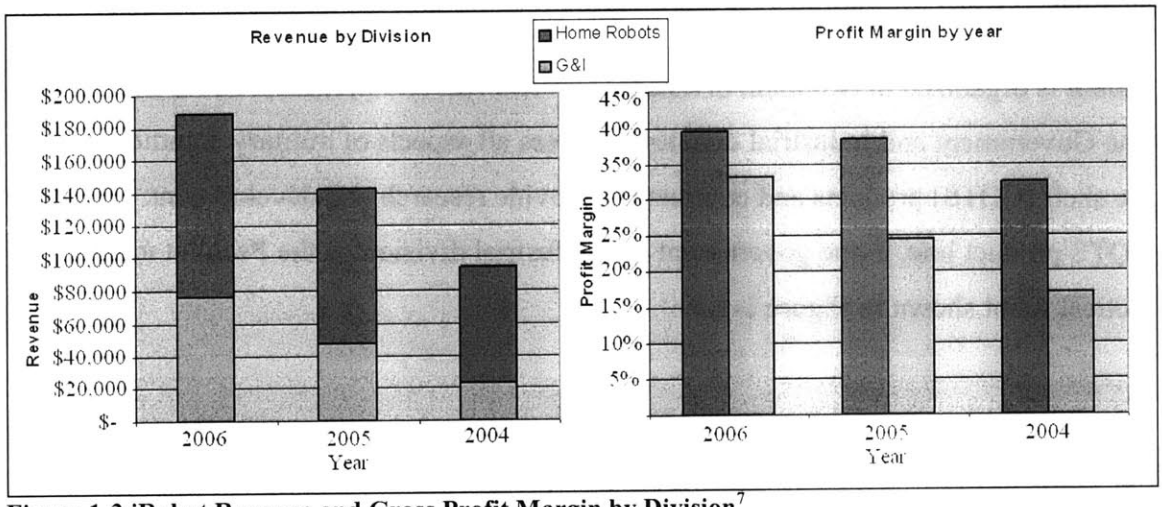


Figure 1-3 iRobot Revenue and Gross Profit Margin by Division[7]

The research presented in this thesis was performed under the guidance of iRobot's Government and Industrial Division. The Government and Industrial division has about five major projects under research and development contracts from various organizations within the Department of Defense. The largest, as measured by obligated dollars, is a small portion of the Army's FCS program, the SUGV. The SUGV was the stimulus for

this thesis; an introduction to the FCS program and the SUGV are provided in the following section.

1.2 The FCS Program

The Future Combat System (FCS) program is a $108 Billion Department of Defense program to transform the Army from a slow and heavy fighting force to a fast and light force capable of meeting the challenges of the 21st century. The research and development phase of the FCS program began in 2003 and is expected to take in excess of 10 years to complete.

The Army thinks of the development of the FCS program as a System of Systems; a system of systems meaning: one system to control a host of systems. The FCS program consists of developing 18 manned and unmanned ground vehicles, air vehicles, sensors, and munitions. These systems are illustrated in Figure 1-4. The scope of the FCS program is unprecedented in the history of the Army's development efforts.[8]

[8] GAO-05-428T, "Defense Acquisitions, Future Combat Systems Challenges and Prospects for Success", 2005

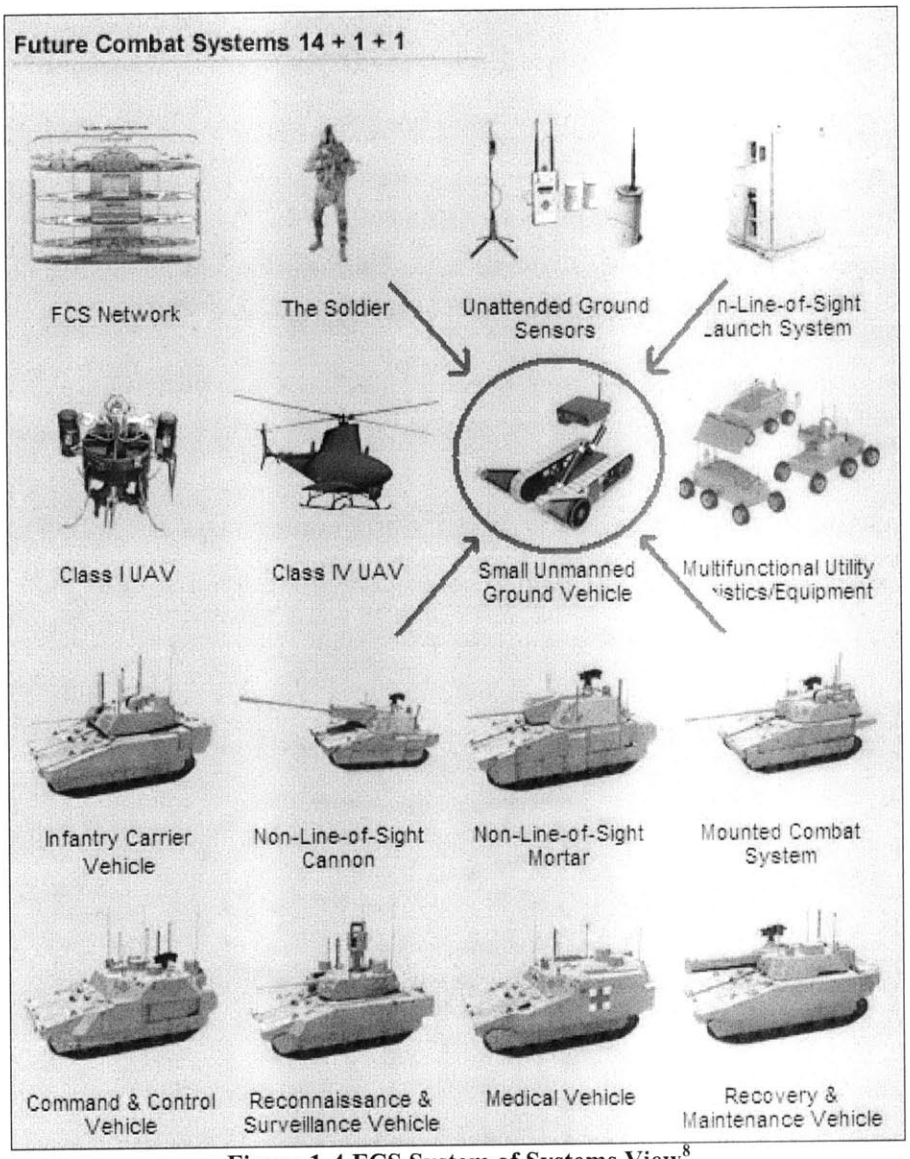


Figure 1-4 FCS System of Systems View[8]

iRobot has been given the research and development contract for the Small Unmanned Ground Vehicle (SUGV), a contract expected to be worth $51.4 million--a significant program for iRobot.[9]

---

[9] iRobot Press Release 18 May 2005, case 05-068 Available from http://www.irobot.com/sp.cfm?pageid=86&id=147&referrer=169

1.2.1 Small Unmanned Ground Vehicle

The SUGV is a small, man-portable, vehicle capable of supporting military operations in urban terrain tunnels, sewers, and caves. The SUGV's main purpose is to provide intelligence, surveillance, and reconnaissance (ISR) without exposing soldiers to harm. The SUGV is required to weigh less than 30 pounds and be capable of carrying plug-and-play payloads weighing up to 6 pounds.[10]

iRobot was selected for the SUGV contract award for a number of reasons. One of the key contributing factors was iRobot's existing Packbot™ product line of mobile tactical robots. The Packbot line has proven its usefulness to soldiers over the years and has become a well known necessarily in the Iraq and Afghanistan wars as measured by sales.

1.2.2 FCS Concerns

Despite the positive outlook for the SUGV, recent FCS program office budgetary issues and missteps have created some uncertainty in the future of the FCS program. This uncertainty could have significant implications for iRobot.

A recent government accountability office (GAO) report, released on March 16, 2005, brought to light some of the concerns around the FCS program plan. The GAO stated (bold added for emphasis):

> *The FCS [program] is at significant risk for not delivering required capability within budgeted resources. Currently,* ***about 9½ years is allowed*** *from development start to production decision. DOD typically needs this period of* ***time to develop a single advanced system, yet FCS is far greater in scope****. The program's level of knowledge is far below that suggested by best practices or DOD policy: Nearly 2 years after program launch and with $4.6 billion invested, requirements are not firm and* ***only 1 of over 50 technologies are mature****. As planned,* ***the program will attain the level of knowledge in 2008 that it should have had in 2003****, but things are not going as planned. Progress in critical areas—such as the network, software, and requirements—has in fact been slower, and FCS is therefore*

---

[10] [2]

> *likely to encounter problems late in development, when they are very costly to correct.* ***Given the scope of the program, the impact of cost growth could be dire.***[11]

It is not clear what the consequences of Congress' view of the FCS program will be or how Congress will react to future program set backs. Given the uncertainty of the FCS program, a few strategies are available to the smaller contractors, like iRobot, to mitigate some of the risk. First, a smaller program, say a class I unmanned aerial vehicle (UAV), that is currently part of the FCS program can petition military commanders to have the vehicle removed from the FCS program and brought to Congress as a separate program. Second, the class I UAV contractor can appeal directly to Congressional delegates to maintain authorization and appropriations for their UAV despite general FCS cutbacks.

There are risks associated with the FCS program and doing business in general with the Department of Defense. These risks are outlined further in chapter 2 with the overview of defense contracting. iRobot's manufacturing environment will be discussed in the next section to provide the constraints the SUGV faces at iRobot.

1.3 iRobot Manufacturing

The Packbot™ line of robots is manufactured by a variety of third party contract manufacturers. These manufacturers range in geographical distribution and capabilities. Prior to iRobot's IPO, all production manufacturing was outsourced to contract manufacturers, including assembly and fulfillment activities. As it stands today, iRobot does not touch its products before they go to customers. An excerpt from iRobot's 10-k provides the reasoning for this decision (bold added for emphasis):

> *Our core competencies are the design, development and marketing of robots. Our* ***manufacturing strategy is to outsource non-core activities,*** *such as the production of our robots, to third-party entities skilled in manufacturing. By relying on the outsourced manufacture of both our consumer and military robots, we can focus our engineering expertise on the design of robots. Using our engineering team of over 100 roboticists,*

---

[11] GAO-05-428T, "Defense Acquisitions, Future Combat Systems Challenges and Prospects for Success", 2005

> *we believe that we can rapidly prototype design concepts and products to achieve optimal value, produce products at lower cost points and optimize our designs for manufacturing requirements, size and functionality.*[12]

iRobot views itself as a design, development, and marketing company, with manufacturing secondary in terms of competence. In both the Government and Industrial division and the Home Robots division, a lead integrator has been identified to coordinate the supply chain and fulfill customer orders. For the Government and Industrial division, manufacturing is led by Gem City Engineering, a metal tool products manufacturer located in Dayton Ohio. Separately, the Home Robots division has its manufacturing led by the Jetta Company Limited located in China.

1.4 Supplier Selection Problem Introduction

The contract award for the SUGV was only for the design and development phases of the system. The production contract for the SUGV will be awarded at a later time. iRobot's deliverable for its existing contract is a few prototypes and a set of documentation sufficient to reproduce the prototypes in a production environment. It is anticipated that if iRobot performs well on the design phase, then the follow-on procurement contract will be awarded.

In preparation for the procurement contact bid, iRobot sought to understand the requirements the government would place on it in a manufacturing environment. This effort was to determine if its existing supply base was capable of supporting a procurement contract for the SUGV under the FCS program. If it was found that the existing environment was not capable of supporting the requirements, then iRobot would seek to determine what manufacturing environment would meet these requirements. The question of capability arose because the military products being manufactured were generally classified as commercial-off-the-shelf (COTS). These products were not required to be manufactured with the stringent requirements that the Department of Defense routinely puts on military grade equipment. To illustrate the difference between military grade manufacturing and COTS manufacturing, if the government wanted to

---

[12] iRobot 2006 10-K filed with the SEC

purchase a component for field use it would go directly to a supplier and make the purchase from the supplier's inventory, which is how the Packbot is routinely purchased by military organizations. If, however, the government wanted to have a product manufactured specifically for field use, there are a set of rules that must be adhered to in the design and manufacturing of the product. These rules can be as stringent as stipulating the nationalities of the labor force.

iRobot's research question for the Leaders for Manufacturing internship could be summarized as: who should iRobot use as its lead integrator and fulfillment service provider on the SUGV program given the uncertainty of the FCS program? Not only does the selection need to be technically sufficient but the selection process itself must also be sufficiently objective to meet Federal Acquisition Regulation requirements.

1.5 Thesis Goal

The goal of this thesis research was to develop an objective supplier selection process for military contractors that are seeking to outsource manufacturing. The selection process has three main objectives.

The first objective is reducing the risk to the Department of Defense and the Defense Contractor of outsourcing manufacturing to a contract manufacturer. To meet this objective, the process analyzes relevant macroeconomic and political aspects in the geographical regions where potential suppliers reside. The process also simulates the manufacturing and logistics environment to provide advanced warning of supply issues.

The second objective is to provide a defensible supplier selection decision by keeping the process objective. This objectivity is met by: 1) systematically finding qualified contract manufacturers, 2) independently developing selection criteria, and 3) evaluating those contractors in a systematic manner.

The final objective of this thesis is to provide a process with general applicability to other supplier selection problems.

1.6 Approach

A six step approach was instituted to meet iRobot's supplier selection problem. The first step was to determine relevant literature on contract manufacturing selection. The second was to analyze the Department of Defense's requirements for design and manufacturing on the FCS and SUGV programs. The third step was to determine the capabilities of iRobot's existing supply base and contrast it with the Department of Defense's requirements. The fourth step was to find qualified contract manufacturers, manufactures that are capable or can be made to be capable of meeting the Department of Defense's goals. The fifth step was to develop a set of criteria to eliminate potential contract manufacturers. And the sixth step was to evaluate and eliminate those contract manufactures that fall short of these criteria.

This thesis provides a supplier selection process, as illustrated in Figure 1-5, that consists of the following tasks: 1) aggregating potential suppliers from multiple sources, which included: directories, personal contact or networking, approved vendor lists, existing suppliers, and industry lists, 2) developing industry, business, financial, and location filters to rapidly eliminate potential suppliers from the aggregated pool, 3) developing key skill criteria relevant to both the product and the process being considered, 4) issuing a Request for Pricing/Proposal and soliciting feedback, and 5) simulating operations with these suppliers and evaluating the results.

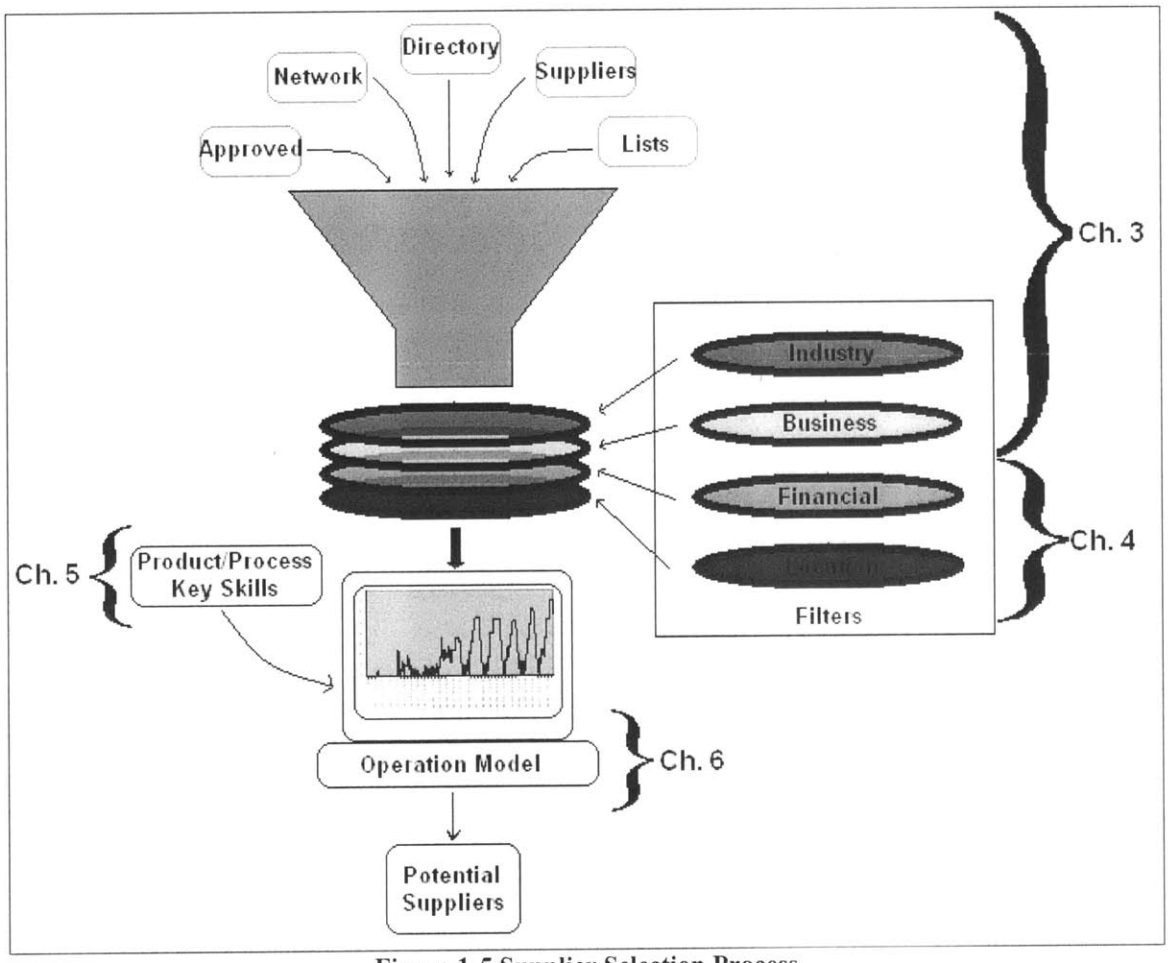


Figure 1-5 Supplier Selection Process

1.7 Thesis Overview

The research conducted at iRobot is prevalent throughout this thesis. It is illustrated as a use case for this selection process. An overview of each chapter that follows is provided below.

Chapter 2 provides an introduction to military procurement and contract manufacturing as they are relevant to the decisions made in this thesis. The risks associated with contract manufacturing in this context are also presented.

Chapter 3 introduces the SUGV and presents the supplier finding methods used by iRobot for this research. The chapter includes the development of the first two filters, business and industry, for iRobot's SUGV supplier selection problem. The quantitative aspects of iRobot's selection through the first two filters are provided here.

Chapter 4 covers the development of the third and fourth filters, location and financial. The location filter is developed by utilizing a state risk assessment to evaluate supplier locations based on macroeconomic factors. The chapter includes a discussion of the implications of the state risk assessment for states that are seeking new investment. The role congress plays in Defense contracting is also covered, including a regression analysis of historical congressional allotments developed by Rundquist[13]. The risk reduction strategies available to Defense contractors are discussed with an example provided. The financial filter is developed through financial ratios relevant to manufacturing operations. The quantitative aspects of iRobot's selection through the last two filters are provided here.

Chapter 5 provides a framework for developing the key skills needed in a manufacturing operation. An overview of the robotics industry is provided as it relates to the key skills iRobot must anticipate its contract manufacturers possess. And an analysis of the SUGV's subsystems is presented.

Chapter 6 introduces the operations model for manufacturing environment simulation. A potential manufacturing environment for iRobot is detailed and simulated. The results of the simulation are discussed.

Chapter 7 summarizes the thesis and the applicability of this work to other industries.

---

[13] Rundquist, Barry S., "Congress and Defense Spending: the Distributive Politics of Defense Spending," University of Oklahoma Press, 2002.

This page is intentionally left blank

2 Defense Contracting with Contract Manufacturers

This chapter provides an introduction to military procurement and contract manufacturing as they are relevant to the selection of contract manufacturers in the Defense industry. The risks associated with outsourcing and defense contracting are discussed as well.

2.1 Defense Contracting

Defense contracting is any type of contracting that is funded by the Department of Defense (DOD). This overview of defense contracting should not be considered comprehensive; it is merely a brief overview to provide the context for this thesis. For a more complete reference visit the Department of Defense's web site; guides are provided there to help companies to start working with the defense department.

The Department of Defense's mission is to provide the US military with forces needed to deter war and to protect the security of our country[14]. In 2006 the Department of Defense was appropriated $363 billion dollars in Government Fiscal Year (GFY) 2006 for operations, procurement, and research and development projects. The Department's budget is set by Congress but recommended by the president.

2.1.1 Department of Defense Requirements

The DOD, using its long term objectives and marketing from contractors sets the goals for weapons development for future years. They will package the programs they would like to have researched, developed, evaluated, and tested in preparation for contractor bidding. Depending on the contact value the bidding process formality will vary. For small projects the Small Business Initiative Research (SBIR) system is used while larger contacts create a prime and sub prime contractor structure. For some purchases, the DOD may just direct the contract to a specific contractor, but in this case the DOD must be able to justify this "sole source" or "directed" contract using objective measures.

Once a contract has been awarded restrictions on contract execution will be detailed, but these restrictions can be negotiated. Sourcing requirements play a particularly interesting

---

[14] [3]

role in defense contacts as a contractor generally must source the majority of labor and material content from domestic sources if available.[15]

2.1.2 Appropriations vs. Authorization

The funding of Department of Defense programs is a two step process. Each body of the legislature, both the House and the Senate, must approve (authorize) DOD programs each year. Approval does not allocate funds to the program however. Funding (appropriation) is provided through a separate resolution in the Congress. The Congressional Research Services has provided the following brief on the difference between authorization and appropriation:

> *In this two-step process, the authorization and appropriation functions are separated. First, Congress considers* ***authorization bills*** *making substantive policy -bills that establish, continue, and change agencies, programs, and activities and set the terms and conditions under which agencies and programs operate. The authorizing bills also may recommend spending levels for programs and activities. After the authorization bills are enacted, Congress considers annual* ***appropriation measures****. These bills generally provide funding (or budget authority) for the authorized agencies, programs, and activities. Under this process, the appropriation acts may provide less than the amounts recommended in the authorization acts, or may provide no funding at all for authorized programs*[16].

For example, if the Department of Defense wanted to create a new line of weapons to meet a specific threat, it would submit the request to Congress through the President's office. Congress upon receiving the request reviews the weapon system as it deems necessary and either authorizes or rejects the system. At a later date, Congress would then determine the level of funding to allocate to the development of the system. Typically, the authorization bill provides a target funding level, but Congress is not obligated to adhere to this level unless obligation is specifically called out in the

---

[15] H.R.5803.ENR, "Department of Defense Appropriations Act", 1991

[16] Tyszakiewicz, Mary T., Daggett, Stephen, "A Defense Budget Primer." CRS Report RL30002

authorization bill. If funding is appropriated to the program then work can commence on the legislated start date. Funding is not guaranteed even if authorization is granted.

2.1.3 Research and Development vs. Procurement

There are three main types of programs the DOD includes in its budget request[17]. The first, known as **Military Personal and Operations**, provides the pensions, pay checks, and routine maintenance of equipment to run ongoing military operations and readiness.

The second, **Procurement**, is the acquiring of systems, weapons, or other equipment that immediately goes to use by military personal. This is also known as a reset, getting the military back to a particular readiness level.

The third program type is **Research, Development, Test, and Evaluation**. This program type provides for the research, development, evaluation, and testing of future military systems. These are programs that may not ever be procured for use by military personnel but their development is worthwhile to the military's objectives. Once a program has matured through the Research, Development, Test, and Evaluation phase of funding, it is ready to be procured for use by military personnel.

Returning to the weapon system example, assume Congress has approved the appropriation of funds in the Research and Development phase for the system. A defense contractor will use the funds to produce a requirements specification, architectural design, detailed design, and manufacturing guide for the system. These documents should be of sufficient detail to allow contractors skilled in the area to modify, improve upon, and manufacture the system. This detailed documentation segments the development of a system from its corresponding manufacture and allows the Department of Defense to allocate development to those firms that are effective at design and manufacture to those that are the effective at manufacturing.

Once the weapon system has been completely developed, generally with working prototypes, the DOD will submit a budget request to the President for the procurement of

---

[17]H. Report. 109-119, "House Defense Appropriate Act Committee Report", 2006

the system. This procurement will fund the manufacture of the system and get it into the hands of military personnel. Once the system is established in the field, and degradation sets in, funding requests for reset will be submitted under additional procurement programs. If the weapon system requires ongoing maintenance or expends ammunition as a result of normal operations a recurring budget request will be submitted to the President under the operations and military personal category.

2.2 Contract Manufacturing Overview

This section provides a brief overview of the contract manufacturing industry and introduces a few of the elements of outsourcing that are relevant to this thesis.

The contract manufacturing industry was created in the early 1980's when IBM entered the Personal Computer (PC) market. IBM did not consider the development of the PC core to its business and as such asked Microsoft to develop the operating system and Intel to provide the computer chips. Since this early time of outsourcing, corporations have looked inward to determine core capabilities outsourced as resources allowed[18].

A company that manufacturers products for another company is referred to as a Contract Manufacturer (CM), and a CM that manufactures electronic components is referred to as an Electronic Contract Manufacturer (ECM). The hiring company, the company with the consuming customers, is referred to as the Original Equipment Manufacturer (OEM).

2.2.1 Assembly Services

ECMs typically deal with the manufacturing of components using machines and tools. There is a new breed of CMs, however, that is focusing on providing assembly services, assembling larger systems out of the manufactured component parts. For example, an assembly CM might work for a printer OEM. In this case it would have inputs of: cases, power supplies, circuit boards, cartridges, cables, boxes, marketing material, etc. Its output would be a boxed printer ready for shipment to the customer.

---

[18] Luthje, Boy. Electronic Contract Manufacturing: Global Production and the International Division of Labor in the Age of the Internet, Industry and Innovation, Dec., 2002.

2.2.2 Fulfillment Services

Some CMs have moved into the fulfillment services area as well. Fulfillment is filling OEM customer orders directly. The product flows directly from the CM to customers without going through the contracting OEM's facilities. In some cases, returns can even go directly back to the CM. iRobot currently uses CMs with manufacturing, assembly, and fulfillment services and, at this point, will only consider CMs for future products that perform all three of these activities: 1) electronics manufacturing, 2) assembly, and 3) fulfillment.

2.2.3 Obsolescence

Obsolescence occurs when a product is no longer available. This can happen due to a number of reasons, a few of which are: 1) as a result of decreasing demand a supplier will discontinue the production of a product and 2) as a result of government regulation, such as RoHS[19], the manufacturing process must change, which stipulates product reengineering. Obsolescence is of concern in this thesis for two reasons. First, OEMs do not want their supply base to hold inventory that may become obsolete and consequently written-off. And second, OEMs do not want their suppliers to discontinue vital parts required for the OEM's product.

To illustrate this, assume a robotics company is using a gyroscope component in a robot. The robot's requirements stipulate operation at -13 degree C. The gyroscope specifications across the industry only allow for -10 degree C operation. Because a suitable gyroscope can not be found, the robotics manufacturer tests gyroscopes until one is found that consistently works. Upon finding a gyroscope that works reliably at -13 degrees C, the robotics manufacturer goes into production. What can occur next is an upgrade (revision) by the gyroscope manufacturer. The new gyroscope meets the published specification of -10 degrees C but no longer functions at the -13 degrees C it previously had.

---

[19] The Restriction of the use of certain Hazardous Substances [12]

2.3 Potential Risks

There are four main risks in utilizing contract manufacturers for Defense contracts.

The two step congressional funding process. The authorization vs. appropriations process can deceive contractors. Even though congress authorizes a program it does not guarantee funding. Funding is only guaranteed after an appropriations bill is passed.

Procurement not guaranteed. Despite being awarded a research and development contract, the subsequent procurement contract will be available for bid to all qualified Defense contractors. Procurement contracts with contract manufacturers should anticipate this scenario. iRobot is attempting to mitigate this by showing a high readiness level for a procurement contract.

Communication Barriers. By removing the manufacture of a component from its design, delays and miscommunication can cause adverse effects.

Manufacturing Procurement. Once a weapon system is ready for procurement, contract manufacturers are typically in a better cost position to fulfill a procurement contract than the designer. iRobot has significant contributed intellectual property in its designs and hopes this will mitigate this risk.

2.4 Chapter Summary

Defense contracting and contract manufacturing each have risks. Combing contract manufacturing with defense contracting can increase those risks. But there are risk mitigation strategies available, which will be discussed in the next two chapters.

3 Supplier Selection Process Introduction

This chapter introduces iRobot's SUGV, which is being developing for the FCS program. The selection process illustrated in Figure 1-5 is introduced and is followed by discussing the methods of finding potential suppliers. The development of the first filter, Industry, is presented along with iRobot's criteria for it. The second filter, business, is also presented with iRobot's criteria. Quantitative aspects of iRobot's selection process are provided through the first two filters.

3.1 Product Description

This section describes the SUGV in more detail. The SUGV's actual requirements are classified as government sensitive so a mock set of requirements are used in order to illustrate the process. These requirements will not only simplify the explanation of the SUGV but they will also allow this thesis to focus on those items that most differentiate the SUGV with regards to manufacturing.

3.1.1 Functional requirements

The SUGV's mock functional requirements are as follows:

1. The vehicle must be capable of traveling 30 miles on desert terrain without recharging.
2. The vehicle must weight less than 30 lbs.
3. The vehicle must be capable of a controlled ascent and descent at an incline of up to 60 degrees.
4. The vehicle must be capable of being remotely controlled without line of sight at a distance of 2 miles.
5. The vehicle must provide remote video surveillance in both the infrared and visible spectrums.
6. The vehicle must be capable of optically zooming the IR and visible spectrum cameras up to 30 times.
7. The vehicle must be capable of navigating through a 40" diameter pipe.
8. The vehicle must be capable of withstanding active jamming devices.
9. The vehicle must have an AUPP of under $30K.
10. The vehicle must be capable of service for 5 years.
11. The vehicle must be constructed in such a way that field operators can replace 80% of failed parts.
12. The vehicle must be capable of carrying 6 lbs. of payload and provide 40 Watts of continuous power.
13. The vehicle must operate using class A4b military batteries.

14. The vehicle must be capable of providing video around 90 degree corners without exposing vehicle body.

3.1.2 Subassembly identification

The requirements stipulate systems and components that provide the following:

Requirement	Implied Systems
1	Motors, Tracks, Gears, Motor Controllers, Heat dissipation, Power system
2	Light weight components
3	Flippers, center of gravity manipulation, neck and head required.
4	Antenna, Radio, computer system.
5	IR, Visible Light cameras. Video encoders. Large bandwidth.
6	Lenses, lens controllers
7	Independent track control. Restricted width and height.
8	Complex circuitry and computer algorithms. High processing capability.
9	Inexpensive parts, investment in tooling
10	Durable parts and materials.
11	Subassembly approach with line replaceable units (LRU)'s
12	Payload bay, connections, extra power.
13	Common battery connection.
14	Cameras must be in head, head must be capable of pan and tilt.

Table 3-1 SUGV Implied Systems

Figure 3-1 provides a rough illustration of the SUGV to assist in the explanation of systems and subsystems that comprise this mock vehicle. There are 14 main subsystems that comprise this vehicle, many of which are shown. The subsystems not shown are: the radio, the electronics package, and the computer system. These subsystems all reside in the chassis.

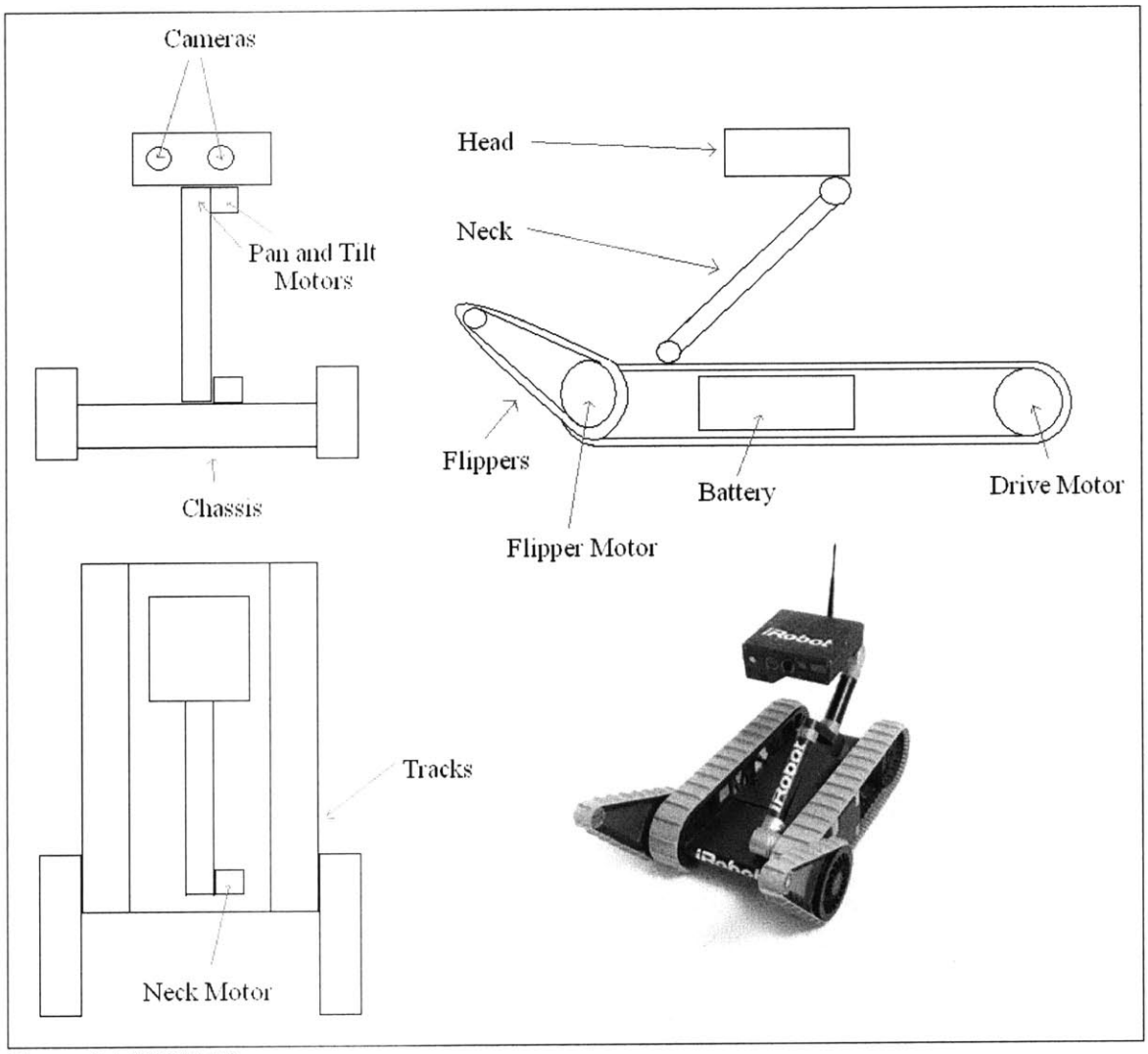


Figure 3-1 SUGV Diagram

Two subsystems are of note here, the flippers and the neck. The flippers allow the robot to ascend and descend stairs. Without the flippers, the robot would topple down stairs on descent and could not even begin to ascend stairs. The flippers can also be used to flip the robot over if it gets overturned.

The neck is the other noteworthy subsystem. This subsystem comprises interconnect electronics and three actuators for the panning and tilting of the "head" and the angling of the "neck". The panning capability comes from the requirement to look around corners without exposing the chassis. The tilt requirement arises because the camera must be in the head and the driver must be able to ascend and descend stairs. The tilt allows the

operator to see the tracks and stairs while the chassis is inclined. The neck will be discussed in more detail in a later section as it causes some issues in manufacturing.

Requirement number 11, that 80% of failed parts be replaceable by field operators, requires a modular approach to manufacturing. Components must be swappable by field operators as parts fail. To meet this requirement, the vehicle is composed of three line replaceable units or LRU's. These units are: head and neck, chassis, and flippers (tracks are replaceable too, but not classified as LRU's). It can be seen from the illustration that the head and neck LRU contains the cameras and three motors, the flippers contain basic mechanical parts, and the chassis contains everything else. This approach enables the field operator to replace parts without sending the entire vehicle back for repairs.

3.1.3 Manufacturing and Logistics Needs

iRobot expects the CM to provide the following services: supply chain management, final product integration, final testing, packaging, and fulfillment. iRobot also expects the CM to store LRUs in sufficient quantity to meet customer demand for replacement units.

iRobot will develop the testing harnesses and procedures, but it will be up to the CM to perform the testing activities. Testing results must be collected and incorporated with in field part failures to develop a database for root cause analysis. The DOD logistics system also known as Performance Based Logistics (PBL), requires the ongoing monitoring of failures to determine inventory levels at repair depots and service stations. Automated part ordering is required in support of PBL[20].

3.2 Constraints on Supplier Selection

The FCS program has placed a few specific requirements on iRobot due to the sensitivity of the SUGV. These requirements limit the breadth of the search for potential lead CMs to build the SUGV. As a result, the CMs that will be considered are those that demonstrate the following criteria:

---

[20] GAO-04-715 "Opportunities to Enhance the Implementation of Performance-Based Logistics", 2004

1. The lead CM must be owned by a domestic interest.
2. The lead CM must assure that foreign persons will not have access to documentation, material, or the assembly area.
3. The lead CM must allow the government to retain ownership of all tooling.
4. The lead CM must be capable of attaining a secret government clearance.

3.3 The Selection Process

The selection process consists of: 1) finding companies through available sources, 2) filtering those companies based on four objective filters: industry, business, location, and financial, 3) soliciting feedback on requirements using an RFI/RFP, and 4) simulating the manufacturing environment. The selection process is outlined in Figure 3-2 with the expected number of potential suppliers identified on the edges.

Figure 3-2 Selection Process Flow

3.3.1 Initial Finding

The initial finding activity is a process of getting as many of the relevant CMs in the selection pool as possible. iRobot did not want to constrain their selection of a lead integrator to a few local or popular candidates. Due to this broad approach however, an automated form of filtering potential CMs was adopted, a combination of the industry and business filters. The finding methods used by iRobot are discussed in the remainder of this section.

3.3.1.1 Networking

Networking is the process of using past personal experience along with the experiences of colleagues and friends to find companies that might be appropriate for the work to be done. A systematic networking effort was not undertaken but a formal solicitation for suggestions was extended to the iRobot production group. iRobot also considered current suppliers in the networking sourced category.

The companies found through networking are not subjected to the first three filters, industry, business, and location. This is due to a higher level of confidence in these

companies' abilities to perform the work. iRobot felt that these companies should bypass the majority of the automated rejection criteria.

3.3.1.2 Directories

Directories provide company lists that may not be obtained through other finding methods. Directories often come in the form of databases on the internet or subscription based CDs. The directories used to source potential CMs for the SUGV were Dun & Bradstreet and CorpTech. Dun & Bradstreet was selected because of the breadth of companies included, around 1.6 million.[21] CorpTech was selected because of its focus on the high tech industry. iRobot also used an industry list, the Top 50 Contract Manufacturers provided by Electronics Supply & Manufacturing.

3.3.2 Filtering

Once a significant number of companies were identified, the filtering process began. Those companies that were sourced through networking (personal contact) were not filtered at this stage of the process. The filter at this stage consists of industry alignment and business size by revenue. Each of these filters is discussed in detail in the following sections. Figure 3-3 illustrates the filter process with the expected number of companies indicated on the edges (the location filter will be discussed in the next chapter).

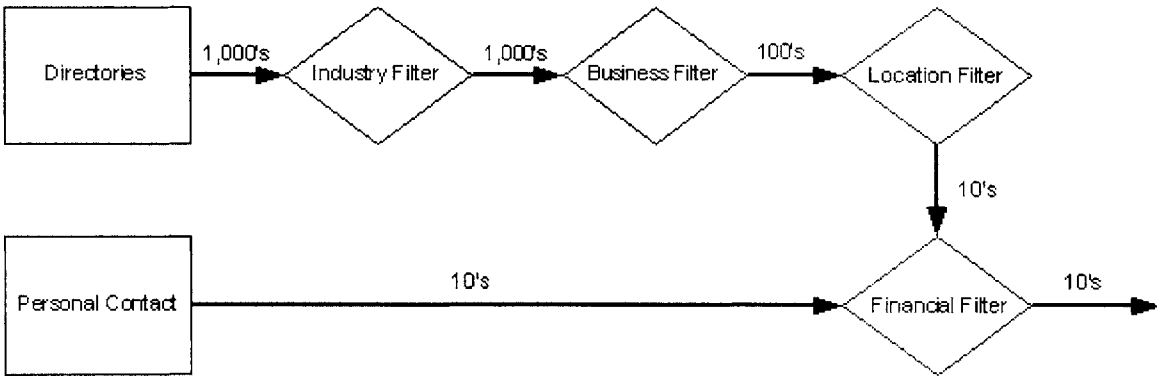


Figure 3-3 CM Filtering Flow

[21] [4]

3.3.3 Industry Filter (NAICS Codes)

The Office of Management and Budget (OMB), part of the US federal government, has developed an industry classification system for statistical analysis of the economy. In this system, each economic entity (company) is assigned a North American Industry Classification System (NAICS) code that corresponds to the processes used to produce goods or services.

Each company is given a single number by the Census Bureau based on which processes the company uses to produce the majority of its revenue, but access to these numbers is restricted. As a result, companies self identify a few NAICS codes under which they feel they operate. These companies then provide these codes to the directory companies (D&B, Reference USA, etc.).

In this part of the filtering process, a review of the NAICS codes was undertaken. The Census Bureau supplied a list of NAICS codes that was categorized by industry and function. Using the anticipated processes for production of the SUGV, several NAICS codes were selected that indicate similar operations.

iRobot primarily selected companies in defense industries that involved a high level of electrical and mechanical integration and test. Some medical device manufacturers were also selected because of the similarity in quality control and electro-mechanical integration required. Table 3-2 captures the NAICS codes that were selected for this filter.

NAICS Code	NAICS Code Description
334511	Search, Detection, Navigation, Guidance, Aeronautical, and Nautical System and Instrument Mfg.
334519	Other Measuring and Controlling Device Manufacturing
336412	Aircraft Engine and Engine Parts Manufacturing
336413	Other Aircraft Parts and Auxiliary Equipment Manufacturing
336414	Guided Missile and Space Vehicle Manufacturing
336419	Other Guided Missile and Space Vehicle Parts and Auxiliary Equipment Manufacturing
339112	Surgical and Medical Instrument Manufacturing
339113	Surgical Appliance and Supplies Manufacturing

Table 3-2 NAICS Filter Codes

3.3.4 Business Filter (Revenues)

The second filter was to eliminate those companies that were not the right size given then product complexity and volume. It was understood that companies in the directories and lists were of sizes ranging from the two people operation to the one hundred thousand person operation. A target revenue range had to be determined. Consideration was given to the following: 1) the CM must have the funds necessary to finance the inventory and operations required to produce the SUGV, 2) the CM must remain solvent through the sales cycle and random demand shifts, and 3) iRobot did not want to be such a small portion of the CMs revenue that it cannot demand attention when needed.

The size was determined by iRobot's desire to represent between 20-30 percent of the CMs total revenue. iRobot feels that this is enough to get responsiveness form the CM and not too much that demand swings could endanger the financial position of the CM.

Here is the process used:

$$BusinessPercent = \frac{iRobotRevenue}{ExistingCMRevenue + iRobotRevenue}$$

Rearranging to solve for the Existing CM's revenue:

$$ExistingCMRevenue = \frac{iRobotRevenue}{BusinessPercent} - iRobotRevenue$$

Soling for iRobot's revenue assuming 10 years of production and a total demand of 7,666 units, iRobot's revenue from SUGV is:

$$iRobotRevenue = \frac{TotalUnitDemand}{NumberOfYears} \times AUPP$$

$$iRobotRevenue = \frac{7666}{10} \times \$30000 = \$23M$$

And lastly solve for the existing CM's revenue at 20% and 30%:

$$ExistingCMRevenue = \frac{\$23M}{20\%} - \$23M = \$92M$$

$$ExistingCMRevenue = \frac{\$23M}{30\%} - \$23M = \$53M$$

Therefore the existing CM's revenue should be between $53M and $92M. The business filter was then applied to the potential supplier pool.

3.4 Quantitative Aspects of Industry and Business Filters

After applying the first two filters to the potential supplier pool there are roughly 1,000 companies left. Figure 3-4 illustrates the filtering process to this point.

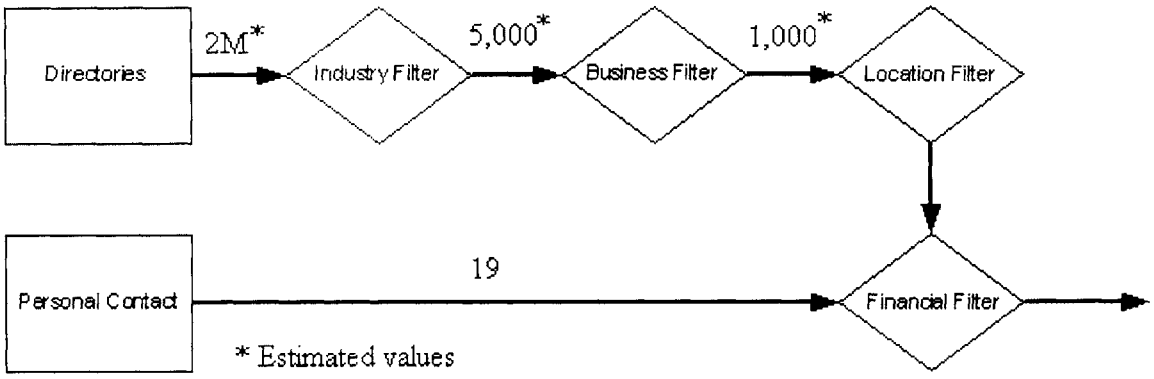


Figure 3-4 Quantitative Results after Industry and Business Filters

3.5 Chapter Summary

This chapter presented the SUGV and its major subassemblies. Companies were found through several sources and the aggregated pool of suppliers was filtered by the first two filters, which were developed in this chapter. Chapter 4 will continue the filtering process by generating the location and financial filters and applying those to the remaining suppliers.

This page is intentionally left blank

4 Location and Financial Filters

This chapter develops the location and financial filters of the selection process. The location filter is developed through the use of detailed a state risk assessment that prioritizes states based on macroeconomic factors. A discussion of the implications of the state risk assessment for states that are seeking new investment is provided here. In addition to the state risk assessment, the role that congress plays in Defense contracting is also covered, including a regression analysis of historical congressional allotments presented by Rundquist[22]. Understanding the role that Congress has in Defense contracting can reduce the risk of working with the DOD. Because congressional districts are geographically based, this discussion has been included in the location filter development section. The financial filter is developed in this chapter through the use of financial ratios relevant to manufacturing operations. The quantitative aspects of iRobot's selection through the last two filters are provided here.

4.1 State Risk Assessment (Location Filter Part 1)

The State Risk assessment is a process of systematically collecting and analyzing data on locations to determine the relative best fit for an operation. This effort is largely based on A.T Kearney's paper Where to Locate[23] and thesis work at MIT by Steven Vasovski[24].

This assessment collected data from a variety of sources, a few of which include: The Bureaus of Labor and Education, the National Center for Education Statistics, the Bureau of Economic Affairs, the Bureau of Labor Statistics, and the Census Bureau. A description of each data item pulled from these sources follows along with an argument for its inclusion in the risk assessment. A team from iRobot was polled to determine relevance to their operations and the manufacture of the SUGV. For each criterion, each state is ranked from 1 to 50 based on that State's position as determined by the particular criterion. The state with the best possible condition was assigned a 1 and toe worst was

---

[22] Rundquist, Barry S., "Congress and Defense Spending: the Distributive Politics of Defense Spending," University of Oklahoma Press, 2002.
[23] A.T. Kearney. "Selecting a Country for Offshore Business Processing: Where to Locate," 2003
[24] Vasovski, Steven. "A Global Sourcing Strategy for Durable Tooling." MIT, 2004

assigned 50. This ranking allowed for the sorting and comparison of the states across the set of criteria.

4.1.1 Gross State Product

The Gross State Product (GSP) is a measure of the economic goods or services produced by the state. The aggregate of all state's GSP is the Gross Domestic Product (GDP) of the nation. The GSP servers as a measure of the economic size of the state and has relevance in this case as an indicator of the opportunities a supplier has in receiving goods or services in a timely and cost effective manner. The GSP for each state is taken from the Bureau of Economic Analysis[25].

4.1.2 Gross State Product Compound Annual Growth Rate

The Gross State Product Compound Annual Growth Rate (CAGR) is a year over year average growth rate of the state's GSP. This is an indicator of how quickly the economy is growing in the state. This is useful in comparing states especially when one state's economy is declining and another increasing. The source of the GSP CAGR is the Bureau of Economic Analysis' summary statistics by State[26].

4.1.3 NAICS Industry Size

The NAICS Industry Size is the portion of the GSP that comes from the NAICS economic entities that iRobot has identified. This gives a comparative indication as to the level of the targeted industry size in each state. The size of the industry relates to the expertise of the labor force in the industry. The NAICS specific industry sizes were taken from the Bureau of Economic Analysis.[27]

4.1.4 NAICS Industry Compound Annual Growth Rate

The NAICS Industry Compound Annual Growth Rate (CAGR) is the NAICS industry growth rate on a year over year basis. This is an indicator of how quickly the industry is growing in the state. This provides an indication of in availability of the industry

---

[25] Bureau of Economic Analysis, Summary Statistics 1997-2005 available from http://www.bls.gov/
[26] Ibid
[27] Ibid

workforce. These data were taken from the Bureau of Economic Analysis summary statistics.[28]

4.1.5 Total Labor Force

The total labor force is an indication of the number of people either working or looking for work in the state. It is not broken out by industry or function. The data comes from the Bureau of Labor Statistics.[29]

4.1.6 Unemployment Rate

The unemployment rate is the ratio of people registered with the government as unemployed and actively seeking employment divided by the total labor force. This rate gives an indication as to the economic well being of the state. A low unemployment rate indicates a good economy, which results in greater availability of goods and services. A high unemployment rate tends to indicate social issues and other underlying economic problems in the region. iRobot assumes that employees are willing to work for new entrants regardless of the general unemployment rate. The unemployment rates by state are taken from Bureau of Labor Statistics.[30]

4.1.7 Hourly Wage

The hourly wage is the average hourly wage earned by all legal hourly workers in the state. The hourly wage is relevant because it indicates the costs that are associated with working in the state. The Bureau of Labor Statistics provided the state by state hourly wages.[31]

4.1.8 NAICS Industry Hourly Wage

The NAICS Industry Hourly Wage is the average hourly wage earned by all legal hourly workers in the NAICS industries identified by iRobot. The relevance of this item is its

---

[28] Bureau of Economic Analysis, Summary Statistics 1997-2005 available from http://www.bls.gov/
[29] Bureau of Labor Statistics, Summary Statistics 2005 and Summary Statistics 2001-2005 Available from www.bls.gov
[30] Ibid
[31] Ibid

indication of the costs to produce not only in the state but also in the specific industries iRobot is targeting. The Bureau of Labor Statistics provided this information.[32]

4.1.9 Public Debt

The Public Debt is the amount of debt the state itself has. The amount of public debt in a state tends to indicate the economic conditions present in the state. A highly leveraged state may resort to cutting social programs or increasing taxes as time progresses. The lower the public debt rate, the better the economic conditions tend to be. The public debt rates were provided by the US Census Bureau.[33]

4.1.10 Education Attainment

The Education Attainment item is a percentage of individuals within the state who have been awarded at least a bachelors degree. This is an indicator of the population's ability to solve the myriad of problems that come up in industry. The educational attainment by state was provided by the Bureau of Education.[34]

4.1.11 Literacy Rates

Literacy Rates indicates the general competence of the population. Due to the high literacy rates in the US, this data is no longer collected consistently in the form of literate versus illiterate. Because literacy rates could not be collected a proxy was used. An assessment is done of children in the 4^{th}, 8^{th}, and 12^{th} grades that effectively measures the level of literacy from both a verbal and mathematical perspective.[35] The average of these assessments within a state was used to compare the competency of the state with other states. The assessment is called the National Assessment of Education Progress (NAEP). The following is an excerpt from the National Center for Education Statistics FAQ:

---

[32] Bureau of Labor Statistics, Summary Statistics 2005 and Summary Statistics 2001-2005 Available from www.bls.gov

[33] US Census, Financial Data, 2005 available from http://www.census.gov/

[34] US Census, Education Attainment available from http://www.census.gov/population/socdemo/education/cps2005/tab01-01.xls

[35] National Center for Education, Digest of Education Statistics, 2005 Available from http://www.nces.ed.gov/programs/digest/d05/tables/dt05_112.asp

NAEP, or the National Assessment of Educational Progress, is often called the "Nation's Report Card." It is the only measure of student achievement in the United States where you can compare the performance of students in your state with the performance of students across the nation or in other states. NAEP, sponsored by the U.S. Department of Education, has been conducted for over 30 years. The results are widely reported by the national and local media...[36]

The NAEP scores were provided by the National Center for Education Statistics.[37]

4.1.12 Corporate Tax Index

The Corporate Tax Index is an indicator of the relative cost of doing business in a state. It does not represent an absolute cost or percentage of sales but it does provide guidance as to which states are cheaper to operate in. The Tax Foundation, a non-profit Washington monitoring group, has produced the Business Climate Index. This index is a measure of the relative tax burden on a per state basis. It is provided by the Tax Foundation.[38]

4.1.13 Distance from Headquarters

The distance from iRobot's headquarters was an important aspect of the state risk summary. As a result of outsourcing, travel is bound to be required. A convenient travel time becomes increasingly important as supply issues develop. The straight-line distance between state capitals was used as a proxy for the true travel time. These distances were provided by Proximity One.[39]

4.1.14 State Risk Assessment Data

The State Risk Assessment data are included in Appendix A -. The following sections develop the weighting criteria for each element and select the top seventeen sates for consideration. The number seventeen was selected to result in a manageable number of companies for the financial filter.

---

[36] National Center for Education, Digest of Education Statistics, 2005 Available from http://www.nces.ed.gov/programs/digest/d05/tables/dt05_112.asp
[37] Ibid
[38] Tax Foundation, "State Business Tax Climate Index 2006," Available from http://www.taxfoundation.org/taxdata/show/1371.html
[39] [13]

4.1.15 Weighting Development

A survey was taken of each development and production area on the SUGV to determine the macroeconomic factors that were relevant for the manufacturing operations being planned. The poll of iRobot operations personnel resulted in the following weighting factors regarding the state risk assessment:

Factor	Weight
Economic Growth	7%
NAICS Industry Growth	12%
Unemployment Rate	7%
NAICS Average Wages	21%
Public Debt	7%
Education Level	9%
Literacy Levels	5%
Tax Rates	14%
Distance from iRobot	18%

Table 4-1 State Risk Assessment Weights

4.1.16 State Risk Assessment Results

By applying the weights to the state risk assessment data, included in Appendix A -State Risk Assessment Data, the top 17 states for iRobot to operate in are: Vermont, Delaware, South Dakota, New Hampshire, Virginia, Idaho, Wyoming, North Dakota, Florida, Montana, Georgia, Alaska, Nevada, Kansas, Tennessee, Maine, and Rhode Island. The scores for each individual state are include in Figure 4-1. States are ranked from 1 to 50, with 1 being the best possible condition and 50 the worst.

	RANKS										
State	State Growth Rate	NAICS Industry Growth	Unemployment	NAICS Average Wage	Public Debt	Education Level	Reading	Taxes	Distance	Score	State
Vermont	20	16	3	8	23	8	7	47	6	15.64	Vermont
Delaware	19	21	16	23	9	21	18	8	10	16.15	Delaware
South Dakota	14	20	10	7	13	28	8	2	38	16.64	South Dakota
New Hampshire	21	50	6	28	27	4	4	6	3	17.56	New Hampshire
Virginia	5	14	4	40	10	9	11	18	13	17.63	Virginia
Idaho	6	1	7	2	3	41	23	24	45	18.23	Idaho
Wyoming	2	45	5	4	1	44	10	1	39	18.76	Wyoming
North Dakota	26	3	2	10	17	30	2	31	35	19.25	North Dakota
Florida	3	10	8	31	37	23	41	4	24	19.97	Florida
Montana	15	30	11	14	24	27	6	9	41	21.54	Montana
Georgia	18	17	34	26	4	18	40	21	23	22.07	Georgia
Alaska	10	6	48	3	49	26	35	3	50	22.16	Alaska
Nevada	1	2	15	18	38	36	44	5	46	22.2	Nevada
Kansas	32	12	32	12	32	14	14	34	30	22.74	Kansas
Tennessee	28	4	41	24	6	38	34	15	27	22.83	Tennessee
Maine	35	35	23	5	33	40	3	45	7	23.08	Maine
Rhode Island	17	23	29	17	46	20	31	49	2	23.25	Rhode Island
Maryland	12	38	14	44	11	5	32	22	11	23.46	Maryland
Iowa	43	13	20	9	5	39	13	42	28	23.51	Iowa
Utah	13	15	17	21	36	12	28	19	43	23.66	Utah
Massachusetts	27	28	24	48	50	2	1	27	1	24.45	Massachusetts
Alabama	33	27	12	20	26	46	46	14	26	25.59	Alabama
Pennsylvania	41	41	27	27	47	29	15	16	9	25.77	Pennsylvania
Minnesota	25	8	13	43	25	10	9	38	29	26.27	Minnesota
Oklahoma	16	47	18	15	7	43	33	17	36	26.28	Oklahoma
Wisconsin	36	29	21	22	39	25	17	32	22	26.36	Wisconsin
District of Columbia	7	7	46	51	2	1	51	46	12	26.44	DC
Indiana	40	9	37	41	14	47	30	11	18	26.52	Indiana
Arkansas	31	18	25	11	12	50	38	41	31	27.16	Arkansas
West Virginia	46	37	28	6	31	51	42	35	15	27.46	West Virginia
Texas	8	26	35	34	28	35	36	7	37	27.76	Texas
Colorado	9	32	31	42	34	3	20	12	40	27.83	Colorado
Oregon	24	5	45	30	42	24	27	10	49	28.2	Oregon
New Jersey	38	24	19	49	29	6	5	50	8	28.35	New Jersey
Nebraska	44	25	9	25	15	33	12	43	33	28.76	Nebraska
North Carolina	22	43	33	29	8	42	37	37	17	29.67	North Carolina
Mississippi	47	40	51	1	16	49	48	29	32	29.72	Mississippi
New York	29	42	30	33	51	13	19	51	5	29.72	New York
Missouri	48	39	38	38	22	16	21	20	25	29.91	Missouri
South Carolina	37	11	49	35	44	32	39	30	19	30	South Carolina
Washington	23	19	40	37	45	15	22	13	47	30.14	Washington
Connecticut	42	48	26	45	35	7	24	39	4	30.31	Connecticut
Illinois	45	22	42	46	43	19	26	23	21	31.17	Illinois
Ohio	49	31	43	32	18	34	16	48	14	31.27	Ohio
Louisiana	30	33	50	16	19	45	45	36	34	31.77	Louisiana
Michigan	51	34	47	36	30	37	29	26	16	31.8	Michigan
Arizona	4	46	22	39	20	17	43	25	44	32.07	Arizona
New Mexico	34	51	36	19	21	31	47	28	42	33.18	New Mexico
Kentucky	50	44	44	13	48	48	25	44	20	33.32	Kentucky
California	11	36	39	47	40	11	49	40	48	38.04	California
Hawaii	39	49	1	50	41	22	50	33	51	40.4	Hawaii

Figure 4-1 State Risk Assessment Scores

4.1.16.1 Public Policy implications of the State Risk Assessment

This is a section highlighting the implications of public policy on the state's attractiveness for new business. Based on iRobot's consideration of states based on macroeconomic and socioeconomic conditions, a state's public policy can effect investment. iRobot's selection of locations to consider based on macro and

socioeconomic conditions is a real-world case of supplier selection and as such should be valuable to states looking to attract companies like iRobot.

The public policy decisions that states have made in the past have an effect on the types of businesses that decide on the margin to locate within them. State governments have some control over these issues and as such are in a position to influence the types of businesses that select their states for operations.

Most of the factors included in the State Risk Assessment are not directly controlled by public policy; taxes and public debt are the exception. States can influence, but not control the following areas of the assessment: state growth rate, industry growth and wages, unemployment, education level, and reading scores. Over time all of these factors can be affected by public policy but immediate adjustments are limited to taxes and public debt.

To illustrate the use of this data, let us assume that Massachusetts would like to take the position held by Rhode Island in the risk assessment, Rhode Island is the state with the lowest score of the 17 that were included by iRobot for consideration. Massachusetts would need to lower its score from 24.45 to something below 23.25, let's say 23.24.

$$\Delta Score = NewScore - OldScore$$
$$\Delta Score = 23.24 - 24.45$$
$$\Delta Score = -1.21$$

The score is composed of the relative position of the criterion times the weighting factor.

$$Score = \sum_n Criterion_n Rank \times Weight_n$$
$$\Delta Score = \sum_n Criterion_n NewRank \times Weight_n - \sum_n Criterion_n OldRank \times Weight_n$$
$$\Delta Score = \sum_n Weight_n \times (Criterion_n NewRank - Criterion_n OldRank)$$

By limiting the public policy changes to those factors that are directly controllable through public policy the equation can be simplified to:

$$\Delta Score = (NewPublicDebtRank - OldPublicDebtRank) \times PublicDebtWeight$$
$$+(NewTaxRank - OldTaxRank) \times TaxWeight$$

Solving for the new ranks:

$NewTaxRank =$

$$\frac{\Delta Score + (OldPublicDebtRank - NewPublicDebtRank) \times PublicDebtWeight}{TaxWeight} + OldTaxRank$$

$NewPublicDebtRank =$

$$\frac{\Delta Score + (OldTaxRank - NewTaxRank) \times TaxWeight}{PublicDebtWeight} + OldPublicDebtRank$$

At this point, the relative difficulty of making tax or debt changes would be considered. Let us assume that Massachusetts can make changes to the tax code to reduce its tax rank from 27 to 25, effectively matching Arizona's tax rates. The remainder of the public policy adjustment must be made through Public Debt. Solving for the new public debt rank yields:

$$NewPublicDebtRank = \frac{-1.21 + (27 - 25) \times 0.14}{0.07} + 50 = 36.7 = Rank : 36$$

Given this result, Massachusetts must adjust its public debt to be ranked 36th in the assessment. Utah currently holds the 36th position, which means Massachusetts must select a public debt policy just below Utah's. Utah's public debt is currently at 15.8791%, which means Massachusetts' debt rate should be at least 15.879% (public debt is measured as a percentage of GSP) therefore:

$$PublicDebtPercentage = \frac{PublicDebt - DebtToRetire}{GSP}$$

$$DebtToRetire = PublicDebt - PublicDebtPercentage \times GSP$$

$$DebtToRetire = \$72898.064M - 0.15879 \times \$328535M = \$20.792B$$

In order for Massachusetts to be considered over Rhode Island, the state legislature would need to reduce taxes commiserate with Arizona and retire $23.792B in debt.

4.2 Congressional Consideration (Location Filter Part 2)

This section discusses the consideration that can be given to congressional districts in selecting locations for Defense work. The research conducted at iRobot in preparation for this thesis reviewed these aspects but did not include them in the location filter. iRobot felt that the best possible technical solution was the best outcome, but I will present it for completeness.

4.2.1 Congressional Budgeting

It is helpful to understand the flow of funds in defense contracts to understand the risks in defense contracting. The funding process originates with military personal determining their needs and concludes with Congressional acts authorizing programs and appropriating money to pay for them.[40]

4.2.1.1 Committee structure

Once a military program request goes to Congress for approval or funding it is not evaluated by all representatives. Congress is divided into committees and sub-committees, with each specializing in a particular matter of government. For example, the House of Representatives has 24 committees[41], which range from Taxation to Agriculture.

---

[40] Tyszakiewicz, Mary T., Daggett, Stephen, "A Defense Budget Primer." CRS Report RL30002
[41] [5]

There are two committees in each of the Senate and House that are relevant to defense contracting for the purposes of this thesis. The House's Armed Services and Appropriations committees, and the Senate's corresponding committees by the same names. Under these committees, sub-committees exist to perform the ground work required to research program requests and develop the laws that will be presented to the general committee and later to the full legislative body for formal enactment.

The House of Representatives' Armed Services Committee, for example, has 7 sub-committees[42] ranging from terrorism to sea power. The rules of these sub-committees are determined by the committee under which they are formed.

The four subcommittees that this thesis will consider are the following:

1. House Armed Services Committee/Tactical Air and Land Forces Subcommittee
2. House Appropriations Committee/Defense Subcommittee
3. Senate Armed Services Committee/Subcommittee on Air Land
4. Senate Appropriations Committee/Subcommittee on Defense

These sub-committees represent both the authorization of programs related to this thesis and the appropriations of funds to those programs. Given that these subcommittees have the final authority as to what programs are authorized and how much funding is allocated, an effective risk mitigation strategy is to understand the needs and motivations of the subcommittee members. The case of Rockwell Collins on the B1-B bomber project is presented here to illustrate the potential mitigation.

4.2.1.2 Rockwell Collins and the B1-B Project

There is a significant gain possible by leveraging the Congressional districts of members of Congress. Defense contracting brings money and jobs to Congressional districts throughout the United States. These districts would not otherwise have this economy boosting activity. When defense programs with production in these Congressional

---

[42] [6]

districts are threatened it can be advantageous to contact the Congressional representatives to plead the case for continuation.

The efforts of Rockwell Collins on the B-1 bomber project exemplify the usage of Congressional districts as a funding continuation strategy. The B-1 was being developed as a replacement bomber for the B-52 during the 60's and 70's. In the early 70's mission requirement changes caused a significant change in the B-1's design, as a result, a new designation was given to the B-1, the B-1A. The B-1A's cost per plane increased over its development life from $30M to over $100M, typical for large defense projects, but within expectations for the mission requirement adjustments[43]. As a result of the cost increases, President Carter cancelled the B-1A program in 1977 stating "the B-1 bomber is an example of a proposed system which should not be funded and would be wasteful of taxpayers' dollars."[44]

President Carter's cancellation of the B-1A was a controversial matter. While the costs seemed prohibitive, the Mutual Assured Destruction (MAD) doctrine demanded a deep penetration vehicle capable of reaching far into Soviet defenses. The B-2 bomber seemed an appropriate vehicle to replace the B-1, but its development required at least a decade above and beyond the B-1's planned development. Carter argued that intercontinental ballistic missiles (ICBM)'s could be used as a stop gap. With wide support in the democratic House and Senate the B-1A program was canceled.

Ronald Reagan, after taking office in 1981, reconstituted the B-1 program under the designation B-1B. A few changes were made in the mission requirements but the program was back in front of Congress for approval. President Carter enjoyed democratic majorities in both the Senate and the House when he canceled the program. Reagan, however, upon submitting his B-1B budget, only had a republican majority in the Senate. Approval of the B-1B program looked unlikely.

---

[43] [7]
[44] [1]

Rockwell Collins and the Air Force underwent an extensive campaign to spread defense subcontracts across Congressional districts.

> *Rockwell and Air Force lobbyists armed themselves with meticulous lists of every B-1 subcontract location, cross-referenced by state, town, and Congressional district. The studies, prepared both by Rockwell and by the Air Force comptroller's office, showed how many dollars of B-1 money flowed into a Congressional district each month. This information allowed the lobbyists to show members of Congress down to the last dollar how their constituents benefited from the B-1. The data became even more potent as subcontractors, majors and union leaders were enlisted to lobby their members of Congress*[45].

This lobbying activity increased the popularity of the program in Congress and won the appropriations and authorization needed to solidify the program. The B-1B had its maiden flight three years later.

By law, military officers are forbidden from lobbying Congress or coordinating with the defense industry to do so. There is no restriction however on the Defense industry's lobbying Congress. In fact, Congress often relies on lobbying activities to bring critical information to light.[46] See for a more complete reference on the subject. Regardless of lobbying activities however, considering facilities in key Congressional districts should not be overlooked.

4.2.1.3 Congressional makeup

The Congressional makeup of the Senate and House in 2006 is considered in this case. During this time, Republicans controlled the presidency and both Houses of Congress. Speculation on elections or power shifts in Congress is not considered in this review. There are two aspects of Congressional makeup that are detailed. The first is the number of representatives that sit on an FCS related defense committee, this gives a notion of the power that state has with respect to the FCS program and program continuation

---

[45] Kotz, Nick, "The Chesapeake Bay goose hunt, the beautiful secretary, and other ways the defense lobby got the B-1," Washington Monthly, 1988. Available from http://www.encyclopedia.com/doc/1G1-6351543.html

[46] Skroggs, Steven. "Army Relations with Congress: Thick Armor, Dull Sword, Slow Horse." Praeger, 2000

opportunities. The second is the percentage of representatives from a state that sit on an FCS related defense committee. This gives an indication of how important FCS related programs are to the state (representatives are limited as to the number of committees they can participate with).

Figure 4-2 illustrates not only the number of representatives from each state on the relevant committees but also the percentage of representatives from the state that sit on these committees. For example, California has 3 representatives that sit on the relevant committees, indicated by the shading, but only 6 percent of their total Congressional delegation is represented on these committees. Contrast this with Hawaii's having 3 representatives on the relevant committees, which represents 75 percent of Hawaii's total delegation. To summarize, in Figure 4-2, the shading indicates power while the percentage indicates importance. The complete data set can be found in Appendix B - Congressional Representation Data.

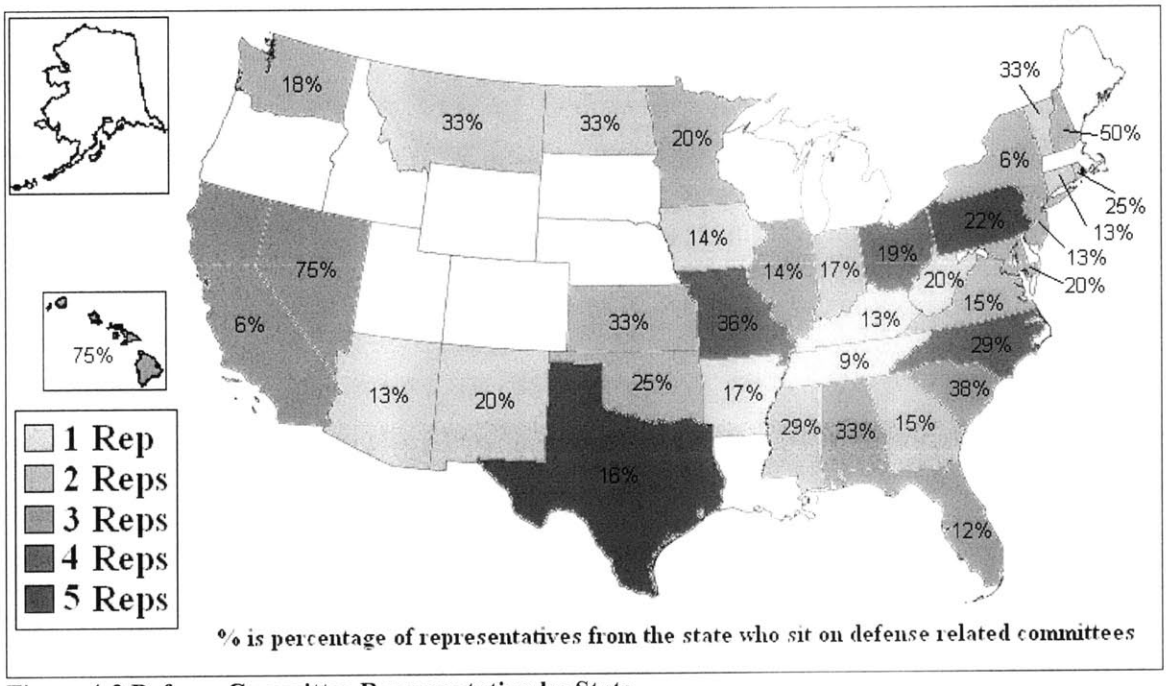


Figure 4-2 Defense Committee Representation by State

The notion that the number of representatives represents power is not entirely true as individual seniority within the committee also plays a role. In addition there may be other reasons that a representative would sit on a defense committee besides maintaining

funding levels for the state. The dataset does however provide a general feel for the situation in Congress and the role the state plays in defense programs.

4.2.1.4 Regression analysis of previous defense appropriations

A deeper look into defense expenditures in the past is needed to understand the impact Congressional districts have on defense outlays on average. Rundquist[47] has done just this for the government fiscal years 1963 through 1995. Rundquist considers the following for each Congressional district: the previous year's benefits, representation on a defense committee by party, Senate and House partisanship, ideology, district capacity, and economic conditions.

Rundquist's conclusion was somewhat surprising. Representation by a democrat on a House defense committee brought in $11 per capita over republican representation. In the Senate, representation by a republican on a defense committee brought in $41 per capita over democratic representation. It was found that party representation made the most significant impact on defense spending allocations.

4.2.2 Congressional District Consideration Summary

Though the congressional makeup was not considered by iRobot in making the SUGV supplier selection decision, as shown in this section, it can be source of risk mitigation. The FCS program is in danger of being reduced from its current makeup, which could include the cancellation of major systems. By selecting contractors in key congressional districts, such as 1) those represented by a senate republicans, and 2) those in states with a high proportion of representation on relevant defense committees, the risk to the SUGV program can be minimized.

4.3 Application of the Location Filter

The location filter was applied to those companies that were not sourced through personal contact. The quantitative results after applying the location filter are presented in Figure

---

[47] Rundquist, Barry S., "Congress and Defense Spending: the Distributive Politics of Defense Spending," University of Oklahoma Press, 2002.

4-3. There are 111 companies still in consideration from sources other than personal contact. 19 companies are still being considered from personal contact. The following section will develop the final filter in the selection process, the financial filter.

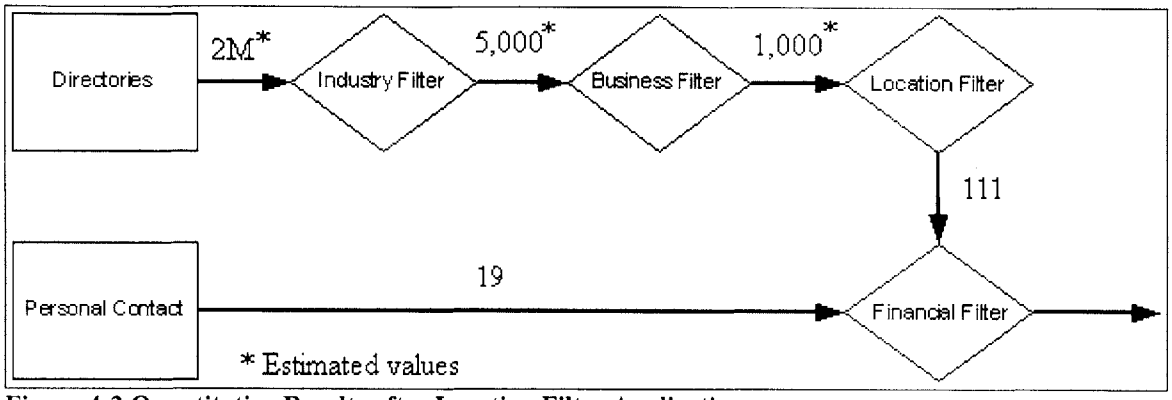


Figure 4-3 Quantitative Results after Location Filter Application

4.4 Financial Filter

To final filter, financial, is developed in this section. This filter ranks the remaining companies according to a financial analysis then eliminates those that are not in the top twenty five. Twenty five was chosen because iRobot determined it was the maximum manageable number of companies to work with on an RFI.

This analysis relies on the public financial statements of the companies remaining for consideration. Private companies were considered on a case by case basis. At this point, iRobot only had a few private companies left for consideration. These were handled on a case by case basis at management's discretion.

Public financial statements were obtained in order to do this analysis. Poring through the 100+ financial statements by hand was not feasible as many companies use a proprietary filing format. For automation purposes, COMPUSTAT on the Wharton Research Data Analysis web site was utilized.[48] The system allowed the simultaneous download of several companies' financials in a standard format. An Excel macro was used to perform

[48] [15]

the financial calculations and sorting. For each company a display was generated from the macro as shown in Figure 4-4.

Ticker: **BA**	Company: **BOEING CO**		
Income Statement	**2005**	**2004**	**2003**
Revenues			
Sales	$ 54,845.00	$ 52,457.00	$ 50,485.00
Cost of Sales	$ 44,757.00	$ 43,550.00	$ 42,868.00
Gross Profit	$ 10,088.00	$ 8,907.00	$ 7,617.00
Gross Margin	18%	17%	15%
Operating Expenses			
SG&A	$ 6,433.00	$ 5,536.00	$ 4,419.00
Depreciation	$ 1,092.00	$ 1,125.00	$ 1,450.00
EBIT	$ 2,563.00	$ 2,246.00	$ 1,748.00
Taxes	$ 257.00	$ 140.00	$ (168.00)
Extra Items	$ (266.00)	$ 234.00	$ 1,198.00
Net Income	**$ 2,572.00**	**$ 1,872.00**	**$ 718.00**
DuPont Analysis +	**2005**	**2004**	**2003**
Assets - Total	$ 60,058.00	$ 53,963.00	$ 53,035.00
Equity	$ 11,059.00	$ 11,286.00	$ 8,139.00
PP&E	$ 8,420.00	$ 8,443.00	$ 13,976.00
Long Term Debt	$ 9,538.00	$ 10,879.00	$ 13,299.00
Interest Expense	$ 713.00	$ 790.00	$ 873.00
Net Profit Margin	4.69%	3.57%	1.42%
Total Asset Turnover	0.91	0.97	0.95
Leverage Factor	5.43	4.78	6.52
ROE	23%	17%	9%
PP&E Turnover	6.51	6.21	3.61
Debt to Equity	86%	96%	163%
Interest Coverage Ratio	3.59	2.84	2.00

Figure 4-4 Example Financial Analysis Summary

The items marked in yellow were used in the ranking, while the remaining items were either static or not considered. Each of the financial ratios will be discussed in the section following.

4.4.1 DuPont Analysis

The filter's ratios comprise the elements of the DuPont formula as well as a few additional ratios that provided insight into company operations. The return on equity for of the DuPont formula is illustrated below. The components of the DuPont formula are briefly introduced below as well.

$$ROE = \frac{NetIncome}{Sales} \bullet \frac{Sales}{Assets} \bullet \frac{Assets}{Equity}$$

Net Income is the total value added to the firm after taxes, depreciation, and all other expenses.

Sales is the total amount in sales that the firm achieved.

Assets is the total fixed and current asset value the firm has ownership of.

Equity is defined as total assets minus total liabilities.

Considering the components of the DuPont analysis independently, the Net Income/Sales ratio is also known as the **Net Profit Margin**. This is a measure of how much money the company keeps of its product's final sale price. It measures how effective the company is on an item sell basis. The Sales/Assets ratio is also known as the total **Asset turnover**, which is a measure of how many times the company turns its assets. The Assets/Equity measure is also known as the **Leverage factor**. It indicates how indebted the company is, the higher the value the more indebted.

4.4.2 Other Analysis

Three additional ratios were used in the financial filter: Fixed Asset Turnover, Debt to Equity, and Interest Coverage ratio. These ratios were included to increase the insight into company operation. They are calculated as indicated below.

$$FixedAssetTurnover = \frac{Sales}{TotalFixedAssets}$$

$$DebtToEquity = \frac{TotalLongTermDebt}{Equity}$$

$$InterestCoverageRatio = \frac{EarningsBeforeInterestAndTaxes}{InterestExpense}$$

The Fixed Asset Turnover ratio gives an indication of the company's use of its fixed assets. These are the assets that are used to produce goods or services and are not

considered inventory, whereas the DuPont Asset turnover ratio considers short term assets, which could include inventory.

The Debt-To-Equity ratio is similar to the ROE DuPont Leverage Factor but considers only the long term debt of the company. The Long Term debt of the company can have impacts on iRobot's decision because it can limit the financial flexibility of the company in create a greater risk for financial distress.

The Interest Coverage Ratio gives an indication as to how well the company is able to make its interest payments. For manufacturing, this is very important in that inventory needs to be financed short term and swings in demand can have a large impact on the interest expense.

4.4.3 Financial Ratio Weighting

A survey was taken of each development and production area on the SUGV to determine the financial factors that were relevant for the manufacturing operations being planned. The poll of iRobot operations personnel resulted in the following weighting factors regarding the financial assessment.

Ratio	Weight
Return on Equity	13%
Net Profit Margin	19%
Asset Turnover	19%
Leverage Factor	13%
PP&E Turnover	9%
Debt-to-Equity	13%
Interest Coverage Ratio	13%

Table 4-2 Financial Ratio Weights

Each company was ranked from best to worst on each ratio. The corresponding weight was applied to the each ranking and the sum was used as a total score. Those companies that were not in the top twenty five were eliminated at this stage.

The results of iRobot's finding and filtering steps are quantified on the edges of Figure 4-5.

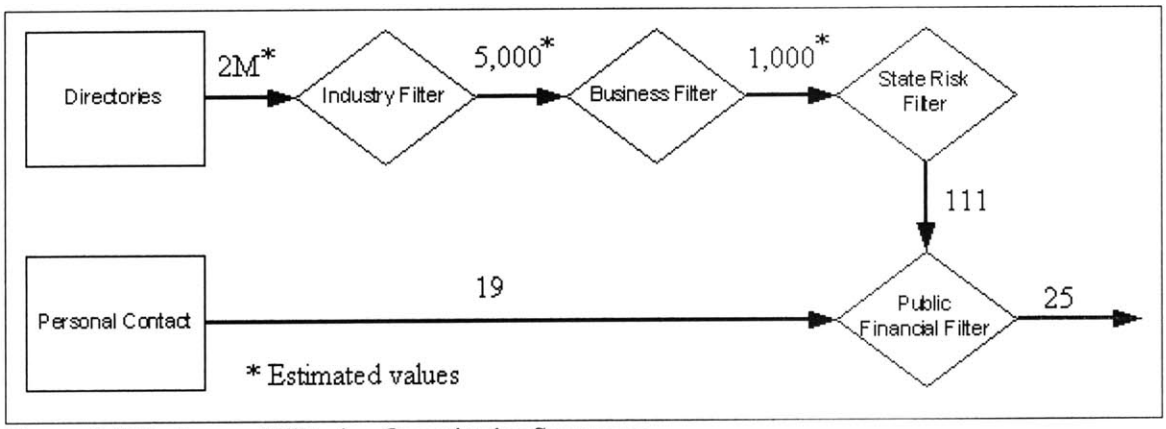


Figure 4-5 Finding and Filtering Quantitative Summary

The 25 companies that remain in consideration were geographically dispersed as illustrated in Figure 4-6. The shading of the states indicates the relative number of companies in each state. The next step in narrowing down the prospective companies is to review the facility requirements of the SUGV program, which will be covered in the following chapter.

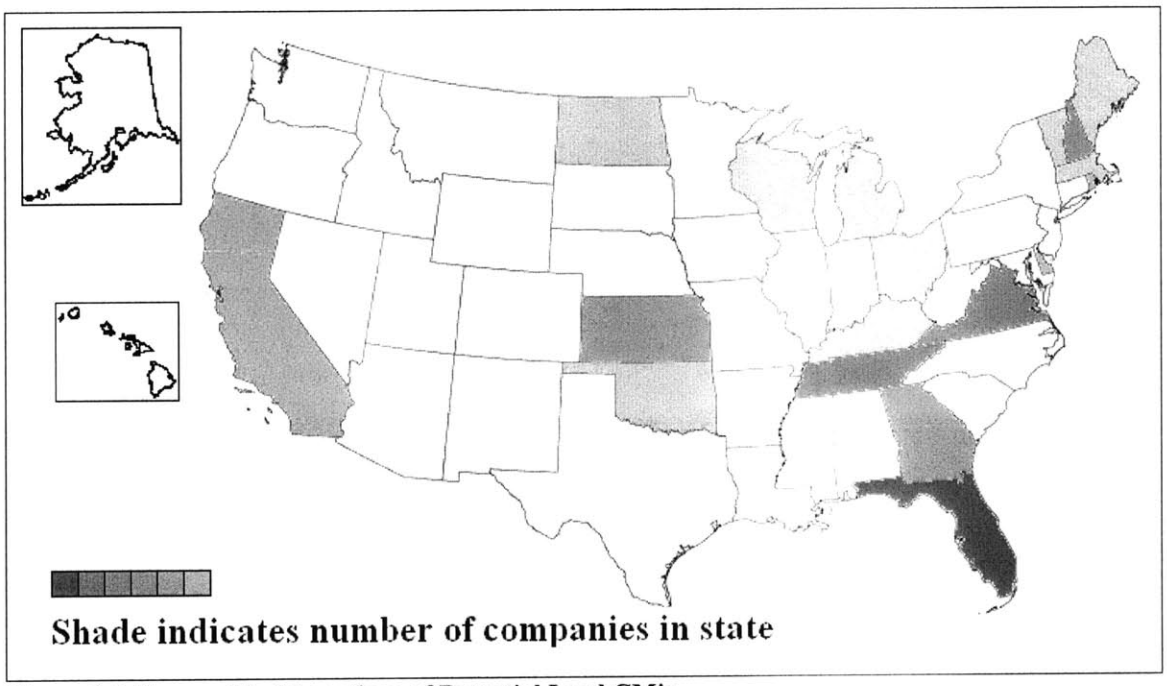


Figure 4-6 Geographical Locations of Potential Lead CM's

4.5 Chapter Summary

This chapter developed the final two filters of the selection process. The location filter was comprised of a state risk assessment, which determined the top 17 states for iRobot

to operate in. A discussion of State public policy was provided to illustrate the implications of iRobot's use of macroeconomic factors. iRobot's use was a real-world scenario of a company considering locating in a state based on these factors and as such should be taken seriously by state governments seeking consideration. The Congressional makeup of the US House and Senate was presented and an argument for its consideration in the location filter was provided. It was determined by iRobot that congressional makeup was not to be considered to ensure the best technical solution. The financial filter was developed using a combination of the DuPont analysis components and three additional ratios.

This page is intentionally left blank

5 Key Skill Identification

This chapter advances the selection process by developing key characteristics for company evaluation. A key characteristics framework is proposed and illustrated using the differences between manufacturing bolts and bullets. The framework is applied to a contract manufacturer of generic circuit boards and that of a SUGV manufacturer. To assist in this comparison activity, an overview of the SUGV's systems is provided. This comparison brings to light those skills that are substantial for the SUGV, and thus provide a second set of criteria for company evaluation.

This chapter also presents the distinguishing characteristics of robotics manufacturing and uses a request for information (RFI) to determine potential contractors' capabilities. A method for sharing detailed product specifications with potential Contract Manufacturers for RFI response is also provided. The proposed method increases the speed with which Contract Manufacturers can grasp the product's interrelationships, and thus increases the accuracy of the response.

5.1 Key Skill Identification Framework

This section illustrates a framework to generate the key skills required in a manufacturing operation. The framework stipulates: 1) performing a baseline analysis of an existing manufacturing operation, 2) performing a baseline analysis of the new operation, and 3) analyzing the differences between these along several criteria. The criteria used in this framework are: raw material inputs, storage, equipment, tooling, maintenance, special skills (processing, testing, handling, etc.), supplier and customer relationships, economies of scale, financing, distribution, and sales. For each criterion, the differences between the manufacturers along cost, responsiveness, and quality are analyzed.

The framework's usage is illustrated in this section using the differences in manufacturing bolts and manufacturing bullets. Bolts and bullets may seem similar enough from a manufacturing standpoint but after applying this framework, the differences stand out.

A baseline for performance has to be established first. This allows the comparison of the bolt manufacturer making bullets against an entrenched bullet manufacturer. The baseline that will be used is that of Cost, Responsiveness, and Quality. Cost indicates the cost for which the bolt manufacturer can produce the bullets, responsiveness indicates the bolt manufacturer's ability to adapt the product to customer needs and fulfill urgent demand, and quality indicates the bolt manufacturer's ability to produce bullets that are comparable to the entrenched manufacturer.

The following areas will be used in the comparison between the manufacturing operations for bolts and bullets: raw material inputs, storage, equipment, tooling, maintenance, special skills (processing, testing, etc.), supplier and customer relationships, economies of scale, financing, distribution, and sales. These criteria were selected based on the following observations:

Raw material inputs. There are significant differences in manufacturing process inputs. Some materials require specialized handling, special purchasing agreements, unique transportation needs, and variable lead times. Quality inspection for incoming material can also be a concern.

Storage. Both raw material and finished goods require storage in some form. Some materials may require special containers, handling, and environments. Storage space can also vary significantly among raw and finished goods materials.

Equipment. Equipment requires special operators, space, facilities, and operational procedures.

Tooling. Tooling requires monitoring and handling skills that are unique to the tool type. Specialized labor is required to maintain tool effectiveness.

Maintenance. Specialized skills are required to maintain the tooling, equipment, facilities, and inventory.

Special Skills. Some manufacturing processes require special skills such as testing, debugging, and processing.

Supplier and Customer Relationships. Supplier and customer relationships can be significantly different between manufacturing processes. Commodity purchasing typically requires shopping a bid around while custom part ordering requires specialized quality inspections and change requests among other things.

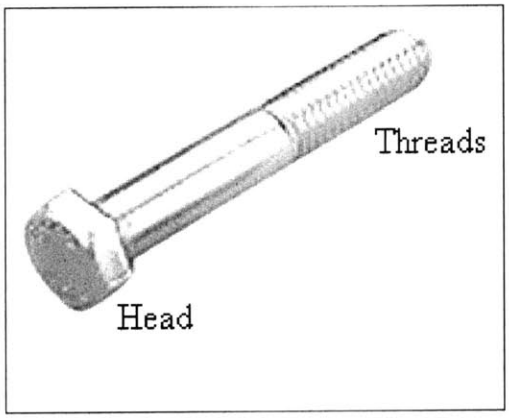
Economies of Scale. Economies of scale can be significantly different between manufacturers. These economies depend on the other types of business the manufacturers are involved with.

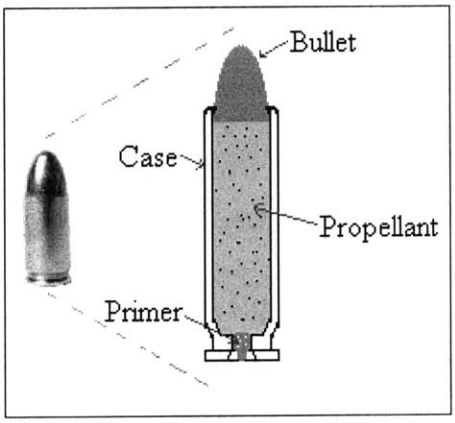
Financing. The way the manufacturer finances the inventory can influence its inventory policies. Some manufacturers may also need loans to perform the required work.

Distribution. Distribution channels can vary widely depending on the items being sold. Some channels require stocking shelves and others filling warehouses.

Sales. The sales force may be incapable of communicating the product's specifications.

The illustration of the framework using the manufacturing of bolt and bullets follows. It begins by describing the manufacturing processes for each, then utilizes the framework to distinguish the differences.





A bolt is made in two parts, a threaded cylinder and a head. First the head is made by taking a steal cylinder slug and pressing in a process called heading. Second, the threads are made by taking the cylinder with the head and rolling it against two threaded dies in a

hydraulic press, a process called thread rolling. Once the head and threads have been formed, the part is complete.

Bullets, here after called cartridges, are made up of four components: a projectile (also know as the bullet), a case, propellant, and primer. Each of these components is manufactured using a different process. The projectile on a .22 caliber round is typically made from a thick lead wire using a press (also known as swaging). The case is made using a piece of brass sheet metal using drawing and punching in a hydraulic press. The propellant is typically made of smokeless powder. And the primer is typically made out of two pieces of metal, a cup and anvil, and an explosive material, typically lead styphnate. The cup and anvil are made of brass and formed through swaging.

Bullet assembly is completed through a series of steps. The primer is assembled first by filling the primer cup with lead styphnate then lightly pressing the anvil into place. Next, the primer is pressed into the case. The propellant is added to the case above the primer. And lastly, the bullet is placed (seeded) atop the propellant and crimped into the case[49].

The baseline criterion for each manufacturing operation is presented in Table 5-1.

Criteria	Bolt	Bullet
Inputs	Cylindrical steal slugs	Brass sheet metal Smokeless powder Lead wire Lead styphnate
Equipment	Hydraulic Press	Hydraulic press
Tooling	Heading dies Threading dies	Case drawing dies Anvil dies Cup dies Projectile dies Primer dies Primer-case seed die Cartridge die

[49] [8] [9] [10] [11]

Criteria	Bolt	Bullet
Storage	Slugs Headed slugs Completed slugs	Lead styphnate Smokeless powder Brass sheet metal Brass coins Anvils Cups Primers Lead wire Projectiles Cases Cases with primer Cartridges
Maintenance	Monitor die wear	Monitor die wear
Processing	Heading Thread rolling	Drawing Swaging Crimping Pressing
Handling		Lead styphnate Smokeless powder Primer Cartridges
Testing	Statistical Process Control (SPC) Dimensional Strength	SPC Dimensional Strength Accuracy Velocity Burn
Test tooling	Harnesses, Stress tester	Harnesses Electronic Primer ignition Velocity monitor Accuracy monitor
Relationships	Single supplier type Multiple customer types	Multiple supplier types Multiple customer types
Economies of scale	Possible	Possible
Inventory Policy	Build to stock	Build to stock
Financing	Self financed	Self financed
Distribution	Hardware stores Manufacturers Contractors	Hunting stores Military
Sales force	Specialized	Specialized

Table 5-1 Bolt vs. Bullet Skill Identification

The tabulation above provides the comparison across individual criterion. The next step is to determine how theses differences influence the performance baseline of: Cost, Responsiveness, and Quality. Assuming that the bolt manufacturer is considering moving into bullet manufacturing because of a market opportunity, the question arises: given these differences in operation can the bolt manufacturer perform competitively with an entrenched bullet manufacturer?

Table 5-2 compares the bolt manufacturer making bullets against the entrenched bullet manufacturer along the baseline performance measures of Cost, Responsiveness, and Quality. For example, the first row, inputs, identifies for each of cost, responsiveness, and quality, the bolt manufacturer's disadvantage in competing with the entrenched bullet manufacturer. In this example, the bolt manufacturer would have a higher cost due to the complexity of working with several raw materials when they have been accustomed to using a single raw material.

Criteria	Cost	Responsiveness	Quality
Inputs	Learning curve in dealing with multiple sources of raw material and precision materials	Supply base is disadvantaged due to the new relationship	Procurement will need to learn performance specifications of raw materials
Equipment	Minimal concern	Minimal concern	Minimal concern
Tooling	Faster tool wear due to inexperience	Complex dies make changes difficult for entrant	Operating complex dies decreases quality
Storage	Additional intermediate storage items require additional inventory control	Minimal concern	Minimal concern
Maintenance	Minimal concern	Minimal concern	Minimal concern
Processing	Significant learning curve	Slow to make changes	Poor quality due to process expertise
Handling	Special handling procedure development	Lack of experience limits on-the-spot manipulation	Decreased due to handling errors
Testing	Lack of expertise requires addition time	Changes difficult due to learning curve	Minimal concern
Test tooling	Faster tool wear due to inexperience	Changes difficult due to learning curve	Minimal concern
Relationships	New relationships in general are costly. Adapting to multiple amplifies the cost	Responsiveness in relationships takes time to build	Minimal concern
Economies of scale	Poor economies at first, leads to high cost	Fewer investments make for a nimble position	Minimal concern
Inventory Policy	Minimal concern	Minimal concern	Minimal concern
Financing	Minimal concern	Minimal concern	Minimal concern
Distribution	Takes time to build distribution	Entrant could be more responsive due to low investment	Minimal concern
Sales force	Expertise based sales will take time	Poor adaptability to changes	Minimal concern

Table 5-2 Bolt to Bullet Cost, Responsiveness, and Quality

It can be seen from the tabulation that though the bolt manufacturer could change operations to produce bullets, it would be at a significant disadvantage if competing against the entrenched bullet manufacturer. This has a few implications for selecting a CM to perform services for iRobot. It can not be assumed that the CM can hire the expertise, buy the tooling and equipment, and perform on the contract with equal performance. The manufacturer will work through the learning curve but may continue at a disadvantage due to economies of scale.

This framework will now be applied to the SUGV using a circuit board contract manufacturer and the target environment for a SUGV Lead CM. The circuit board manufacturer was selected because several of the companies remaining for consideration have experience manufacturing circuit boards. To begin the process, a discussion on the robotics industry is presented. Those characteristics that separate robotics manufacturing from run-of-the-mill manufacturing are called out here.

5.2 Robotics Industry

This section provides a brief overview of the robotics industry. The overview discusses the current state of the industry in the life cycle model framework developed by Christensen[50] and Fine[51]. The distinguishing characteristics of robotics development are discussed with an example to illustrate. The level of electro-mechanical integration found in the industry is discussed and the differences between modular and integral design are introduced.

Christensen and Fine argue that all industries are cyclical. At some point in the evolution of an industry the product technologies overshoot customers' ability to utilize product enhancements, at which point customers become cost conscious and companies feel pressure to standardize in order to reduce costs. As standardization occurs, more players enter the market, thus driving the product to commodity pricing. As technological

---

[50] Christensen, Clayton. Raynor, Michael, "The Innovator's Solution: Creating and Sustaining Successful Growth." Harvard Business Press, 2003

[51] Fine, Charles H., "Clock Speed; Winning Industry Control in the Age of Temporary Advantage." Perseus Books, 1998.

advances occur that allow customers to utilize better performance, companies are pressured to integrate to streamline product performance along the new performance areas.

The robotics industry today is in the Integral Product, Vertical Industry, and Proprietary Standards part of its life cycle. This is based on 3 valuable observations: 1) the performance characteristics of robots are far below that of what a consumer could utilize, 2) very few standards stipulate the communication and connections among components that make up a robot, and 3) most robotics manufacturers must develop the product from the ground up to meet the product objectives.

To illustrate these observations, assume iRobot wants to develop a robot that would fetch a newspaper out of a yard every morning. This product objective implies that this robot is going to need five major components: 1) a way of getting around (moving), 2) a way of determining where it is (seeing), 3) a way of determining when to expect the paper (timing), 4) a method of grasping the paper (manipulator), and 5) a form of power.

iRobot's options for designing such a robot are limited. The probability of iRobot's being able to select existing commercial parts and integrate them to perform the objective is negligible. The variation in the product's environment creates a dynamic interplay between the subsystems, subsystems that can not handle the myriad of input variations they are subjected to. This interplay is illustrated further in the following analysis.

The first of the five major components of this robot is the mode of transportation. Based on the robot's objective to fetch a newspaper from a yard, the transportation component must be capable of handling the following environments: 1) in-house conditions such as carpeted, hardwood, and tiled flooring, 2) porch and stair conditions, and 3) yard conditions such as grass, dirt, cement, and cobblestone. The variations in terrain are immense and there is most likely no commercially available component that can be utilized to navigate these variations.

The second major component is the sight for the robot. The robot needs to be able to see where it is going to avoid obstacles and to detect the newspaper. A camera could provide

the sight, but the interpretation of images generated by the camera would require sophisticated algorithms for distinguishing obstacles, terrain, and the newspaper. These sophisticated algorithms imply the need for a powerful image processor, which implies a significant power source. The significant power source implies heat dissipation issues and additional power to move the additional weight around. Again, iRobot would need to develop this system with these implications in mind.

The consequences of the product's objective on the major subsystems must be considered simultaneously to ensure the system as a whole meets the product objectives. Components do not exist on the market today that are versatile enough to meet all of iRobot's needs in this case. iRobot will need to develop the robot from the ground up utilizing commercial parts wherever possible and custom designing all the remainder.

If this were a modular industry iRobot could in theory take the objective to fetch the newspaper then select the components from suppliers and perform the integration. But, in today's environment, there are too many physical problems to overcome to design a robotic system modularly. There may be a few exceptions for well known applications such as manufacturing, but in general the robotics industry is vertically integrated.

5.2.1 Electro-Mechanical Integration

This section briefly illustrates the level of electro-mechanical integration found in the industry. This level of integration is unique to robotics and has a few repercussions, which are discussed.

The robotics industry is unique in its level of electro-mechanical integration. The semi-conductor equipment industry is close, but it does not generally have the same size and power constraints. A typical robot is made up of sensors, actuators, circuits, power, and structural elements. Putting these different components together creates a dynamic system with a large degree of internal interaction. Often the system is too complex to determine the exact behavior before building and testing it. Thus prototyping is a key aspect of robotics development.

Returning to the fetching robot example, assume that iRobot is now testing the robot to see how well it performs its job. When the robot encounters the stairs it begins its stair climb operation. This operation draws a significant amount of power from the batteries. Simultaneously, the actuator holding the newspaper is being clamped to maintain a hold. When the climb begins the power drops just enough to cause the manipulator to release the paper. Independently, the robot could have performed both operations, but when combined in this dynamic system, adverse effects can occur.

5.2.2 Modular vs. Integral Development

This section illustrates the differences between modular and integral parts. iRobot utilizes modular components whenever possible, but is forced, at times, to design and use integral components to meet product objectives.

In a modular world, components are forgiving at the boundary of their specified limits. Components are interchangeable when standards have been set. Standards set a range of inputs for which a predicable output will result. When modular equipment manufacturers create their products it is assumed that there will be a wide range of inputs that the product will be subjected to, often times, the specification will call out the range. When these manufacturers design and build these products, protections and buffers are incorporated to allow for this input variation. Standard interfaces also create an environment where innovations occur that allow these modular components to operate even when in conditions outside of the specification range.

As a result of the characteristics of this industry, iRobot must design a significant portion of each subsystem in each robot. The following section takes these conditions into account when developing the key characteristics of the SUGV manufacturer.

5.3 SUGV Key Characteristic Generation

This section develops the differences between the existing contract manufacturer producing electronic boards and the contract manufacturer performing the final integration of the SUGV. This process is completed in three steps: 1) discussing the

mock SUGV's subsystems, 2) discussing the manufacturing flow of the mock SUGV, and 3) utilizing the key characteristics framework.

5.3.1 SUGV System Analysis

Based on the mock requirements presented in section 3.1 the following logical and physical subsystems could result. These subsystems are fictional but are representative of the major components on the SUGV and will be used to produce the key manufacturing characteristics.

Communications. The communications subsystem provides communication among the myriad of electronic components on the SUGV. The electronic components attached to the communication system are: video, radio, actuators, sensor network, main computer, and the battery. Communications are handled through a proprietary interface based on a single pair of wires. This communications protocol is capable of sending data in 512 byte packets at a rate of 100 Mbits. Each packet has a 4 byte header identifying the target device for the packet. The communications controller handles bus arbitration according to the proprietary protocol. Each device attached to the communications bus contains a communications front end that ignores packets destined for other devices.

Power Management. The power management subsystem monitors battery performance and subsystem draw. This system is a virtual subsystem handled in the software of the main computer.

Each electronic component that is attached to the communications bus has a power save mode, which is utilized when the main computer does not need the device. The power management system has access to the communications bus through the main computer and uses a Sleep and Awake packet to control the electronic components' power state.

Each electronic component is supplied with power on a dual bus. The bus is comprised of two sets of wires providing 5 and 12 volts of direct current. The 5 and 12 volt lines power the majority of electronic circuits and the actuators respectively. The power management system does not have the capability to eliminate power to any particular component.

Sensors. The mock SUGV has two types of sensors: direct and indirect. The direct sensors are connected directly to the main computer through single or paired wires. The main computer utilizes a multiplexer to get the sensor signals into the CPU. There are two types of direct sensors, electrical (voltage and current), and temperature. These two types of sensors use analog signals to communicate status.

The indirect sensors are attached directly to the communications bus and send digital updates. These sensors are more complex than their direct sensor counterparts. They encode the same information but the indirect sensors require digitization circuitry between the analog sensor and the communications bus. Sensor status is provided upon the receipt of a status update packet.

Actuators. The SUGV is comprised of six actuators, two to drive the main tracks, one to drive the flipper position, and three to drive the neck. Each actuator is sized according to the torque and speed requirements for its use. Each actuator has a motor driver board, encoder, communications bus connection, heat sensor, voltage sensor, and current sensor. The main computer controls the motor by specifying a velocity, acceleration, or current to the component over the communications bus. The motor driver board interprets the commands and sensor feedback to supply the appropriate signals to the actuator. All sensor data is provided back to the main computer over the communications bus as requested.

Video. The video subsystem is comprised of two cameras (visible and infrared), zoom lenses and position encoders, two image processors, and a communications bus connection. Each camera lens and each image processor are independently controlled through the communications bus. Low bandwidth image data is transferred through the communications bus while high bandwidth image data goes directly to the radio. The video subsystem is contained in the head of the SUGV.

Radio. The radio is located in the head alongside the antenna. The radio is capable if transferring data at a rate of 100Mbits. A communications bus connection allows the main computer to transmit and receive data to the remote controller. The direct video link to the camera components allows high speed video to be transmitted.

Neck. The neck is a bottleneck for signals as it has little room for wiring. The communications bus is used to communicate with all head and neck components. This keeps the number of wires passing through the neck to six, communications and power.

The majority of space in the neck is taken by the three actuators (pan, tilt, and elevation). As a result of the size constraint in the neck, each of the actuators requires a 0.5mm Ball Grid Array FPGA to communicate with the bus. This BGA creates problems in manufacturing as alignment and inspection are difficult. At 0.5mm spacing between solder balls, the slightest imperfection in pick-and-place equipment or adhesion during reflow can cause shorts and opens. In addition, because the BGA is a blind connection, X-ray verification must be used to ensure proper connections.

Main computer. The main computer resides in the chassis portion of the mock SUGV. It contains the communications bus controller, sensor interpretation circuitry, a 1 GHz microprocessor, 5 Mb of flash memory, 256MB of random access memory, and power connections. The main computer has a Linux operating system with the iRobot AWARE software, which allows for the behavioral control of the robot and provides standard software interfaces to sensor data and control signals.

5.3.2 SUGV Key Manufacturing Processes

This section discusses the key manufacturing processes and logistics planned for the SUGV. These processes are presented here to provide the necessary information to complete the key characteristics framework.

Circuit boards. Circuit boards are created on a Surface Mount Technology (SMT) line with an optional thru-hole operation as required by board design. Standard optical and X-ray inspections will be performed to ensure consistent quality. Test harnesses are provided by iRobot for functional compliance testing prior to subassembly integration.

Circuit boards are shipped to electronic subassembly providers. The electronic subassemblies are: 1) radio and antenna, 2) actuators, 3) battery, 4) video, and 5) main computer. Each subassembly provider integrates the electronic boards with mechanical

elements. iRobot provides each electronic subassembly provider with test harnesses to ensure subassembly functional compliance.

Mechanical elements. Mechanical elements are manufactured at a variety of facilities through numerous manufacturing processes including: injection molding, milling, casting, and brazing. These elements are shipped to either the final integration facility or to subassembly providers.

Final Integration. Final integration takes place at the Lead Contract Manufacturer's facilities. Both electronic and mechanical subassemblies are integrated by the lead CM. Testing points in the integration process are established by iRobot. iRobot also provides the test harnesses and procedures for integration.

5.3.3 Risks

There are three main risks associated with the manufacturing of these parts.

0.5mm BGA. The 0.5mm BGA in the neck actuators are difficult to process consistently with existing technology. Small variations in Pick and Place equipment can cause alignment problems due to the small connection spacing. In addition, the adhesion of the BGA to the surface must be consistent through reflow to ensure settling does not cause misalignment.

Ownership. With iRobot defining the assembly procedures and providing the test harnesses, subassembly providers may be resistant to taking ownership for failures. This can delay rework and root cause analysis.

Through Hole and Wave Soldering. Some electronic boards have both SMT and thru-hole technology. When SMT parts are subjected to wave soldering operations there is a potential for reflow to occur, which can upset existing connections.

5.3.4 SUGV Comparison

This section compares the electronic contract manufacturer producing electron boards to a contract manufacturer performing the final integration of the SUGV. The bolt vs. bullet framework is used to determine the difference in operations.

Table 5-3 provides the characteristics of the electronic board and SUGV manufacturer's operations. This table is used to perform the next step in the framework, determining the cost, responsiveness, and quality implications of making the changeover to produce SUGVs.

Criteria	Electronics Contract Manufacturer	SUGV Final Integration
Inputs	Active and passive electronic components, Printed circuit boards (PCB), SMT and thru-hole materials.	Mechanical components, mechanical sub-assemblies, electro-mechanical sub-assemblies.
Equipment	SMT line, Wave soldering	None
Tooling	Solder paste screens	Integration harnesses
Storage	PCBs, Electronic boards movable in quantity.	Several parts both small and large. Fixed storage space.
Maintenance	In house technicians, equipment OEM technicians	Minimal
Processing	Streamlined SMT line, little labor content	Unique procedural
Handling	Trays	Racks
Testing	Streamlined X-Ray, Optical, visual	iRobot mandated procedures
Test tooling	X-Ray, Optical Inspection	iRobot supplied test harness.
Relationships	Hands off with suppliers, account manager for customer	iRobot integrated, supplier integrated
Economies of scale	Large	None
Inventory Policy	Build to order	Build to order
Financing	Various	Self financed, iRobot loan possible.
Distribution	Direct	Direct
Sales force	Account based	Account based

Table 5-3 SUGV and ECM Manufacturing Baseline

The Contract Manufacturer making electronic boards has a uniform operations practice across each of its products. The manufacturing organization performs the same operations regardless of the product being produced. Inputs, such as PCBs, and tooling, such as solder screens, are switched out but the operation and manufacturing flow is ubiquitous.

The SUGV manufacturer requires unique processing and storage procedures, a few of which will be discussed. Input parts are specialized, which reduces economics of scale. Supplier management is required to ensure quality, timing, and cost as a commodities market does not exist for these parts. Testing procedures are unique to the SUGV, and the level of human interaction in the integration efforts are far greater than those required for board manufacturing.

Table 5-4 provides a comparison across cost, responsiveness, and quality for the electronic board manufacturer to manufacturer the SUGV.

Criteria	Cost	Responsiveness	Quality
Inputs	Learning curve associated with new types of inputs	Specialized assets have little reuse potential	Mechanical component management needs experience
Equipment	Minimal concern	Minimal concern	Minimal concern
Tooling	Faster tool wear due to inexperience	iRobot's ownership restricts responsiveness	Learning curve
Storage	Fixed storage space, may require capital expense	Minimal concern	Minimal concern
Maintenance	Minimal concern	iRobot owned limits	Minimal concern
Processing	High cost, out of normal operations	Low until learning curve takes effect	Low until learning curve takes effect
Handling	Minimal concern	Minimal concern	Minimal concern
Testing	High cost, out of normal operations	Electro-mechanical testing requires specialized labor	Electro-mechanical testing requires specialized labor
Test tooling	Minimal concern	Requires iRobot changes	Minimal concern
Relationships	Higher due to supplier monitoring	Learning curve effects	Learning curve effects in supplier management
Economies of scale	Minimal concern	Minimal concern	Minimal concern
Inventory Policy	Minimal concern	Minimal concern	Minimal concern
Financing	Minimal concern	Minimal concern	Minimal concern
Distribution	Minimal concern	Minimal concern	Minimal concern
Sales force	Minimal concern	Minimal concern	Minimal concern

Table 5-4 ECM producing SUGV Cost, Responsiveness, and Quality

The significant characteristics brought to light in Table 5-4 are: 1) handling odd parts, 2) iRobot's ownership of tooling, fixtures, and procedures, 3) commodity purchasing to supplier management, and 4) Electro-mechanical testing. The implications of each of these characteristics will be discussed briefly.

Handling and Storage Space. Existing contract manufacturers store common parts such as resistors, capacitors, chips, and PCBs. Operations at these facilities are streamlined based on the mobility and flexibility of these components. The SUGV has several parts that are not standard size and are difficult to transport within the facility. The SUGV requires a large storage area, which may stipulate expansion at the manufacturer's facilities.

iRobot's ownership of tooling, fixtures, and procedures. iRobot's owning the tooling, fixtures, and procedures limits the flexibility of the contract manufacturer to troubleshoot and streamline operations. This requires additional levels of communication between the parties to troubleshoot manufacturing problems. The additional communication and reduced responsiveness can lead to higher overall costs for iRobot.

Commodity purchasing to supplier management. Contract manufacturers producing electronic boards typically buy parts on a commodity market. The CM's purchasing agents are rewarded based on prices attained. Moving to a SUGV manufacturing operation requires purchasing specialized parts from a select few suppliers. The CM must establish procedures for managing these suppliers and monitoring the quality of the parts received. This can lead to higher costs and reduced quality as purchasing agents come up to speed.

Electro-mechanical testing. The level of skill required to test and troubleshoot electro-mechanical devices is significantly different from the testing and troubleshooting of circuit boards. Circuit boards are tested using automated techniques such as X-ray and Automated Optical Inspection, whereas the SUGV test procedures call out labor intensive procedures.

These characteristics were considered in the final evaluation criteria to ensure the best possible outcome. A request for information (RFI) is used to draw out the individual characteristics of the potential suppliers. Included in this RFI are elements that indicate willingness and capability to execute on these key characteristics.

5.4 RFI

The RFI provides a non-binding mechanism for sharing information regarding the product and soliciting CMs for their capabilities. iRobot has determined that the RFI should go out to no more than 25 CMs. In issuing an RFI, the iRobot expects that the CMs will invest time to perform the following activities: 1) research the product, 2) develop manufacturing requirements, 3) review internal operations, 4) develop a plan for product introduction, and 5) develop an RFI response.

5.4.1 BOM Sharing and Quick analysis

Once the RFI has been sent to the potential CMs it is in iRobot's best interest to assist those CMs in evaluating the product and manufacturing requirements. The typical mode of assisting is to send documents that outline the products characteristics. This is a time consuming process and often leads to several requests for additional information from the CMs. A difficult part in understanding the product is interpreting the relationships among product parts and associated documentation.

A software tool was developed as a result of the internship research that assists CMs in gaining the relational understanding of the documentation and product parts. This tool allows the CM to understand the documentation and part relationships quicker and as a result can increase the accuracy of the RFI response. Figure 5-1 illustrates the output from this tool.

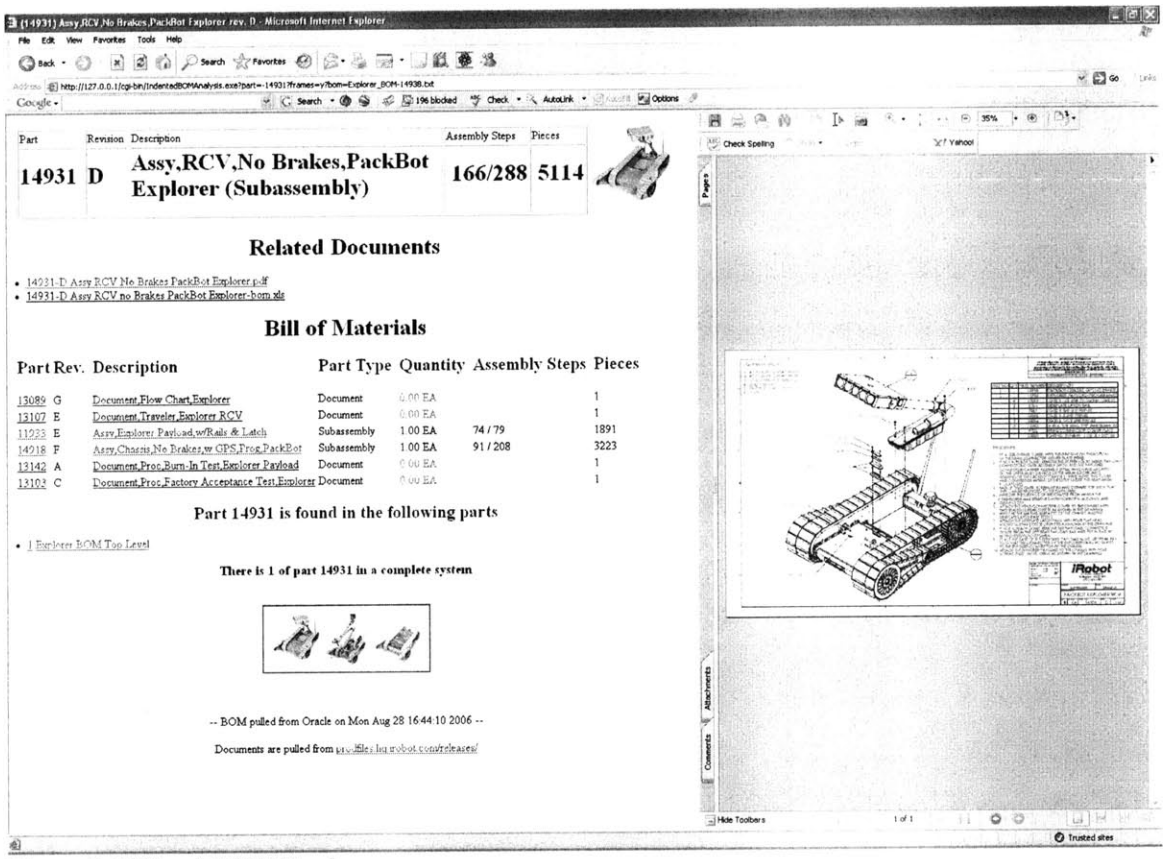


Figure 5-1 Visual BOM Explorer

This application allows the CM to explore the subcomponent parts of the SUGV on-line by clicking on the links on the left, which are tied to the Bill of Material. The drawing of the part appears on the right and additional documentation available for the part are presented as links on the left. Other parts that the current part under review is found in are also identified as links on the left.

The source code for this tool is included in Appendix C -BOM Relational Exploration Source Code.

5.5 Chapter Summary

This chapter developed the key characteristics for company evaluation using a key characteristics framework. The framework was applied to a contract manufacturer of generic circuit boards and that of a SUGV manufacturer. An overview of the SUGV's systems was provided to assist in the key characteristics generation.

An RFI was used as the mode to solicit feedback from the remaining CMs under consideration. This chapter also presented a method for sharing detailed product specifications with potential contract manufacturers to improve the speed and quality of responses.

The next chapter completes the selection processes by presenting a method for modeling the manufacture of the SUGV using the CMs' responses and the planned supply chain.

This page is intentionally left blank

6 Manufacturing Environment Simulation

This chapter provides a simulation model to evaluate the suppliers that remain for consideration. The research conducted at iRobot did not include this modeling exercise. At the time of this writing, iRobot was awaiting RFI responses. The primary insights that can be gained through use of this model are: 1) optimal inventory levels, 2) part pricing, 3) on-time delivery expectations, and 4) disruption effects. By simulating the manufacturing environment with each of the potential suppliers, objective comparisons can be made along these 4 criteria.

A mock manufacturing environment was used here as an illustration of the model's capabilities. Because the actual data was not available, input values were generated qualitatively for this exercise. The results of the mock manufacturing environment are discussed to illustrate the insight that can be gained through its use. In addition to simulating the manufacturing environment, this model can be used to simulate supply disruptions and supplier change. A sensitivity analysis using the mock environment is presented based on a shipping route disruption.

When iRobot is ready to perform this modeling exercise, the responses from the RFI presented in section 5.4 should be used to the extent possible in determining the model inputs. Qualitative analysis can be used to generate the input values, but the more accurate the input values the better the predictive capabilities of the model.

After the modeling exercise is completed by iRobot, the final selection of the lead contract manufacturer will proceed. It is beyond the scope of this thesis to go through the final steps, but it is anticipated that iRobot will use a credit report, historic and pending litigation, and numerous references in making its final decision.

This chapter presents the model by: 1) introducing the order cycle and supplier interactions, 2) describing the simulated parts and their relationships, 3) forecasting part volume, 4) describing the model, 5) presenting the model parameters for the mock environment, 6) discussing the model results, and 7) presenting a supply disruption case.

These sections describe several parts of the SUGV supply chain. The scenarios discussed here represent the plan for the SUGV as of this writing, but is not expected to match the actual implementation; the SUGV plan is still being developed and as such modifications are expected.

6.1 Order Cycle and Supplier Interaction

iRobot will give the lead CM control over the supplier base. In some instances, however, iRobot will direct sourcing to suppliers that are deemed strategic. The lead CM is expected to provide assembly, test, and fulfillment services. The traditional manufacturing processes of transforming material will be left to the lead CM's supply base.

The order cycle will be monthly; orders will be received at the beginning of the month and product will be shipped by the end of the month. Parts that do not conform to iRobot's specifications or fail within the warranty period will be replaced as soon as possible. These replacements can be ordered at any point within the order cycle, iRobot does not have to wait until the next order cycle to put in replacement part orders.

6.2 Subassembly Identification

Because iRobot wants the lead integrator to focus on the assembly, test, and fulfillment of the final product only subassemblies will be supplied to the lead integrator. For this mock simulation, the following subassemblies have been identified for the SUGV:

Name	Description	Integrates with
SUGV	Final SUGV Product	Top Level
Chassis	Completed chassis without flippers.	SUGV
Flippers	Flippers for stair handling and flip control.	SUGV
Head and Neck	Head and Neck component integrated.	SUGV
Battery	Power source for SUGV	SUGV
Chassis Track	Rubber track	Chassis
Chassis Side Panels	Side panels chassis. Provides structure connections.	Chassis
Electronics Box	Chassis electronics	Chassis
Flipper Hubs	Hubs for the flipper	Flipper
Flipper Track	Rubber track for the flipper	Flipper
Flipper Guide	Guides the track for the flipper	Flipper
Head	Head component	Head and Neck
Neck	Neck component	Head and Neck
Electronics Package	All electronic boards for SUGV processing.	Electronics Box
Electronics Housing	Houses the electronics package	Electronics Box
Head Lid	Top portion of the Head, contains cameras and processors.	Head
Head Base	Bottom portion of the Head, contains the interconnect circuits and housing.	Head
Neck Tube	Structural element of the neck	Neck
Neck Actuators	Motor driver and motors for pan and tilt	Neck

Table 6-1 Mock SUGV Subassembly Relationships

Figure 6-1 has been provided to help illustrate the relationships defined above. Horizontal alignment indicates the level of subassembly with the solid bars indicating the constituent parts for the subassembly. This pictorial shows the subassemblies that are up to three levels deep. This mock SUGV does not include subassemblies deeper than this.

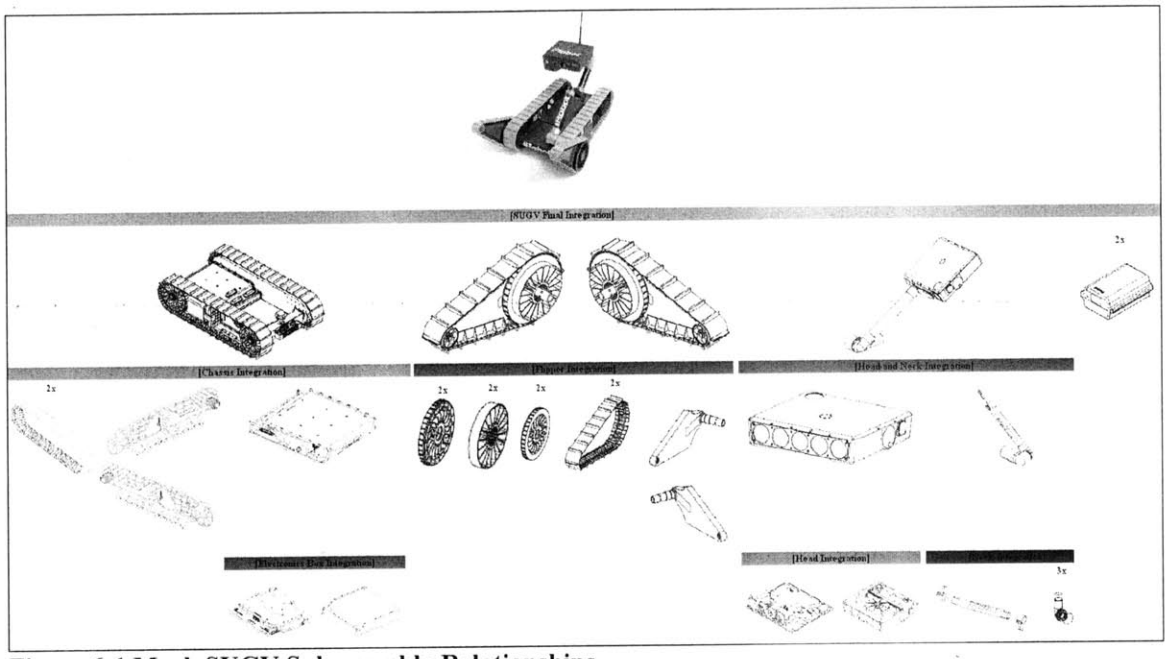


Figure 6-1 Mock SUGV Subassembly Relationships

These subassemblies were chosen for this mock SUGV because they represent the LRUs discussed in section 3.1.2 and they constitute all the final elements of SUGV integration.

The next section generates the demand volume for each of these subassemblies based on new issues to military units and maintenance requirements.

6.3 Planned Volume and forecast identification exploded

This section explodes the SUGV demand for initial fielding and maintenance requests. This data will be used in the model to simulate part demand and ramp. In order to complete this exercise a few assumptions are made: 1) the army will systematically equip its field units with SUGVs at a constant monthly rate over the years 2008 to 2017, 2) 7,666 SUGVs are required to completely equip the army, 3) replacement part demand is determined solely by the number of SUGVs in the field, and 4) if a part fails in the field it is discarded, including any constituent parts.

For each subassembly, the infield failure rate is estimated in Table 6-2. Rates are in units expected to fail each year for a single fielded SUGV. The "Quantity" column identifies the number of the specific subassembly that comprises a complete SUGV. The "Independent replacement" column identifies the number of that type of subassembly that

will fail independent of other failures on the SUGV; it is calculated by multiplying the subassembly failure rate by the quantity per SUGV.

Subassembly	Failure Rate (per year)	Quantity (per SUGV)	Independent replacement (per year)	Cumulative Replacement (per year)
SUGV	0.05	1	0.05	0.05
Chassis	0.05	1	0.05	0.1
Flippers	0.2	2	0.4	0.5
Head and Neck	0.1	1	0.1	0.15
Battery	1	2	2	2.1
Chassis Track	1.5	2	3	3.2
Chassis Side Panels	0.1	1	0.1	0.2
Electronics Box	0.1	1	0.1	0.2
Flipper Hubs	0.3	2	0.6	1.6
Flipper Track	1.3	2	2.6	3.6
Flipper Guide	0.2	2	0.4	1.4
Head	0.15	1	0.15	0.3
Neck	0.15	1	0.15	0.3
Electronics Package	0.17	1	0.17	0.37
Electronics Housing	0.05	1	0.05	0.25
Head Lid	0.15	1	0.15	0.45
Head Base	0.15	1	0.15	0.45
Neck Tube	0.15	1	0.15	0.45
Neck Actuators	0.06	3	0.18	1.08

Table 6-2 Subassembly Failure Rates

The "Cumulative replacement" column indicates the actual demand for subassemblies. This column adjusts the replacement rate to account for assumption 4 (parts that fail in field are discarded, including constituent parts). The "Cumulative replacement" column accounts for higher level subassembly failure in addition to the independent failure rate. For example, a Neck Actuator needs replacing on average 0.06 times a year. Given that there are three of these in a SUGV, on average, 0.18 actuators will need to be produced each year to replace the failed ones (this value is captured in the "Independent Replacement" column). If however, the entire Neck subassembly needs to be replaced, which occurs 0.15 times a year, then 3 Neck Actuators will be manufactured because there are 3 Neck Actuators in each Neck.

With the military fielding rate and the subassembly failure rates, a demand profile can be generated. This profile is presented Table 6-3 and illustrated in Figure 6-2. Each period

is defined as a single month. The military fielding period is120 months or 10 years and ongoing maintenance occurs at a constant rate after 120 months to maintain military readiness levels.

The "1st Period Demand" column is the number of subassemblies demanded to meet the military's fielding goals. The "Last Period Demand" column is the number of subassemblies that will be demanded in the last period. It is calculated by taking the fielding rate and adding the replacement demand, where the replacement demand is a function of the number of units in the field. The calculation is shown here:

$$PeriodDemand(t) = MonthyRate + CummulativeReplacementRate \bullet \int_{1}^{t} MonthlyRate \cdot \partial t$$

The "Ramp Rate" column is increase in demand each month due to increasing failures. It is calculated by taking the derivative of the PeriodDemand function shown above. The "Maintenance Period Demand" is the steady state demand, which occurs after the army has been completely outfitted with SUGVs. And the "Equipping Period Total Demand" is the number of units supplied during the 10 year outfitting period. It is calculated by taking the integral of the PeriodDemand function over the 120 month outfitting period.

Subassembly	1st Period Demand t = 1	Last Period Demand t = 120	Ramp Rate 0 < t ≤ 10 years (increase per month)	Maintenance Period Demand t > 10 years	Equipping Period Total Demand 0 < t ≤ 10 years
SUGV	64	95	0.3	32	9,559
Chassis	64	127	0.5	64	11,458
Flippers	128	444	2.7	319	34,310
Head and Neck	64	159	0.8	96	13,357
Battery	128	1,457	11.2	1,341	95,080
Chassis Track	128	2,153	17.0	2,043	136,859
Chassis Side Panels	64	190	1.1	128	15,256
Electronics Box	64	190	1.1	128	15,256
Flipper Hubs	128	1,140	8.5	1,021	76,089
Flipper Track	128	2,407	19.2	2,298	152,051
Flipper Guide	128	1,014	7.4	894	68,493
Head	64	254	1.6	192	19,054
Neck	64	254	1.6	192	19,054
Electronics Package	64	298	2.0	236	21,713
Electronics Housing	64	222	1.3	160	17,155
Head Lid	64	349	2.4	287	24,751
Head Base	64	349	2.4	287	24,751
Neck Tube	64	349	2.4	287	24,751
Neck Actuators	192	875	5.7	689	63,999

Table 6-3 Exploded SUGV Demand and Rampup Rates

Given the failure rates in Table 6-2 it can be seen in Figure 6-2 that the majority of demand in the later years is generated through maintenance. If these rates are illustrative of the realized rates, iRobot has perpetuity in demand for product parts and could likely sell the units to initially outfit the military at a loss. This thesis will not speculate beyond this, but a mention was in order.

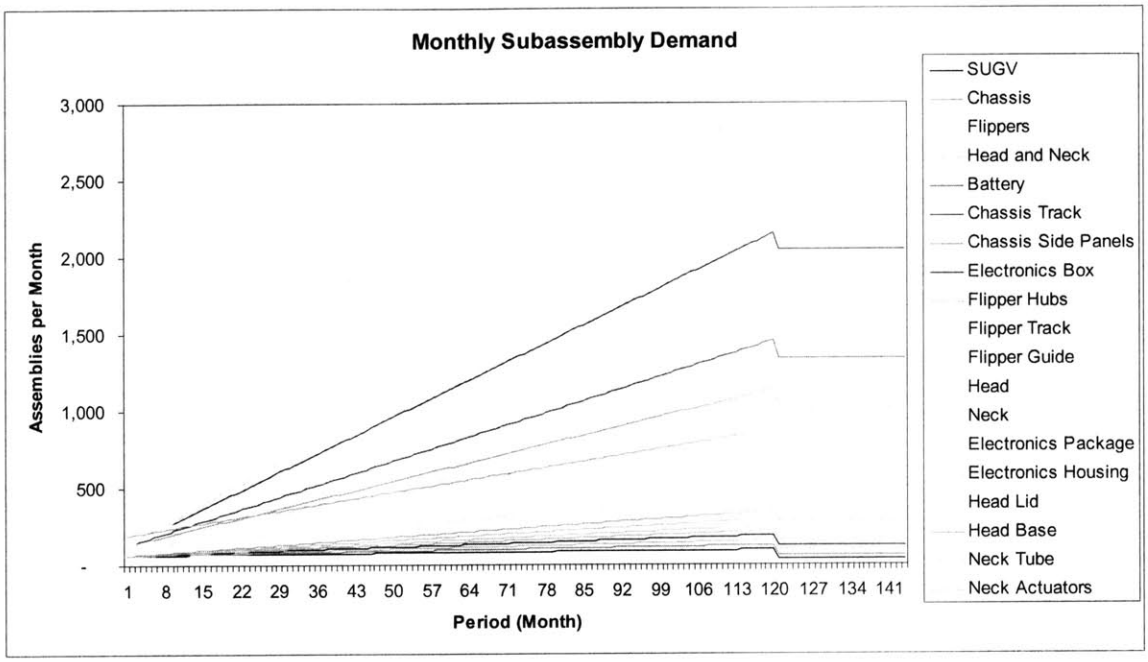


Figure 6-2 Subassembly Demand per Year

These ramp-up rates have implications on the manufacturers. For example, initially, the Flipper Track manufacturer will need to produce 128 tracks per month to meet demand. Each month for 10 years this demand will increase by 19.2 units. At the end of the ten year outfitting period the manufacturer will need to supply 2,407 units per month to meet demand for both new units and failed tracks. After the outfitting period is complete, the manufacturer will need to produce 2,298 units to meet the replacement demand. The Flipper track manufacturer will need to plan production lines and capital expenditures accordingly.

6.4 Supply Chain Model

This section describes the model used to simulate the mock manufacturing environment. The model theory was developed based on observations of the existing operations at iRobot. By specifying the model parameters, the model is customized to a specific implementation, such as the SUGV. This model could be used to simulate several manufacturing environments to obtain: 1) optimal inventory levels, 2) part pricing, 3) on-time delivery expectations, and 4) disruption effects.

The model was implemented using a C/C++ tool developed by the author. The source code is contained in Appendix D -Manufacturing Simulation Tool Source Code.

The section describes the model by: 1) describing the supplier relationships, 2) presenting the manufacturing operation within suppliers, and 3) describing the model's decision variables

6.4.1 Supplier Interaction

One scenario is considered in this model. It is illustrated in Figure 6-3. This scenario took a subassembly contracting approach. It had suppliers creating subassemblies and left the role of final integration to the Lead CM.

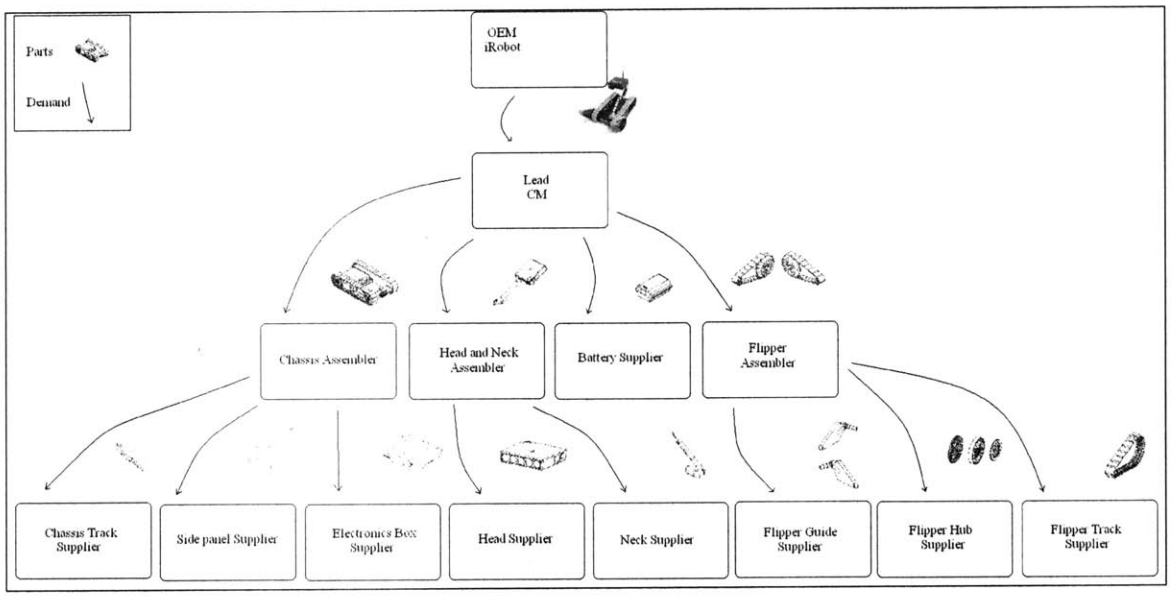


Figure 6-3 Supplier Network

A few of the specifics in the interaction between suppliers in this model are discussed below.

Interaction. Interaction between suppliers is limited to the producers requesting product through demand (orders) and suppliers responding with product.

Delivery Time. All orders have a delivery time window in which the order must be fulfilled or are subject to a late penalty. Once a part is late, it will be shipped individually upon completion and paid for by the late party. Newer orders cannot be fulfilled until all late orders have been filled.

Demand. The model tracks the demand per cycle and allows the supplier to use this information in setting operational parameters. The model assumes demand is normally distributed. All demand must be met; late orders must be filled before other orders.

Part Failure. Parts that fail after shipment are replaced at the part supplier's cost.

Shipping. The cost of shipment is broken down into a fixed cost per shipment and a variable cost for each unit shipped. The model assumes that shipping time is normally distributed for simplicity; it is typically modeled as being poison distributed.

Ownership. Ownership of the parts is transferred to the supplier's customer upon shipment, not upon arrival. Payment is required at the time of shipment.

Sourcing. Each material/subassembly has a single source.

The model had three types of suppliers: OEM, stock suppliers, and modeled suppliers. The OEM type had a predefined demand pattern that mimics the expected demand over the lifetime of the program. The values calculated in Table 6-3 were used as the demand pattern of the OEM. The OEM also automatically reordered parts that were bad upon ownership transfer.

Stock suppliers did not have operational parameters or decision variables. These suppliers fulfilled the materials immediately upon order. Shipping was still applied to the parts and ownership transferred immediately upon order. Parts supplied by stock suppliers were given failure rates similar to those one would expect for these parts.

6.4.2 Internal Operations

This section describes the modeling that takes place for each supplier. The section is broken down into the following areas: 1) inventory flow, 2) storage and holding costs, 3) quality, 4) test coverage and bad part handling, 5) pricing, and 6) model parameters.

Suppliers are modeled as independent agents. They have the ability to make decisions regardless of what is optimal for the environment as a whole. The supplier agents are modeled to make decisions in such a way that their own profits are maximized.

Suppliers choose their operational parameters on a real-time basis. They have access to historical demand, failure rates, delivery times, test effectiveness, production times, and rework times to calculate those parameters. The operating levels the supplier determines are: 1) investment in quality, 2) raw material and finished goods reorder points and quantities, and 3) service levels. Each of these decision variables will be discussed in more detail in section 6.4.3.

6.4.2.1 Inventory Flow

The suppliers' operations are modeled based on multiple raw material inputs, a single output, and a single manufacturing operation to transform the raw material inventory into a finished good as illustrated in Figure 6-4.

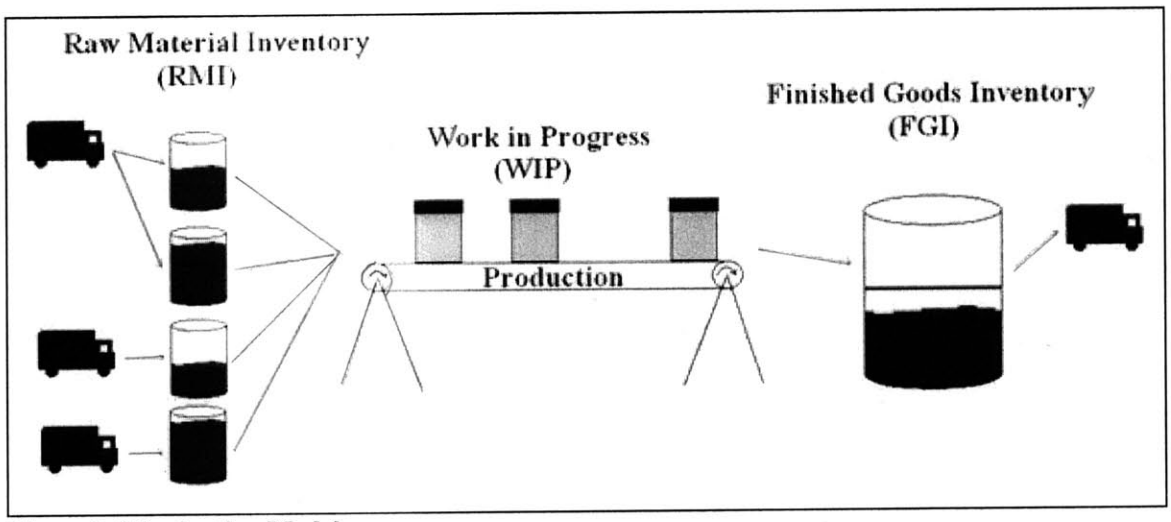


Figure 6-4 Production Model

On the left (pre-production), raw material inventory (RMI) is stored in preparation of undergoing the manufacturing process to generate a finished good. Raw material accumulates here as raw material deliveries are made from suppliers. The RMI is consumed as production runs are made. On the right side (post-production), the finished goods inventory (FGI) awaits orders from customers. Some manufacturers may never build up FGI and others may never build up RMI. The buildup of RMI and FGI is determined by the optimization the individual suppliers make.

The buildup and consumption of RMI and FGI in this model is illustrated in Figure 6-5. If a supplier stocks both RMI and FGI, the RMI and FGI levels will fluctuate as illustrated. As FGI is accumulated, RMI is dispersed. The suppliers set a level at which to replenish RMI; this level is known as the Reorder Point. When inventory levels drop below this level, an order is sent to the supplier (inventory levels are monitored continuously).

FGI is dispersed as orders are fulfilled. In this illustration, orders are fulfilled in bulk at periodic intervals. Suppliers set a minimum level at which to maintain FGI to ensure orders are filled within the delivery window; this level is known as the FGI Safety Level.

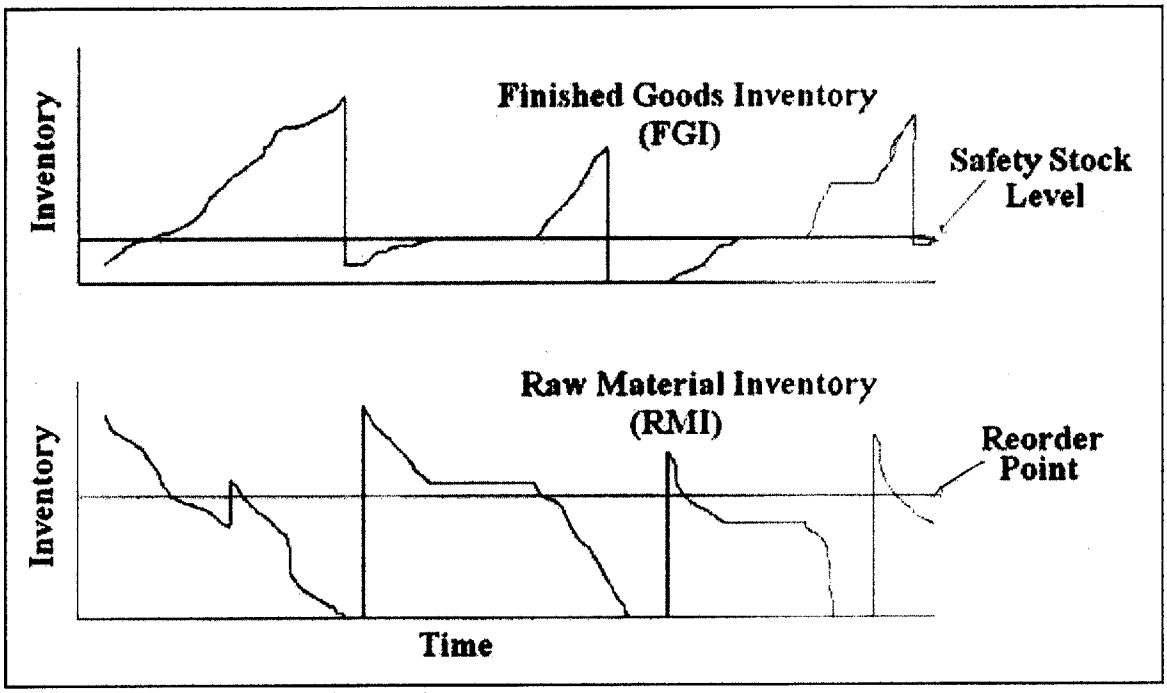


Figure 6-5 Inventory Trend Comparison

6.4.2.2 Storage and Holding Costs

Each part inside a supplier's facility is assessed a storage charge. This charge represents the storage space, handling, and paperwork associated with having material on hand. Each type of material weather raw material or finished goods has a unique storage cost associated with it.

Each supplier is also assigned a weighted average cost of capital (WACC). This WACC is used to represent the cost of holding inventory. When a supplier holds inventory, the invested money used to create that inventory is idle. The WACC creates an incentive to keep inventory levels low. The cost is assessed on the cost of the finished good and the cost of raw material.

The cost to hold inventory is illustrated below:

$$InventoryCost = StorageCost + FinancingCost$$

Where:

$$StorageCost = FGI \bullet FGIStorageCost + \sum_{n} \left(RMI_{n} \bullet RMIStorageCost_{n} \right)$$

and

$$FinancingCost = WACC \bullet \left[FGI \bullet FGICost + \sum_{n} \left(RMI_{n} \bullet RMICost_{n} \right) \right]$$

Combining these two into the Inventory Cost yields:

$$InventoryCost = FGI \bullet \left(FGIStorageCost + FGICost \bullet WACC \right) + \sum_{n} \left[RMI_{n} \bullet \left(RMIStorageCost_{n} + RMICost_{n} \bullet WACC \right) \right]$$

6.4.2.3 Quality

There are three elements of quality considered in this model: 1) inherent design quality, 2) manufacturing process quality, and 3) invested manufacturing quality. These three quality elements were chosen based on the following observations:

Inherent Design Quality. There is a probability that a product will fail due to the way that it is designed. For example: if parts that comprise the product are selected that themselves have high failure rates then the product itself will fail independent of other factors.

Manufacturing Process Quality. Almost all manufacturing processes have yields that are imperfect. This can be due to a number of variations such as: variations in process parameters, variations in inputs, etc. The Manufacturing Process Quality accounts for this type of failure.

Invested Manufacturing Quality. The third quality indicator represents the option to improve the yield of a manufacturing process through investment. For example: investing in additional quality controls to ensure manufacturing process parameters are kept in tight limits. Invested Quality can increase the manufacturing quality, but it does not affect the design quality.

In the model, each supplier is given a manufacturing and design quality. The supplier is not given an invested manufacturing quality, but instead, is given a cost to improve the manufacturing process quality by fifty percent. The supplier analyzes the cost and expected yield improvement to determine its investment in Invested Manufacturing Quality. Because the cost is given for each fifty percent improvement, the cost to make improvements increases exponentially.

To illustrate this process, consider a supplier starting with a manufacturing quality of 95 percent. The supplier has the option of increasing this quality by making an investment. This investment can increase the manufacturing quality by fifty percent to 97.5, but will cost $4000 to do so. To increase the manufacturing quality another fifty percent to 98.75 will cost another $4000.

Figure 6-6 illustrates these three quality indicators. It is important to note that though manufacturing quality can approach 100 percent the design quality will limit the total quality of the part.

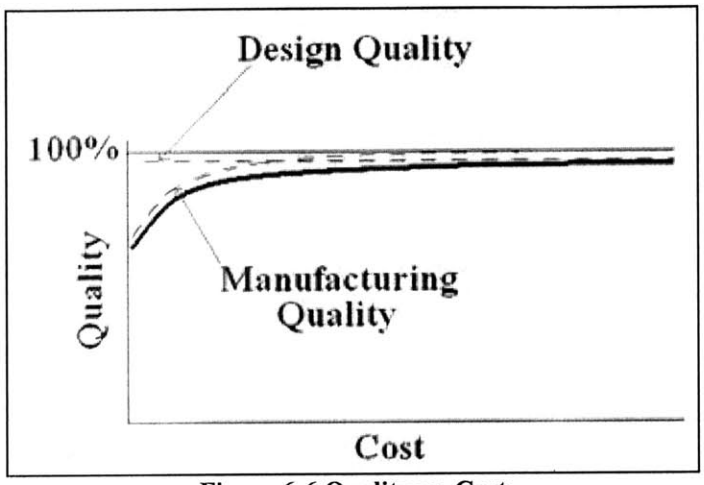


Figure 6-6 Quality vs. Cost

The total quality of a supplier's manufacturing process is the Design Quality times the sum of the Manufacturing Quality and the Investment Quality as illustrated below:

$$Quality = DesignQuality \bullet (ManufacturingQuality + InvestmentQuality)$$

As modeled, Quality is the percentage of parts that do not fail when manufactured.

6.4.2.4 Product failures and Test Coverage

The model considers two types of product failures: 1) constituent part failure, and 2) manufacturing process failure. Constituent part failure occurs when one of the parts constituting a product has failed. Products exhibiting this type of failure can be repaired by replacing the failed part. Manufacturing process failure occurs at random as a product is manufactured. The frequency at which products exhibit manufacturing failure is determined by the supplier's quality. These types of failures can not be repaired and require scrapping the product.

Test coverage represents the ability of the manufacturer to determine if a product has failed. It is measured as percentage of the bad products that are detected prior being shipped to the customer. It is only upon detection that a part will be reworked or scrapped. If a failed part is not detected then it is shipped to the customer just as other parts would be.

6.4.2.4.1 Part rework queue

Products that have bad constituent parts and are detected as such through testing are put into a rework queue. The rework queue has a separate flow time from the production process. Parts are completed on a time basis that follows a normal distribution. The parts in the rework queue can be completed out of the order in which they were introduced.

The rework queue has a separate labor and equipment parameter from that of the production line and there is no capacity limit in the rework queue.

6.4.2.5 FGI Cost and Margin pricing

All pricing is based on suppliers' margin targets. The price of the SUGV in the model is therefore the cost of the raw materials, labor, equipment depreciation, and amortized failures multiplied by one plus the margin. The margin method of pricing was chosen ensure each supplier has positive income.

6.4.2.6 Model Parameters

The following tables identify the parameters associated with each the model. Table 6-4 captures the parameters that must be defined for each supplier. And Table 6-5 captures the parameters that must be defined for each part in the model.

Model Parameter	Description
Contracted Delivery Window (Days)	Contractually obligated time to deliver product after an order is received
Batch Size (units)	Number of units to run on the production at a time
Scheduling Delay Mean (hours)	Mean time to schedule and switch over the production line to the product's configuration
Scheduling Delay Std (hours)	Standard deviation in time to schedule and switch over the production line to the product's configuration
Test Coverage (%)	Percentage of part failures detected before shipment to the customer
Manufacturing Quality (%)	Natural quality level of the manufacturing process (without investment). Percentage of parts produced that do not fail
Design Quality (%)	Percentage of parts that would be good if the manufacturing process were perfect
Line Flow Time Mean (hours)	Mean time to get a batch of product through the manufacturing process
Line Flow Time Std (hours)	Standard deviation in time to get a batch of product through the manufacturing process
Tact Time (hours)	Time between batch starts for the manufacturing process
Rework Flow Time Mean (hours)	Mean time to tear down and rework products that fail due to constituent part failure
Rework Flow Time Std (hours)	Standard deviation in time to tear down and rework products that fail due to constituent part failure
WACC (%)	Weight average cost of capital. Represents opportunity cost, risk, financing cost, etc.
Margin (%)	Average profit margin for the supplier
FGI Storage Cost (year)	Cost to store a single finished good product for one year
Replacement Part Cost ($)	Cost incurred to replace a product that was defective. Represents opportunity cost, good will, shipping, etc.
Late Penalty ($ / unit)	Cost incurred due to shipping product late. Represents shipping, good will, expediting, etc.
Starting Cash ($)	Starting level of cash
Labor Rate ($/hour)	Average cost per hour for the labor force
Equipment Rate ($/hour)	Average cost per hour for the manufacturing equipment.
Fixed Labor Time (hours / batch)	Labor content for each batch

Model Parameter	Description
Variable Labor Time (hours / unit)	Labor content for each product in the batch
Fixed Equipment Time (hours / batch)	Equipment content for each batch
Variable Equipment Time (hours / unit)	Equipment content for each product in the batch
Average tear down and rebuild labor (hours)	Average labor content to perform a tear down and rebuild of a part that has a bad constituent part
Average tear down and rebuild equip. (hours)	Average equipment content to perform a tear down and rebuild of a part that has a bad constituent part

Table 6-4 Supplier Operation Parameters

Parameter	Description
Source (Supplier)	Supplier that manufacturers the part
Destination (Supplier)	Supplier that orders (consumes) the part
Quantity (units)	Quantity of this part that constitutes the destination part (e.g. a bicycle consumes 2 wheels)
Shipping Fixed Cost ($/shipment)	Cost to ship a group of parts from the source to destination
Shipping Variable Cost ($/unit)	Marginal cost to ship an additional unit
Travel Time Mean (hours)	Mean time to ship the part from the source to the destination
Travel Time Std (hours)	Standard deviation in time to ship the part from the source to the destination
Storage Cost ($ / year)	Cost to store the part at the destination's facility for one year
Initial ROP (units)	Initial Reorder point used by the destination to determine when to order new parts
Initial EOQ (units)	Initial order quantity.

Table 6-5 Part Parameters

The parameters presented here customize the model to the specific implementation. Section 6.5 presents the parameters used for the mock SUGV manufacturing environment.

Section 6.4.2 (Internal Operations) has presented the internal operations of each supplier. Before proceeding on to the decision variables, a few additional assumptions must be stated. These assumptions are:

1. All raw materials are required to be present before a manufacturing run can begin; a product cannot be partially built in anticipation of a raw material delivery.
2. All FGI and RMI will be consumed in the long run. There is no concern for obsolescence.

The following section will discuss the decision variables that each supplier calculates in real-time.

6.4.3 Decision Variables

This section describes the calculations performed by each supplier to determine optimal operating levels. These calculations determine: 1) raw material and finished goods reorder points and quantities, 2) service levels, and 3) investment in quality. They are calculated periodically to ensure optimal operating conditions.

Throughout these calculations a number of random variables will be represented. The following annotation will be used to denote a random normal variable: $N(A, \sigma)$, where A is the mean and σ is the standard deviation.

6.4.3.1 ROP and I_s for RMI and FGI

The Reorder Point (ROP) and Safety Stock (I_s) decision variables control when orders are placed with suppliers. These variables are set based on historical demand patterns and a target service level.

The ROP is a function of Monthly Demand, Material Delivery Time, and Service Level.[52] Monthly Demand is a normally distributed random variable representing the historical demand each month. The Material Delivery Time is a normally distributed random variable that represents the time to receive material after placing an order for the material.

---

[52] Anupindi, Ravi, "Managing Business Process Flows," Prentice Hall, 1999

These equations calculate the Lead Time Demand in order to determine the optimal reorder point. The Lead Time Demand represents the expected demand after an RMI replenishment order is made but before the replenishment parts are received.

Reorder point is determined through the probabilistic function outlined below.

Let:

Monthly Demand	=	$N(D, \sigma_D)$
RMI Delivery Lead Time	=	$N(L_T, \sigma_{LT})$
Service Level	=	SL
Safety Inventory	=	I_s
Lead Time Demand	=	$N(D_{LT}, \sigma_{LTD})$

$$N(D_{LT}, \sigma_{LTD}) = N(L_T, \sigma_{LT}) \times N(D, \sigma_D)$$

Ross (1972) showed that the multiplying two random normal variables results in a random normal variable as outlined below:

$$N(X, \sigma_Y) \times N(A, \sigma_B) = N(X \times A, \sqrt{\sigma_Y{}^2 \times A^2 + \sigma_B{}^2 \times X^2})$$

Multiply the Lead Time by the Monthly Demand:

$$D_{LT} = L_T \times D$$

$$\sigma_{LTD} = \sqrt{\sigma_{LT}{}^2 \times D^2 + \sigma_D{}^2 \times L_T{}^2}$$

Safety inventory is the inverse standard normal cumulative distribution evaluated at the service level multiplied by the standard deviation:

$$I_s = f(SL) \times \sigma_{LTD}$$

Where *f(x)* is the inverse standard normal cumulative distribution function, also known as the quantile function, which is defined as follows:[53]

$$f(p) = \sqrt{2} \times erf^{-1}(2p - 1)$$

---

[53] M.D. Springer, Dale, Melvin, "The Algebra of Random Variables", Wiley, 1979.

Where $erf^{-1}(x)$ is the inverse error function, which is defined as:

$$erf^{-1}(x) = \sum_{k=0}^{\infty} \frac{c_k}{2k+1} \left(\frac{\sqrt{\pi}}{2} x \right)^{2k+1}$$

Where c_k is defined as:

$$\boxed{\begin{aligned} c_0 &= 1 \\ c_k &= \sum_{m=0}^{k-1} \frac{c_m c_{k-1-m}}{(m+1)(2m+1)} \end{aligned}}$$

The reorder point is defined as the mean Lead Time Demand plus the safety stock as follows:

$$\boxed{ROP = D_{LT} + I_s}$$

The suppliers recalculate the ROPs for each of the raw material inputs every 15 day. 15 was chosen because it was frequent enough to detect changes in demand and infrequent enough to allow the model to run at a reasonable pace.

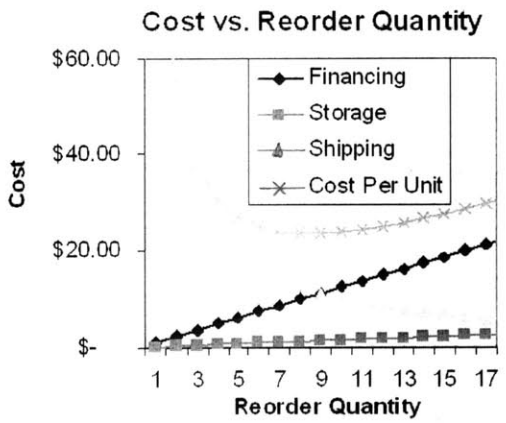
6.4.3.2 Economical Order Quantity (EOQ)

The Economical Order Quantity (EOQ) is the optimal quantity of material to order each time material levels drop below the ROP. The EOQ is selected to minimize the cost of shipping and holding inventory. As the reorder quantity increases, storage and financing costs increase. And as the reorder quantity decreases, the cost of shipping each unit increases. This model does not consider economical lot sizes, which could reflect discounts for full trucks, planes, containers, etc.

The cost as a function of the reorder quantity is illustrated here. Each of the variables is detailed in Table 6-4 and Table 6-5.

Let:

Monthly Demand	=	$N(D, \sigma_D)$
Reorder Quantity	=	Q
Shipping Fixed Cost	=	C_{ship}
Material Cost	=	C_m
WACC	=	W
Storage Cost	=	C_{store}
Total Cost	=	C_T



$$C_T(Q) = \frac{D \times C_{ship}}{Q} + \frac{C_m \times W + C_{store}}{2} \times Q$$

Minimize by taking the derivative and setting it equal to zero.

$$\frac{\partial C_T(Q)}{\partial Q} = \frac{C_m \times W + C_{ship}}{2} - \frac{D \times C_{store}}{Q^2} = 0$$

Solving for Q.

$$Q = \sqrt{\frac{2 \times D \times C_{ship}}{C_m \times W + C_{store}}}$$

It can be seen that the Economical Order Quantity is proportional to the monthly demand and the shipping cost and inversely proportional to the material cost and storage cost.

6.4.3.3 Service Level

A service level of 99.97% is used throughout the supplier network because, at the time of this writing, there was not a closed form solution to the service level. This applies to the availability of Raw Material Inventory for production runs and Finished Good Inventory for spontaneous orders. The equations for determining the optimal service level are presented, but the solution is left to future research.

The setup for this calculation takes the following steps: 1) establish an equation for the total profit of the firm as a function of the average finished good inventory, raw material inventories, and invested quality, 2) take the expected value of the profit function, 3) differentiate with respect to the average inventory, 4) set it equal to zero, and 5) solve for the average inventory.

A walkthrough of this process is outlined in the equations below using a simplified service level profit function. Each of the variables is detailed in Table 6-4 and Table 6-5.

Let:

Total Profit	=	π
FGI Cost	=	C_{FGI}
Margin	=	M
Demand	=	$N(D, \sigma_D)$
FGI Average Stock	=	Q_{FGI}
Storage Cost	=	C_{store}
WACC	=	W
Late Penalty Cost	=	$C_{penalty}$

$$\pi(Q_{FGI}) = (C_{FGI} \times M) \times \min(D, Q_{FGI}) - (C_{store} + W \times C_{FGI}) \times Q_{FGI} - C_{penalty} \times \max(D - Q_{FGI}, 0)$$

This equation states that the profit in a given order cycle is determined by the profit on a given sale ($C_{FGI} x\ M$) times the number of parts sold. The number of parts sold is the minimum of the demand and the number of parts on hand ($min(D, Q_{FGI})$). This profit is offset by the cost of having the material on hand, which is determined by the cost of storage ($C_{store} x\ Q_{FGI}$) plus the financing cost ($W x\ C_{FGI} x\ Q_{FGI}$). In addition to these basic constructs, there is a penalty assessed on the supplier for not fulfilling demand ($C_{penalty} x\ max(0, D\text{-}Q_{FGI})$).

Taking the expected value:

$$\begin{aligned} E[\pi(Q_{FGI})] = {} & (C_{FGI} \times M) \times E[\min(D, Q_{FGI})] \\ & - (C_{store} + W \times C_{FGI}) \times Q_{FGI} \\ & - C_{penalty} \times E[\max(D - Q_{FGI}, 0)] \end{aligned}$$

The expected value of the minimum of a random variable and a constant is illustrated in the equations below.

$$E[min(R, C)] = E[R] \times Prob(R \leq C) + E[C] \times Prob(C \leq R)$$

It can be shown that the Prob ($C \leq R$) equals 1 – Prob ($R \leq C$). Substitute in the equation above yields:

$$E[min(R,C)] = E[R] \times Prob(R \leq C) + E[C] \times (1 - Prob(R \leq C))$$

Collecting like terms

$$E[min(R,C)] = E[C] + (E[R] - E[C]) \times Prob(R \leq C)$$

The expected value of a constant equals the constant, therefore this equation simplifies further to:

$$\boxed{E[min(R,C)] = C + (E[R] - C) \times Prob(R \leq C)}$$

The expected value of the maximum of 0 and the difference between a random variable and a constant is illustrated in the equations below.

$$E[max(R - C, 0)] = E[R - C] \times Prob(R - C \geq 0) + E[0] \times Prob(0 \geq R - C)$$

The expected value of a constant equals the constant, therefore this equation simplifies to:

$$E[max(R - C, 0)] = (E[R] - C) \times Prob(R - C \geq 0)$$

Rearranging terms in the inequality

$$E[max(R - C, 0)] = (E[R] - C) \times Prob(R \geq C)$$

Rearranging the probability to be in the same form as the minimum equation above

$$E[max(R - C, 0)] = (E[R] - C) \times (1 - Prob(R \leq C))$$

Expanding

$$\boxed{E[max(R - C, 0)] = (E[R] - C) - (E[R] - C) \times Prob(R \leq C)}$$

Substitution the expected values of the minimum and maximum arguments into the expected value profit function yields:

$$E[\pi(Q_{FGI})] = (C_{FGI} \times M) \times (Q_{FGI} + (E[D] - Q_{FGI}) \times Prob(D \le Q_{FGI}))$$
$$- (C_{store} + W \times C_{FGI}) \times Q_{FGI}$$
$$- C_{penalty} \times (E[D] - Q_{FGI} - (E[D] - Q_{FGI}) \times Prob(D \le Q_{FGI}))$$

It can be shown that a normally distributed random variable plus a constant shifts the mean of the random variable as illustrated below.

$$N(A, \sigma) + Const = N(A + Const, \sigma)$$

It can also be shown that a normally distributed random variable minus a constant shifts the mean of the random variable as illustrated below.

$$N(A, \sigma) - Const = N(A - Const, \sigma)$$

It can also be shown that the expected value of a normally distributed random variable is equal to the variable's mean as illustrated below.

$$E[N(A, \sigma)] = A$$

Utilizing these random normal relationships the expected value of the profit function reduces to:

$$E[\pi(Q_{FGI})] = (C_{FGI} \times M) \times (Q_{FGI} + (D - Q_{FGI}) \times Prob(D \le Q_{FGI}))$$
$$- (C_{store} + W \times C_{FGI}) \times Q_{FGI}$$
$$- C_{penalty} \times (D - Q_{FGI} - (D - Q_{FGI}) \times Prob(D \le Q_{FGI}))$$

Multiplying out the terms yields:

$$E[\pi(Q_{FGI})] = C_{FGI} M Q_{FGI} + C_{FGI} M D Prob(D \le Q_{FGI}) - C_{FGI} M Q_{FGI} Prob(D \le Q_{FGI})$$
$$- (C_{store} + W C_{FGI}) Q_{FGI}$$
$$- C_{penalty} D + C_{penalty} Q_{FGI} + C_{penalty} D Prob(D \le Q_{FGI}) - C_{penalty} Q_{FGI} Prob(D \le Q_{FGI})$$

The final step in the process of calculating the service level is taking the derivative of the expected value, setting it equal to zero, and solving for the Prob($D \le Q_{FGI}$), which is also known as the service level.

Taking the derivative and setting it equal to zero.

$$\frac{\partial E[\pi(Q_{FGI})]}{\partial Q_{FGI}} = C_{FGI}M - C_{FGI}MProb(D \le Q_{FGI}) - (C_{store} + WC_{FGI}) + C_{penalty} - C_{penalty}Prob(D \le Q_{FGI}) = 0$$

Reducing and collecting like terms.

$$C_{FGI}M - (C_{store} + WC_{FGI}) + C_{penalty} = (C_{FGI}M + C_{penalty})Prob(D \le Q_{FGI})$$

solving for the service level.

$$Prob(D \le Q_{FGI}) = \frac{C_{FGI}M - C_{store} - WC_{FGI} + C_{penalty}}{C_{FGI}M + C_{penalty}}$$

These steps have solved the service level of a simplified version of the total profit function. This profit function lacks in a few areas. First, it only considers the instantaneous availability of finished goods. In reality there is a window from the time an order for goods is received and the time the order must be filled. During this window additional units could be run off of production. Second, it does not consider investment in quality to bring the manufacturing yields up.

The following section presents a unified service level equation that does capture finished good production and invested quality, but as of this writing there was no closed form solution.

6.4.3.4 Unified Service Level Profit Equation

To determine a service level that optimizes the entire supplier's operations the total profit function must consider the finished goods inventory, investment quality, production yields and timing, raw material inventory, raw material failure rates, and raw material delivery variability. The following changes must be made to the equation previously introduced.

Q_{FGI} must be changed from the instantaneous number of finished goods available to the total number of finished goods that can be available once the order must be fulfilled.

This value must consider the number of finished goods that can get through the production process in the time allowed. It must also consider the availability of raw materials to feed that production run. If the necessary raw materials to fulfill the order are not stored in inventory, the delivery lead times for raw material must also be considered.

Let:

FGI available at fulfillment	$= Q$
Demand	$= D$
Order Window Production Capacity	$= P_{capacity}$
Production Batch Size	$= B_{size}$
Average FGI	$= Q_{FGI}$
Order fulfillment window	$= T_{avail}$
Production line setup delay	$= T_{LineDelay}$
Production flow time	$= T_{flow}$
Design Quality	$= Q_D$
Manufacturing Quality	$= Q_M$
Invested Quality	$= Q_I$
On order production capacity	$= P_{OrderCapacity}$
Average Raw Material	$= R_1, R_2, R_3, \ldots, R_n$
Raw Material Delivery Time	$= T_{R1}, T_{R2}, T_{R3}, \ldots, T_{Rn}$
Raw Material Failure Rates	$= F_1, F_2, F_3, \ldots, F_n$
Failed Part Rework Time	$= T_{rework}$
Later Delivery Penalty	$= C_{penalty}$
FGI Storage Cost	$= C_{FGIStore}$
RMI Storage Costs	$= C_{RMInStore}$
RMI Costs	$= C_{RMI}$
WACC	$= W$

$$P_{capacity} = \left(\frac{T_{avail} - T_{LineDelay}}{T_{flow}} B_{size} \prod_n (1 - F_n) + \frac{T_{avail} - T_{LineDelay}}{T_{rework}} B_{size} \left(1 - \prod_n (1 - F_n) \right) \right) Q_D (Q_M + Q_I)$$

$$P_{OrderCapacity} = \left(\frac{T_{avail} - T_{Rn}}{T_{flow}} B_{size} \prod_n (1 - F_n) + \frac{T_{avail} - T_{Rn}}{T_{rework}} B_{size} \left(1 - \prod_n (1 - F_n) \right) \right) Q_D (Q_M + Q_I)$$

$P_{capacity}$ is the total possible number of finished goods that can be produced during the order window. It assumes that raw material is readily available. The $P_{OrderCapacity}$ is the

total possible number of the finished goods that can be produced given raw material is not readily available. It assumes that raw material is ordered at the time of order receipt.

Putting it all together into a unified total profit function yields:

$$\begin{aligned}\pi(Q_{FGI}, Q_I, R_n) = {} & (C_{FGI} M)\min(D, Q_{FGI} + \min(P_{capacity}, \min(R_n + P_{OrderCapacity}))) \\ & - Q_{FGI}(C_{FGIstore} + C_{FGI} W) - \sum_n (R_n (C_{RMInStore} + C_{RMIn} W)) \\ & - C_{penalty} \times \max(0, D - Q_{FGI} + \min(P_{capacity}, \min(R_n + P_{OrderCapacity})))\end{aligned}$$

Because this equation requires taking the expected value of the minimum of several random variables there is no closed form solution. As a result, a service level of 99.97% is used throughout the model. Future research can consider an iterative or Newtonian approach to solving for this value at run-time.

This concludes the discussion on the model's internal operations. Several parameters have been introduced, all of which must be determined either through direct interaction with potential supplies or through qualitative measures. The next section will introduce the model parameters for the mock SUGV environment.

6.5 Model Parameters

The model's parameters were selected to represent the SUGV's environment once established. A detailed analysis was not undertaken to generate these parameters as iRobot is awaiting RFI responses as of this writing. The selected parameters are meant to illustrate the model's capability in determining: 1) optimal inventory levels, 2) part pricing, 3) on-time delivery expectations, and 4) disruption effects. They were not selected to provide iRobot with realistic SUGV data.

The selected model parameters are detailed in Appendix E -Supplier Network Model Parameters. In addition, the raw materials used in this model are captured in Table 6-6. These values were selected as a representative set of suppliers for this effort.

Material	Cost ($)	Failure Rate (%)
Battery Components	35	2
Track Rubber	0.50	1
Aluminum	200	2
Electronics Chips / Components	3400	5
Head Components	2500	4
Neck Components	1000	3
Plastic Pellets	1.5	3
Hub Plastic Stock	40	4

Table 6-6 Stock Raw Material Prices and Quality

The ramp rates for demand indicated in Table 6-3 were used to drive demand increases in the model. The next section illustrates the results of this model running with these parameters.

6.6 Model Results

The model was executed over the first 300 simulated days of operation. It assumes three shifts are running at each of the suppliers. The demand for end product and replacement parts followed a normal distribution with a mean equal to the ramp rates in Table 6-3 and a standard deviation of 5. Each period, the mean of the demands was increased by the ramp rate. Demand was received by the suppliers from the OEM ever 30 days. Based on the operational parameters selected and the supplier scenario, the total cost for this SUGV is estimated to be $14,429.

In the first and second months of operation the SUGV was only delivered on time 68% and 73% respectively. This was due to the model beginning with an empty pipeline of subassemblies. After the first two periods, the supply chain functions at 100% on-time delivery for the remaining periods in the 300 day simulation.

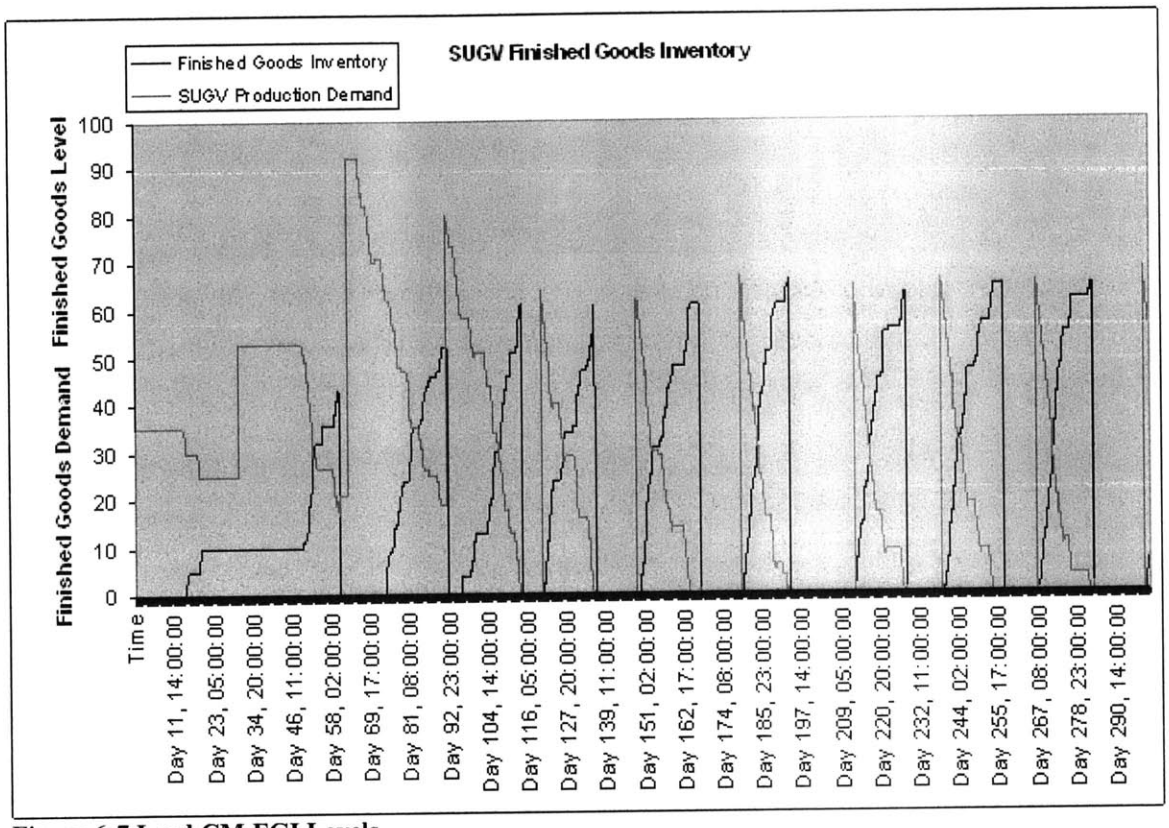


Figure 6-7 Lead CM FGI Levels

Figure 6-7 was captured from the lead integrator. It illustrates the finished goods inventory level and the production demand over the 300 day simulation. The FGI level can be seen to increase at a decent rate then dramatically reduce at a periodic point. This is due to the Lead integrator building inventory to meet demand, then shipping what is on hand to the OEM as the contracted delivery window closes. The Production demand indicates the demand the OEM brought on in addition to the safety level of inventory the Lead CM would like to hold. The Lead CM keeps a desired safety inventory level of 35 units, which accounts for the demand shown before the first order (30 days).

To meet these demand patterns, the Lead CM selected reorder points for its raw material as follows: Chassis ROP of 63 and an EOQ 16. Head and Neck ROP of 45 and an EOQ of 12. Battery ROP of 224 and an EOQ of 61. Flipper ROP of 47 and an EOQ of 17.

To illustrate the ROP and EOQ values, the Lead CM monitors its number of Chassis on hand. If the number on hand drops below 63 it orders an additional 16. The chassis

inventory held by the Lead CM is illustrated in Figure 6-8. The cyclical pattern established after day 160 indicates an optimal solution has been established. The right quantity of chassis has being selected to meet the fluctuating demand.

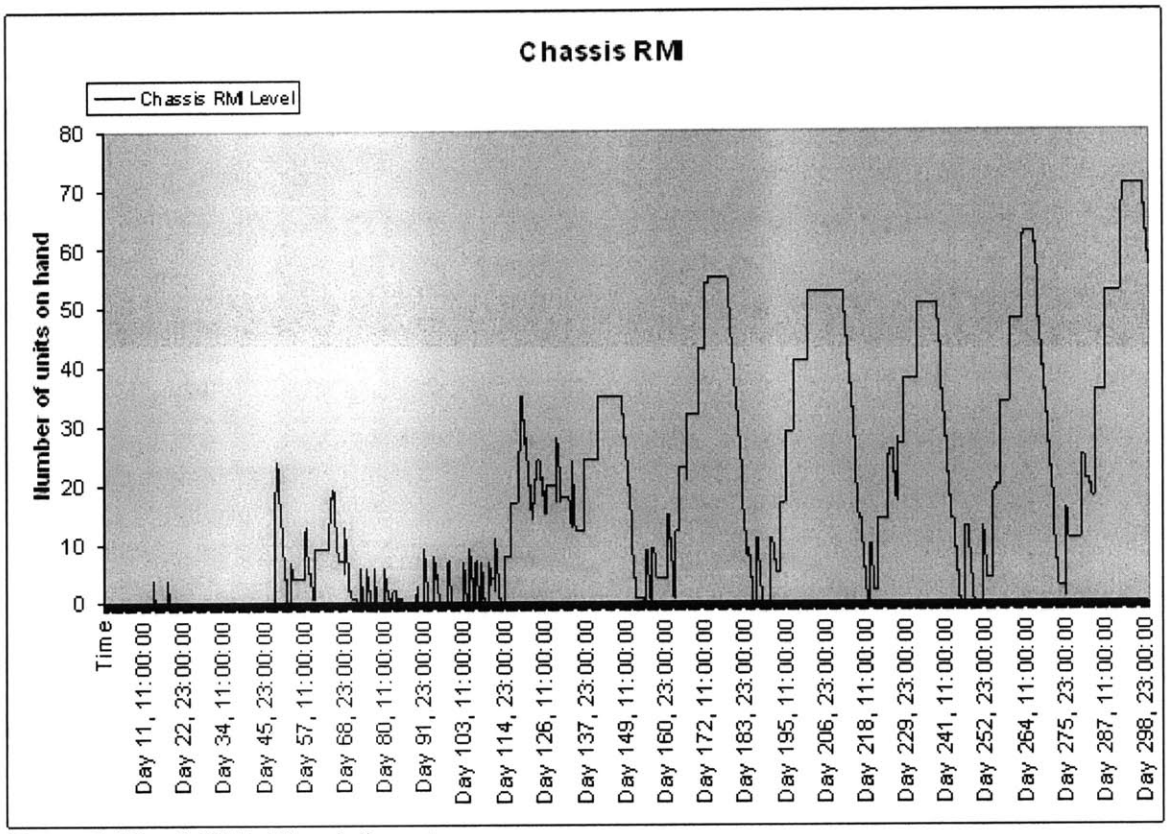


Figure 6-8 Lead CM's Chassis Inventory

This model does not account for major disruptions in the flow of material, such as component changes, supplier changes, or general obsolescence. It is also a simplified representation of the true SUGV.

6.6.1 Disruption Modeling

Disruptions can be introduced by hard coding them into the model. To illustrate the potential, a supply route disruption was added to the model on the 200th day. The disruption causes a permanent delay of 55 days in the delivery of SUGV Chassis to the Lead CM. The model was executed for 400 days to allow the supplier network to adjust to the change in delivery time.

The net effect of this disruption was: 1) an increase in the SUGV cost to $14,524 from $14,429, 2) 178 SUGVs were delivered late to the customer, 3) the reorder point for chassis rose from 63 to 438 and the economical order quantity rose from 16 to 132.

Figure 6-9 illustrates the SUGV demand against the inventory levels at the Lead CM. It can be seen that around Day 212 the disruption began to have repercussions. The demand soared to a maximum point around 160 units 75 days after the event. After 125 days however, the Lead CM had compensated for the disruption, and the demand and inventory reached equilibrium again.

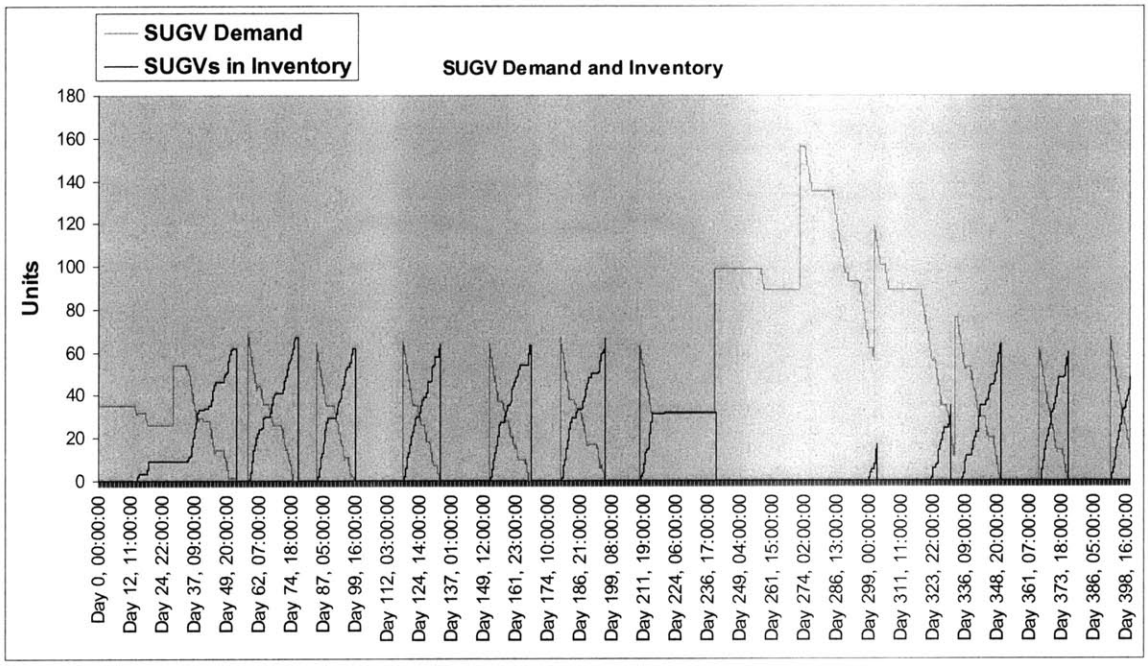


Figure 6-9 Supply Disruption Case Demand vs. Inventory

Figure 6-10 illustrates the effects the disruption had on the raw material inventory held by the Lead CM. The blue line indicates the number of chassis being held in inventory at the Lead CM's facilities. The magenta line indicates the number of chassis in the shipping channel. It can be seen that around day 200 the number of chassis in the shipping channel dramatically increased due to the increased delivery time.

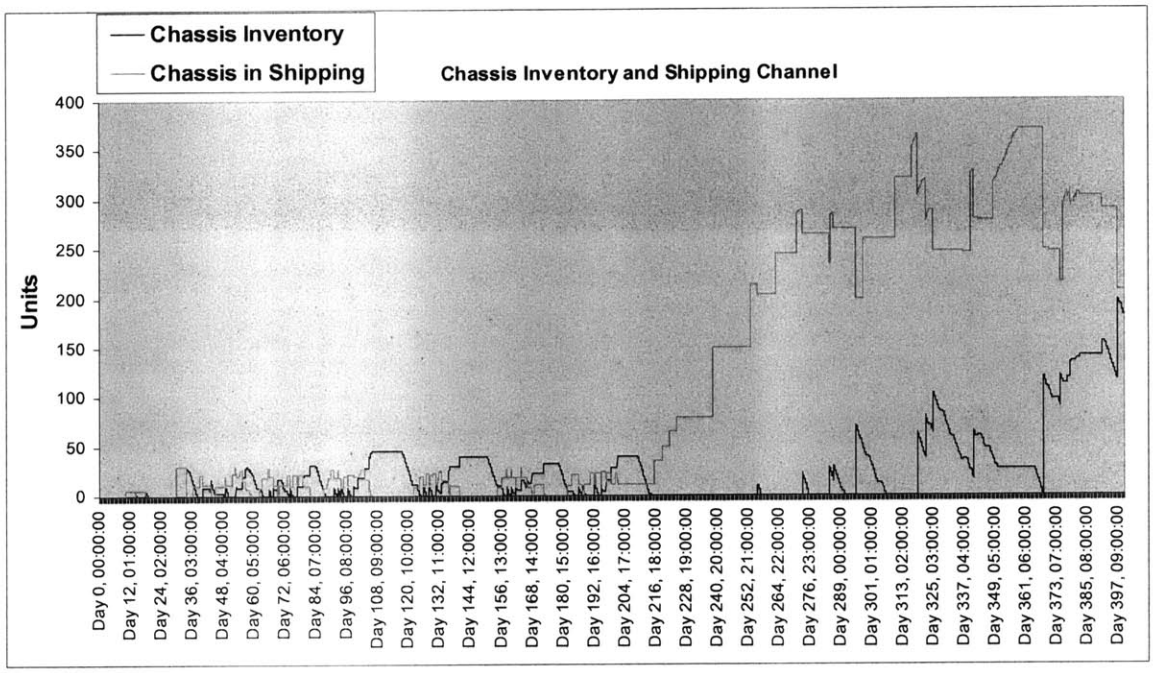


Figure 6-10 Supply Disruption Case Chassis Inventory vs. Shipping

Figure 6-11 illustrates the trend in the cost per SUGV. The cost stabilized at around $14,400 prior to the disruption. Around day 200, the cost trended upwards until it stabilized again around $15,520. This is primarily due to the increased carrying costs of chassis inventory. The Lead CM must finance the additional chassis due to the dramatic increase in the reorder point.

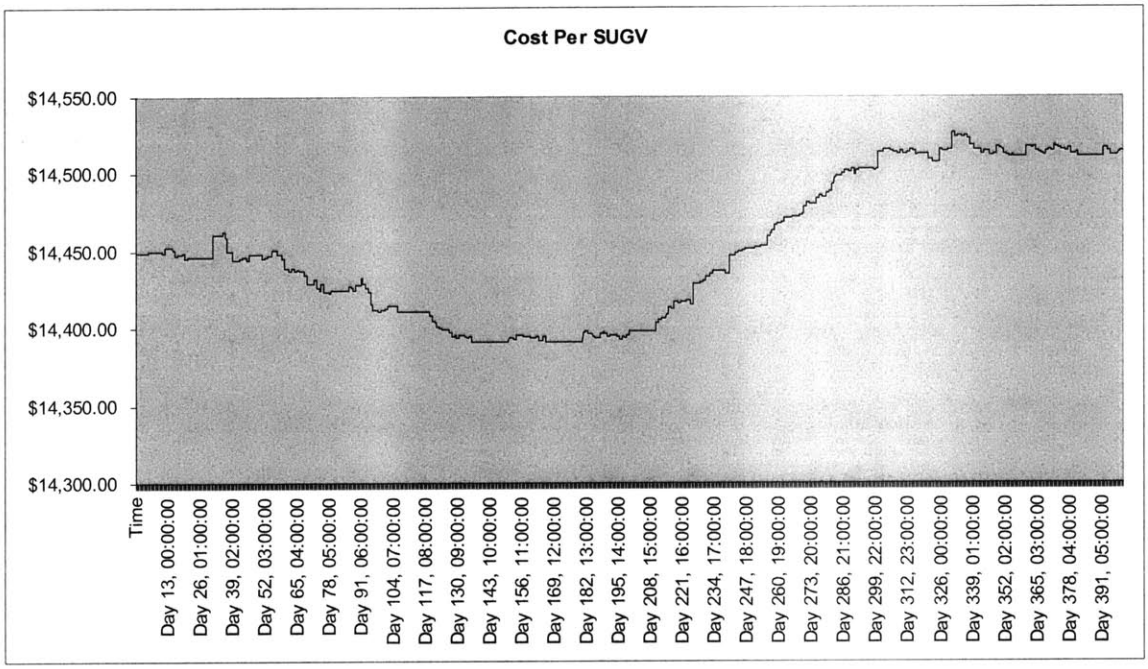


Figure 6-11 Supply Disruption Case SUGV Cost Trend

This disruption case illustrates the capabilities of the model in predicting the outcome of adverse events. If iRobot were to run this disruption case on the actual SUGV environment, it could anticipate the delays in material flow, increases in cost, and the duration of impact.

6.7 Chapter Summary

This chapter has detailed a simulation model for use by iRobot in evaluating the manufacturing environment of the SUGV. The actual environment is still under consideration by iRobot so a mock environment was discussed and modeled. The chapter detailed the order cycle and supplier interactions, the simulated parts and their interrelations, and the volume forecasts for both outfitting and maintaining military units.

These items were used in the simulation model to evaluate: 1) optimal inventory levels, 2) part pricing, 3) on-time delivery expectations, and 4) disruption effects.

This page is intentionally left blank

7 Conclusion

This thesis presented a method for selecting lead contract manufacturers for defense projects. The selection method reduces the risks to the Department of Defense and the Defense Contractor in outsourcing manufacturing to contract manufacturing organizations by: 1) analyzing relevant macroeconomic, socioeconomic, and political aspects of potential contractors' geographical locations, 2) evaluating financial performance, 3) developing key characteristics for the manufacturing environment, and 4) modeling supplier operations.

The process provides a mechanism to simulate the manufacturing environment and determine potential supply issues prior to implementation. With small modifications to the model, iRobot and others can simulate the effects of providing loans to suppliers, having parts go obsolete, and having delivery route disruptions. These scenario simulations allow the user to anticipate supply disruptions and develop mitigation strategies.

The process also provides a defensible selection as required by the Federal Acquisition Regulations. It does so by remaining objective throughout the selection process. Objectivity is assured by: 1) systematically finding qualified contract manufacturers, 2) independently developing selection criteria, and 3) evaluating those contractors in a systematic manner.

This thesis presented a tool that allows potential contract manufacturing partners to quickly come up to speed on product specifications. The tool allows the user to relationally browse the documentation provided by an OEM to gain a deeper understanding of the part relationships. As a result, this tool can increase the speed and accuracy of RFI responses from Contract Manufacturers.

The research in this thesis is not only applicable to the Defense and Robotics industries, but it is also applicable to industries involving general manufacturing outsourcing decisions. The macro and socioeconomic evaluation, financial analysis, key

characteristics generation, and simulation are directly applicable to outsourcing decisions in most manufacturing industries.

This thesis research can be enhanced by taking a deeper look into the following areas:

State Public Policy Shifts for Industry Attraction. The public policy implications introduced section 4.1.16.1 could be expanded with additional cases. It would be useful to take a deeper look at States' ability to make changes that would attract specific industries.

Solution to the Model's Service Level. The Manufacturing environment model presented in chapter 6 does not have a solution to the optimal service level. A unifying profit function is presented in section 6.4.3.4, but a closed form solution does not exist. Modifications to the simulation application could include an iterative or Newtonian approach to solving for the optimal service level at run-time. This would increase the accuracy of the model and provide a better indication to the user as to the percentage of on-time deliveries.

7.1 Recommendations

Electro-mechanical testing. The robotics industry is quite unusual in its level of electro-mechanical integration. This integration requires specific skills in testing and debugging product builds. Suppliers should be evaluated based on their ability to test and debug electro-mechanical devices not just their ability to manufacture.

Handling and Storage Space. The storage and handling requirements for the SUGV are significantly different from traditional electronic contract manufacturing. iRobot should anticipate additional costs associated with these differences.

Ownership of tooling, fixtures, and procedures. iRobot's owning the tooling, fixtures, assembly procedures, and test procedures creates a hands off approach for the Contract Manufacturer. iRobot should look for ways to include the contract manufacturer in ownership of at least the assembly and test procedures.

Commodity purchasing to supplier management. Purchasing commodities, such as chips, capacitors, and resistors is a much different job than purchasing the custom hardware required for the SUGV. A significant learning curve should be anticipated in this effort.

Keep final testing with the manufacturer of the most complex system. Place the final integration and test work with the manufacturer of the most complex subsystem. This will keep ownership of the most complex parts with those closest to the part's manufacture.

Appendix A - State Risk Assessment Data

	BEA Summary Stats '97-'05						
State	GSP (millions)	GSP CAGR	GSP CAGR STD	NAICS GSP CAGR	NAICS GSP Growth STD	NAICS Industry Size (millions)	Industry Size %(GSP)
Alabama	$ 149,796	4.87%	1.78%	0.83%	5.67%	$ 717	0.48%
Alaska	$ 39,872	5.99%	7.31%	6.30%	54.41%	$ 6	0.01%
Arizona	$ 215,759	6.81%	2.26%	-3.29%	9.67%	$ 2,132	0.99%
Arkansas	$ 86,802	4.90%	2.23%	1.90%	6.87%	$ 738	0.85%
California	$ 1,621,843	5.98%	2.78%	-1.01%	7.58%	$ 13,532	0.83%
Colorado	$ 216,064	6.26%	2.80%	-0.69%	7.64%	$ 1,230	0.57%
Connecticut	$ 194,469	4.41%	2.41%	-4.78%	27.78%	$ 1,243	0.64%
Delaware	$ 54,354	5.47%	2.03%	1.64%	8.33%	$ 125	0.23%
District of Columbia	$ 82,777	6.41%	2.36%	4.05%	10.72%	$ 18	0.02%
Florida	$ 674,049	7.03%	1.94%	3.02%	18.35%	$ 2,396	0.36%
Georgia	$ 364,310	5.50%	2.34%	2.11%	9.14%	$ 1,622	0.45%
Hawaii	$ 53,710	4.58%	2.63%	-6.02%	12.30%	$ 10	0.02%
Idaho	$ 47,178	6.50%	3.75%	12.82%	8.60%	$ 824	1.75%
Illinois	$ 560,236	4.17%	1.09%	1.29%	14.53%	$ 5,275	0.94%
Indiana	$ 238,638	4.48%	1.89%	3.22%	7.14%	$ 2,919	1.22%
Iowa	$ 114,291	4.25%	1.98%	2.64%	4.97%	$ 1,739	1.52%
Kansas	$ 105,448	4.87%	1.20%	2.69%	8.25%	$ 616	0.58%
Kentucky	$ 140,359	3.61%	2.28%	-3.06%	37.60%	$ 812	0.58%
Louisiana	$ 166,310	4.92%	3.18%	-0.72%	6.79%	$ 291	0.17%
Maine	$ 45,070	4.84%	1.55%	-0.84%	19.90%	$ 144	0.32%
Maryland	$ 244,899	5.96%	0.88%	-1.10%	7.97%	$ 752	0.31%
Massachusetts	$ 328,535	5.03%	2.51%	0.45%	15.52%	$ 3,800	1.16%
Michigan	$ 377,895	2.97%	1.99%	-0.76%	31.70%	$ 2,401	0.64%
Minnesota	$ 233,292	5.16%	1.55%	3.99%	11.48%	$ 2,669	1.14%
Mississippi	$ 80,197	4.14%	1.60%	-1.74%	14.97%	$ 429	0.54%
Missouri	$ 216,069	3.97%	0.87%	-1.16%	10.57%	$ 1,123	0.52%
Montana	$ 29,851	5.71%	2.19%	-0.23%	9.54%	$ 46	0.15%
Nebraska	$ 70,263	4.20%	1.74%	0.94%	7.01%	$ 510	0.73%
Nevada	$ 110,546	7.96%	2.62%	9.27%	3.38%	$ 253	0.23%
New Hampshire	$ 55,690	5.40%	2.37%	-6.76%	15.87%	$ 732	1.31%
New Jersey	$ 430,787	4.59%	0.93%	1.18%	9.04%	$ 1,681	0.39%
New Mexico	$ 69,324	4.86%	4.82%	-7.77%	6.02%	$ 1,133	1.63%
New York	$ 963,466	4.95%	1.91%	-2.44%	7.59%	$ 3,795	0.39%
North Carolina	$ 344,641	5.25%	1.53%	-2.62%	18.66%	$ 2,536	0.74%
North Dakota	$ 24,178	5.04%	2.86%	7.65%	10.01%	$ 198	0.82%
Ohio	$ 442,440	3.65%	1.46%	-0.50%	15.20%	$ 3,664	0.83%
Oklahoma	$ 120,549	5.59%	2.35%	-4.22%	28.52%	$ 642	0.53%
Oregon	$ 145,351	5.24%	3.96%	6.75%	13.69%	$ 3,539	2.43%
Pennsylvania	$ 487,169	4.47%	0.76%	-2.07%	10.62%	$ 3,051	0.63%
Rhode Island	$ 43,791	5.51%	1.71%	1.20%	7.33%	$ 459	1.05%
South Carolina	$ 139,771	4.62%	1.15%	2.82%	17.11%	$ 1,133	0.81%
South Dakota	$ 31,066	5.79%	2.56%	1.85%	19.04%	$ 361	1.16%
Tennessee	$ 226,502	4.99%	1.46%	6.97%	6.22%	$ 2,553	1.13%
Texas	$ 982,403	6.37%	2.26%	0.94%	22.75%	$ 7,919	0.81%
Utah	$ 89,836	5.95%	1.91%	2.29%	15.78%	$ 562	0.63%
Vermont	$ 23,134	5.42%	0.86%	2.18%	13.05%	$ 308	1.33%
Virginia	$ 352,745	6.58%	1.60%	2.40%	4.78%	$ 1,137	0.32%
Washington	$ 268,502	5.25%	3.04%	1.88%	24.15%	$ 895	0.33%
West Virginia	$ 53,782	4.17%	2.32%	-1.09%	10.13%	$ 95	0.18%
Wisconsin	$ 217,537	4.62%	0.97%	0.00%	6.31%	$ 2,848	1.31%
Wyoming	$ 27,422	7.92%	4.49%	-3.09%	9.97%	$ 13	0.05%

	BLS Summary Stats 2005		BLS Summary Stats '01 - '05		US Census (Fina
State	labor force	unemployment rate	Hourly Wage (Industry Wide)	NAICS Average	State Public Debt (millions)
Alabama	2,154,897	4.00%	$ 16.63	$ 20.65	$ 21,629.26
Alaska	339,305	6.80%	$ 19.33	$ 15.81	$ 8,625.69
Arizona	2,843,997	4.70%	$ 18.35	$ 26.64	$ 29,843.53
Arkansas	1,361,844	4.90%	$ 15.03	$ 18.37	$ 10,409.15
California	17,695,567	5.40%	$ 22.20	$ 31.44	$ 269,934.55
Colorado	2,547,895	5.00%	$ 20.00	$ 28.29	$ 33,841.10
Connecticut	1,817,025	4.90%	$ 25.48	$ 28.93	$ 30,515.93
Delaware	438,003	4.20%	$ 21.48	$ 22.54	$ 6,052.63
District of Columbia	296,131	6.50%	$ 32.05		$ 6,489.85
Florida	8,653,670	3.80%	$ 17.68	$ 25.61	$ 108,763.94
Georgia	4,588,023	5.30%	$ 18.80	$ 24.02	$ 34,848.07
Hawaii	634,613	2.80%	$ 17.48		$ 9,026.64
Idaho	738,739	3.80%	$ 14.83	$ 13.45	$ 4,021.22
Illinois	6,469,338	5.70%	$ 21.03	$ 29.10	$ 102,304.28
Indiana	3,208,969	5.40%	$ 17.03	$ 28.10	$ 29,582.90
Iowa	1,659,800	4.60%	$ 15.90	$ 17.99	$ 11,335.38
Kansas	1,475,791	5.10%	$ 16.28	$ 18.60	$ 16,121.81
Kentucky	1,999,658	6.10%	$ 16.33	$ 18.73	$ 29,142.75
Louisiana	2,071,486	7.10%	$ 16.13	$ 19.15	$ 22,165.49
Maine	711,885	4.80%	$ 15.73	$ 16.83	$ 6,919.50
Maryland	2,935,064	4.10%	$ 21.33	$ 28.72	$ 27,794.72
Massachusetts	3,364,496	4.80%	$ 24.08	$ 31.57	$ 72,898.06
Michigan	5,097,457	6.70%	$ 19.83	$ 26.18	$ 57,609.25
Minnesota	2,947,198	4.00%	$ 19.63	$ 28.37	$ 33,670.18
Mississippi	1,343,287	7.90%	$ 14.30	$ 12.76	$ 10,188.90
Missouri	3,024,478	5.40%	$ 17.28	$ 26.63	$ 30,408.27
Montana	493,407	4.00%	$ 14.03	$ 18.88	$ 4,296.86
Nebraska	986,296	3.80%	$ 15.60	$ 23.15	$ 8,828.87
Nevada	1,215,957	4.10%	$ 18.63	$ 19.70	$ 17,851.10
New Hampshire	732,036	3.60%	$ 19.50	$ 24.57	$ 8,134.54
New Jersey	4,430,373	4.40%	$ 23.78	$ 33.91	$ 64,272.31
New Mexico	935,888	5.30%	$ 15.68	$ 20.14	$ 9,724.37
New York	9,415,861	5.00%	$ 24.98	$ 25.89	$ 219,357.95
North Carolina	4,332,710	5.20%	$ 17.28	$ 24.58	$ 37,973.46
North Dakota	358,960	3.40%	$ 14.40	$ 18.03	$ 3,142.74
Ohio	5,900,354	5.90%	$ 17.95	$ 25.86	$ 57,898.26
Oklahoma	1,741,753	4.40%	$ 15.25	$ 18.94	$ 13,265.35
Oregon	1,860,104	6.10%	$ 17.60	$ 24.99	$ 24,753.17
Pennsylvania	6,292,282	5.00%	$ 19.05	$ 24.33	$ 96,374.06
Rhode Island	569,451	5.00%	$ 18.63	$ 19.42	$ 8,236.72
South Carolina	2,080,517	6.80%	$ 15.83	$ 26.13	$ 25,939.79
South Dakota	432,032	3.90%	$ 14.03	$ 17.13	$ 3,848.58
Tennessee	2,909,562	5.60%	$ 17.23	$ 22.66	$ 24,320.04
Texas	11,225,882	5.30%	$ 19.30	$ 26.09	$ 146,009.32
Utah	1,268,075	4.30%	$ 16.03	$ 20.85	$ 14,265.15
Vermont	355,897	3.50%	$ 16.45	$ 17.85	$ 3,327.14
Virginia	3,933,949	3.50%	$ 20.33	$ 27.01	$ 40,005.87
Washington	3,292,195	5.50%	$ 19.58	$ 26.18	$ 50,370.18
West Virginia	800,383	5.00%	$ 15.08	$ 16.83	$ 8,213.95
Wisconsin	3,041,470	4.70%	$ 17.05	$ 21.95	$ 35,272.40
Wyoming	284,538	3.60%	$ 15.88	$ 16.60	$ 1,835.35

	US Censu	Bureau of Educa	National Center for Edu	Tax Foundation	ProximityOne
State	Public Debt %(GSP)	Percentage of Population with Bachelors	NAEP Reading Scores (Literacy Proxy) *Higher the better	Corporate Tax Index *higher the better	Distance from MA (mi)
Alabama	14%	22.3	251.98	5.78	1,230
Alaska	22%	25.5	258.72	6.99	3,900
Arizona	14%	28	254.79	5.14	2,670
Arkansas	12%	18.8	257.69	4.64	1,440
California	17%	31.7	250.43	4.67	3,020
Colorado	16%	35.5	264.76	5.85	2,000
Connecticut	16%	34.5	264.01	4.68	110
Delaware	11%	26.9	266.01	6.29	378
District of Columbia	8%	45.7	238.2	4.41	450
Florida	16%	26	255.78	6.85	1,160
Georgia	10%	27.6	256.87	5.35	1,110
Hawaii	17%	26.6	248.51	4.87	4,600
Idaho	9%	23.8	264.3	5.17	2,690
Illinois	18%	27.4	263.52	5.19	1,000
Indiana	12%	21.1	261.01	6.01	930
Iowa	10%	24.3	267	4.63	1,340
Kansas	15%	30	266.83	4.84	1,440
Kentucky	21%	21	263.94	4.57	960
Louisiana	13%	22.4	252.69	4.76	1,510
Maine	15%	24.2	269.98	4.48	270
Maryland	11%	35.2	260.78	5.22	430
Massachusetts	22%	36.7	273.72	5.06	-
Michigan	15%	24.4	261.14	5.12	800
Minnesota	14%	32.5	268.36	4.7	1,390
Mississippi	13%	20.1	250.53	5.05	1,460
Missouri	14%	28.1	264.66	5.42	1,210
Montana	14%	25.5	269.22	6.14	2,200
Nebraska	13%	24.8	267.46	4.58	1,470
Nevada	16%	24.5	252.87	6.84	2,750
New Hampshire	15%	35.4	269.65	6.58	71
New Jersey	15%	34.6	269.42	3.96	273
New Mexico	14%	25.1	251.03	5.06	2,220
New York	23%	30.6	265.14	3.91	170
North Carolina	11%	23.4	258.16	4.74	850
North Dakota	13%	25.2	270.24	4.99	1,650
Ohio	13%	24.6	266.77	4.11	660
Oklahoma	11%	22.9	259.63	5.48	1,690
Oregon	17%	25.9	263.16	6.08	3,140
Pennsylvania	20%	25.3	266.82	5.49	320
Rhode Island	19%	27.2	260.97	4.11	50
South Carolina	19%	24.9	257.17	5.03	940
South Dakota	12%	25.5	268.54	7.38	1,910
Tennessee	11%	24.3	259.07	5.6	1,340
Texas	15%	24.5	258.19	6.56	1,750
Utah	16%	30.8	261.87	5.45	2,380
Vermont	14%	34.2	268.77	4.34	180
Virginia	11%	33.1	267.81	5.45	580
Washington	19%	29.9	264.66	5.84	3,020
West Virginia	15%	15.3	255.07	4.77	750
Wisconsin	16%	25.6	266.23	4.92	1,090
Wyoming	7%	22.5	268.12	7.47	1,920

Appendix B - Congressional Representation Data

State	Electoral Votes	House		Senate		Total From State	Percent Representation
		Armed Services[1]	Appropriations[2]	Armed Services[3]	Appropriations[4]		
Hawaii	4	1		1	1	3	75%
Nevada	4	1		1	1	3	75%
New Hampshire	4	1			1	2	50%
South Carolina	8	2		1		3	38%
Missouri	11	2		1	1	4	36%
Alabama	9	1		1	1	3	33%
Kansas	6	1	1			2	33%
Montana	3				1	1	33%
North Dakota	3				1	1	33%
Vermont	3				1	1	33%
Mississippi	7		1		1	2	29%
North Carolina	14	3		1		4	29%
Oklahoma	8	1		1		2	25%
Rhode Island	4			1		1	25%
Pennsylvania	23	3	1		1	5	22%
Maryland	10	1			1	2	20%
Minnesota	10		1	1		2	20%
New Mexico	5				1	1	20%
West Virginia	5				1	1	20%
Ohio	21	2	2			4	19%
Washington	11	1	1			2	18%
Arkansas	6				1	1	17%
Indiana	12		1	1		2	17%
Texas	32	2	2		1	5	16%
Georgia	13		1	1		2	15%
Virginia	13	1	1			2	15%
Iowa	7				1	1	14%
Illinois	22	1	1		1	3	14%
New Jersey	15	1	1			2	13%
Arizona	8			1		1	13%
Connecticut	8			1		1	13%
Kentucky	8				1	1	13%
Florida	25	1	1	1		3	12%
Tennessee	11	1				1	9%
New York	33	1		1		2	6%
California	54	2			1	3	6%
Totals		**30**	**15**	**15**	**19**		

[1] Representatives from the House Armed Services Committee/Tactical Air and Land Forces Subcommittee

[2] Representatives from the House Appropriations Committee/Defense Subcommittee

[3] Representatives from the Senate Armed Services Committee/Subcommittee on AirLand

[4] Representatives from the Senate Appropriations Committee/Subcommittee on Defense

Appendix C - BOM Relational Exploration Source Code

Language: C/C++
Compiled with Microsoft VS.NET C/C++ compiler
Execution: Apache CGI

Source Code:

```
// IndentedBOMAnalysis.cpp : Defines the entry point for the console application.
//

#define WIN32_LEAN_AND_MEAN      // Exclude rarely-used stuff from Windows headers
#define ROOT_PART_NO -1
#include <stdio.h>
#include <tchar.h>
#include <math.h>
#include <map>
#include <vector>
#include <winsock2.h>
#include <Ws2tcpip.h>
#include <io.h>
#include <time.h>
#include <string.h>

using namespace std;

//HTTP location of product documentation
char *ReferenceDocsRoot = "prodfiles.hq.irobot.com";
//Directory location at root where documents are stored
char *ReferenceDocsDir = "releases/";
char *FileName;
char *PartNoStr;
char *FramesStr;
char *ReferringPartStr;

//Reads a Delimited entry from a line delimited by a character delimiter
//Specifically designed for Excel documents saved as text files
//Output is the delimited entry
//Input is a string of characters
//Position indicates which entry to retreive indexed from 0
//Max size is the maximum number of bytes to copy from input to output
int ReadDelimitedEntry(char *output, const char *input, const char delimiter,
                       unsigned int position, unsigned int MaxSize) {
  unsigned int CharactersCopied = 0;
  int CurrentPosition = 0;
  const char *p = input;
  bool inQuotes = false; //Indicates if the search for delimiter is inside quotes
  if (delimiter == '\"') { printf("Unable to delimit with \"\n"); return 0; }
  while (*p) {  //While not end of line
    if (*p == '\"') { //If double quotes then not in quotes, just specifying a quote
      if (*(p+1) != '\"')
        inQuotes = !inQuotes;
      p++;
    }

    if ((*p == delimiter) && (!inQuotes))
    {
      if (CurrentPosition == position) {
        output[CharactersCopied] = 0;
        return CharactersCopied;
      }
      else
        CurrentPosition++;
      p++;
      continue;
```

```
    }

    if (CurrentPosition == position) {
      output[CharactersCopied++] = *p;
      if (CharactersCopied >= (MaxSize-1)) {
        output[CharactersCopied] = 0;
        return CharactersCopied;
      }
    }

    p++;
  }
  output[CharactersCopied] = 0;
  return CharactersCopied;
}

typedef struct {
  int partNumber; //OEM defined part number
  int itemSequence; //Assembly order
  char unitOfMeasure[20];
  float quantity; //Number included in parent
} BOMMaterialEntry;

typedef struct {
  int partNumber;
  char description[80];
  char revision[10];
  char type[30];
  vector<BOMMaterialEntry> material; //List of material entries
} BOMEntry;

FILE *fil;
map<int, BOMEntry> BOMEntries; //BOM entries indexed by Part number
typedef map<int, BOMEntry>::iterator BOMIterator;
char line[5000];

//Returns the part number of the item added
//Sets up the material field of the BOMEntry
//Returns when next level is less than or equal to level or EOF
//Function adds an item to the list of items.
int AddItem(unsigned int level, BOMEntry &myEntry) {
  BOMEntry newEntry;
  myEntry.material.clear();
  BOMMaterialEntry myMaterial;
  char forint[15];

  unsigned int nextLevel;
  long filePos;

  while (!feof(fil)) {
    filePos = ftell(fil);
    line[0] = 0;
    fgets(line, 4999, fil);
    if (feof(fil) && (line[0] == 0)) break;
    sscanf(line, "%u\t%d", &nextLevel, &newEntry.partNumber);
    ReadDelimitedEntry(newEntry.description, line, '\t', 2, 80);
    ReadDelimitedEntry(newEntry.revision, line, '\t', 3, 10);
    ReadDelimitedEntry(newEntry.type, line, '\t', 4, 30);
    ReadDelimitedEntry(forint, line, '\t', 7, 15);
    myMaterial.itemSequence = atoi(forint);
    ReadDelimitedEntry(myMaterial.unitOfMeasure, line, '\t', 12, 20);
    ReadDelimitedEntry(forint, line, '\t', 13, 15);
    myMaterial.quantity = (float)atof(forint);

    if (nextLevel <= level) {
      //Restore position to that before reading line.
      fseek(fil, filePos, SEEK_SET);
      break;
    }

    myMaterial.partNumber = AddItem(nextLevel, newEntry);
```

```
    BOMEntries[newEntry.partNumber] = newEntry;
    myEntry.material.push_back(myMaterial);
  }

  return myEntry.partNumber;
}

//Prints the HTML part link according to the HTTP setup
void printPartLink(int partNo, char *text, int referringPart) {
  printf("<a target=\"_parent\" href=\"IndentedBOMAnalysis.exe?part=%d?frames=y"
         "?bom=%s\">%s</a>", partNo, FileName, text);
}

typedef struct {
  char Location[300];
  char Name[100];
} DocumentStore;

//Function returns the documents associated with the particular part no
//Documents are stored in an HTML directory.  This function queries the html
//server for the content of the directory associated with part PARTNO and
//returns the name and location of each of the documents in the directory
vector<DocumentStore> GetDocuments(int partno,
                                   const char *revision,
                                   bool CheckObsolete = 0) {

  vector<DocumentStore> documents;

  WSADATA wsaData; //Socket
  WSAStartup( 0x0101, &wsaData );

  int sockfd, portno, n;
  struct sockaddr_in serv_addr;
  struct hostent *server;

  portno = 80; //Default
  sockfd = (int)socket(AF_INET, SOCK_STREAM, 0);
  if (sockfd < 0) {
        printf("ERROR opening socket");
    return documents;
  }
  server = gethostbyname(ReferenceDocsRoot);
  if (server == NULL) {
      printf("ERROR, no such host\n");
      return documents;
  }
  memset((char *) &serv_addr, 0, sizeof(serv_addr));
  serv_addr.sin_family = AF_INET;
  memcpy((char *)&serv_addr.sin_addr.s_addr,
    (char *)server->h_addr,
    server->h_length);
  serv_addr.sin_port = htons(portno);
  if (connect(sockfd,(sockaddr*)&serv_addr,sizeof(serv_addr)) < 0) {
    printf("Unable to connect to server\n");
        return documents;
  }

  char buffer[3000];
  char DirectoryLocation[100];

  if (CheckObsolete) //Check to see if the part is obsolete, as defined by the directory
name
    sprintf(DirectoryLocation, "%s%u-OBSOLETE/%u-", ReferenceDocsDir, abs(partno),
abs(partno));
  else
    sprintf(DirectoryLocation, "%s%u/%u-", ReferenceDocsDir, abs(partno), abs(partno));
  if (!strcmp(itoa(atoi(revision), line, 10), revision)) //is it a number?
    sprintf(DirectoryLocation+strlen(DirectoryLocation), "%02d", atoi(revision));
  else
    sprintf(DirectoryLocation+strlen(DirectoryLocation), "%s", revision);
```

```
  //Formulate the HTTP request
    sprintf(buffer, "GET /%s/ Http1.1\n"
      "Host: prodfiles.hq.irobot.com\n"
            "Accept: text/html\n"
      "Accept-Language: En\n"
      "Connection: Keep-Alive\n\n", DirectoryLocation);
  //Send the directory listing request
  n = send(sockfd,buffer,(unsigned int)strlen(buffer), 0);
  if (n < 0) {
    printf("ERROR writing to socket: %d", WSAGetLastError());
    return documents;
  }

  memset(buffer,0, 3000);
  //Get the HTTP directory response
  n = recv(sockfd, buffer, 3000, 0);
  if (n < 0) {
    printf("ERROR reading from socket");
    return documents;
  }

  //Look through the response for href's to documents
  char *seeker = strstr(buffer, "alt=\"[   ]\"> <a href=");
  while (seeker) {

    char *htmlFileName = strstr(seeker, "<a href=\"") + 9;
    char *FileName = strstr(htmlFileName, "\">") + 2;
    seeker = strstr(FileName, "alt=\"[   ]\">");

    //Format for writing
    *strstr(htmlFileName, "\">") = 0;
    *strstr(FileName, "</a>") = 0;

    DocumentStore ds;

    sprintf(ds.Location, "http://%s/%s/%s",
            ReferenceDocsRoot, DirectoryLocation, htmlFileName);

    sprintf(ds.Name, "%s", FileName);
   //Add the file to the documents list
    documents.push_back(ds);
  }
  close(sockfd);
  return documents;
}

//Returns the number of times this part appears in the BOM
float GetPartQuantity(int partNo) {
  //if it's the top level return 1.0 of them.
  if ((partNo == -1) || (partNo == -2) || (partNo == -3) || (partNo == -4)) return 1.0;
  float QuantitySum = 0.0;
  for (BOMIterator i=BOMEntries.begin(); i!= BOMEntries.end(); i++)
    for (unsigned int j=0;j<i->second.material.size();j++)
      if (i->second.material[j].partNumber == partNo)
        QuantitySum += GetPartQuantity(i->first) * i->second.material[j].quantity;
  return QuantitySum;
}

//Returns the quantity of material used for each part
float GetMaterialQuantity(int partNo) {

  float MaterialSum = 0.0;

  if (!BOMEntries[partNo].material.size())
    return 1.0;

  for (unsigned int i=0;i<BOMEntries[partNo].material.size(); i++)
    MaterialSum += GetMaterialQuantity(BOMEntries[partNo].material[i].partNumber) *
            (strcmp(BOMEntries[partNo].material[i].unitOfMeasure, "EA") ?
            (float)1.0 : BOMEntries[partNo].material[i].quantity);
```

```
  return MaterialSum;
}

int GetAssemblyStepCount(int partNo, int distinct = 0) {
  //Recursively determines the number of subassembly steps in a part
  //If the part does not have any sub-assemblies, then just return 1
  //otherwise count the subassemblies in each subassembly and multiply them
  //respectivly by the quantity.
  //If the part is not made of other parts it has 0 steps

  if (!BOMEntries[partNo].material.size()) return 0;

  unsigned int AssemblySteps = 1; //At least one for this item.

  for (unsigned int i=0;i<BOMEntries[partNo].material.size();i++) {
    if (BOMEntries[partNo].material[i].quantity == 0.0) continue;
    int count = GetAssemblyStepCount(BOMEntries[partNo].material[i].partNumber,
                                     distinct);
    if (!strcmp(BOMEntries[partNo].material[i].unitOfMeasure, "EA")) {
      float a = BOMEntries[partNo].material[i].quantity;

      if ((floor(a) == a) && (!distinct)) //Is it a whole number and not distinct request
        AssemblySteps += (int)a*count;
      else
        AssemblySteps += count;

    } else AssemblySteps += count;
  }

  return AssemblySteps;
}

int _tmain(int argc, _TCHAR* argv[])
{

  //Location of the indented BOM text files
  char *SourceFilePath = "..\\BOMs\\";
  char SourceFileName[500];

  //CGI Query String
  char *queryString = getenv("QUERY_STRING");

  //Decode the string for BOM name, Part Number, referring part for back stepping,
  //and whether to use frames (split screen)
  FileName = strstr(queryString, "bom=") + 4;
  PartNoStr = strstr(queryString, "part=") + 5;
  FramesStr = strstr(queryString, "frames=") + 7;
  ReferringPartStr = strstr(queryString, "referringpart=") + 14;
  char *ShowPartsStr = strstr(queryString, "showparts");

  //If an entry is not found, present an error message
  if ((FileName == (char*)4) || (PartNoStr == (char*)5) || (FramesStr == (char*)7)) {
    printf("Content-type: text/html\n\n<html><body><h1>Error decoding query string: "
           "<font color=#FF0000>%s</font><br><h1>Expected bom=\"name.txt\""
           "part=[part number] frames=[Y/N]</h1></body></html>\n", queryString);
    return 1;
  }

  //Remove the delimiter ?
  while (strrchr(queryString, '?'))
    *strrchr(queryString, '?') = 0;

  //Determine the source file name based on the query string input
  sprintf(SourceFileName, "%s%s", SourceFilePath, FileName);

  fil = fopen(SourceFileName, "r");
  if (!fil) {
    //If the file does not exist then return an error message
    printf("Content-type: text/html\n\n<html><body><h1>Error reading file: "
           "%s</body></html>\n", SourceFileName);
    return 1;
```

```
}

//Generate a Name and Entry for the root level item of BOM.  The Excell
//file does not contain this information.  If EOD, Scout, or Explorer
//is in the filename, then use the respective name in the root part
char BOMRootName[50];
sprintf(BOMRootName, "%s BOM Top Level",
  (strstr(FileName, "EOD") ? "EOD" :(strstr(FileName, "Scout") ? "Scout":"Explorer")));

fgets(line, 4999, fil); //Remove header line.
BOMEntry myEntry;
myEntry.partNumber = ROOT_PART_NO;
strcpy(myEntry.description, BOMRootName);
strcpy(myEntry.revision, "");
strcpy(myEntry.type, "");
AddItem(0, myEntry);
BOMEntries[myEntry.partNumber] = myEntry;

int partno;
partno = atoi(PartNoStr);

//Look the for the part request to be displayed
if (BOMEntries.find(partno) == BOMEntries.end()) {
  //If it's not present, then return an error message
  printf("Content-type: text/html\n\n<html><body><h1>Unknown part number: "
         "%d</h1></body></html>\n", partno);
  return 1;
}

//Get the document associated with the part
int Obsolete = 0;
vector<DocumentStore> docs = GetDocuments(partno, BOMEntries[partno].revision);
if (!docs.size()) { //Check for obsolete
  //If the Part does not have any documents then look for the documents in the
  //obsolete directory instead
  docs = GetDocuments(partno, BOMEntries[partno].revision, 1);
  if (docs.size()) Obsolete = 1;
}

//Send the HTML title
printf("Content-type: text/html\n\n<html> <title> (%u) %s rev. %s</title>",
    abs(partno), BOMEntries[partno].description, BOMEntries[partno].revision);

if ((*FramesStr == 'Y') || (*FramesStr == 'y')) {
  //If frames have been requested, the split the page into two
  //One with data and the pdf document if available
  int BOMFramePercent = 60;
  char *DOCSource = "/nopdf.html";
  for (unsigned int i=0;i<docs.size();i++)
    if (strstr(docs[i].Name, ".pdf")) {
      BOMFramePercent = 60;
      DOCSource = docs[i].Location;
    }

  //Setup the frames, one with document, the other with BOM data
  printf("<FRAMESET cols=\"%d%,%d%\">", BOMFramePercent, 100-BOMFramePercent);
  printf("<FRAME name=\"BOM\" src=\"IndentedBOMAnalysis.exe?part=%d?frames=n?bom=%s\">"
      "<frame name=\"DOCS\" src=\"%s\">", partno, FileName, DOCSource);

  printf("<body>");
}

//Check if this request is for the BOM data
if ((*FramesStr == 'n')||(*FramesStr == 'N')) {

  printf("<body>\n");
  //Display the header bar
  //Part  Revision                Description           Assembly Steps   Pieces
  //14931    D      Assy,RCV,No Brakes,PackBot Explorer      166/288       5114

 if (Obsolete) printf("<center><h1><font color=\"#FF0000\">This part is "
```

```
                    "obsolete</font></h1>--Request a new BOM--</center><br>\n");
printf("<center><table border = 0><tr><td>");
printf("<table border =  1><tr><td>Part</td><td>Revision </td>"
       "<td>Description </td><td>Assembly Steps</td><td>Pieces</td></tr>");
printf("<tr><td><h1>%u </h1></td><td><h1>%s </h1></td><td><h1>%s "
       "(%s)</h1></td><td><h1> %d/%d</h1></td><td><h1> %d</h1></td></tr>",
       abs(partno), BOMEntries[partno].revision, BOMEntries[partno].description,
       BOMEntries[partno].type, GetAssemblyStepCount(partno, 1),
       GetAssemblyStepCount(partno), (int)GetMaterialQuantity(partno));
printf("</table></td><td><img src=\"/%s.jpg\" height=128 width=115></td></tr>",
       (strstr(FileName, "EOD") ? "eod" : (strstr(FileName, "Scout")? "scout":
       (strstr(FileName, "Explorer")) ? "explorer" : "unknownbase")));
printf("</table></center><br>");

//Show the related documentation, set target to the frame on the right
printf("<center><h1>Related Documents</h1></center>\n");
if (!docs.size()) printf("None");
for (unsigned int i =0;i<docs.size(); i++)
  printf("<li><a target=\"DOCS\" href=\"%s\">%s</a></li>",
         docs[i].Location, docs[i].Name);

//Display the BOM elements comprising this part
printf("<center><h1>Bill of  Materials</h1></center>\n");
//Look up the sub parts  now
if (!BOMEntries[partno].material.size())
  printf("None");
else {
  printf("<table>\n");
  printf("<tr><td><h2>Part</h2></td><td><h2>Rev. </h2></td>"
         "<td><h2>Description</h2></td><td><h2>Part Type </h2></td>"
         "<td><h2>Quantity </h2></td><td><h2>Assembly Steps"
         " </h2></td><td><h2>Pieces</h2></td></tr>\n");
  For (unsigned int i=0;i<BOMEntries[partno].material.size();  i++) {
    printf("<tr><td>");
    printPartLink(BOMEntries[partno].material[i].partNumber,
                  itoa(abs(BOMEntries[partno].material[i].partNumber),
                  line,  10), partno);
    printf("</td>");

    printf("<td>%s</td>",
            BOMEntries[BOMEntries[partno].material[i].partNumber].revision);

    printf("<td>");
    printPartLink(BOMEntries[partno].material[i].partNumber,
                  BOMEntries[BOMEntries[partno].material[i].partNumber].description,
                  partno);
    printf("</td>");

    printf("<td>%s</td>",
           BOMEntries[BOMEntries[partno].material[i].partNumber].type);

    printf("<td>%s%0.2f %s%s</td>",  (BOMEntries[partno].material[i].quantity <= 0.0)
           ? "<font color=\"#FF0000\">":"",BOMEntries[partno].material[i].quantity,
           BOMEntries[partno].material[i].unitOfMeasure,
          (BOMEntries[partno].material[i].quantity <= 0.0) ? "</font>" : "");

    //Get the number of unique steps
    int DistinctAssembly =
           GetAssemblyStepCount(BOMEntries[partno].material[i].partNumber, 1);
    //Get the total number of steps (ie, if it contains 5 wheels where each wheel has
    //5 steps then there are 25 steps, but only 5 unique steps.
    int TotalAssembly =
           GetAssemblyStepCount(BOMEntries[partno].material[i].partNumber);
    if (DistinctAssembly != TotalAssembly)
      printf("<td>%d / %d</td>", DistinctAssembly, TotalAssembly);
    else
      if (TotalAssembly)
        printf("<td>%d</td>", TotalAssembly);
      else printf("<td> </td>");

    //Display the quantity of material
```

```
        printf("<td>%d</td>",
               (int)GetMaterialQuantity(BOMEntries[partno].material[i].partNumber));

        printf("</tr>");
      }
      printf("</table>");
    }

    if (partno != ROOT_PART_NO)  {

      //If the part is not the root part, then display other occurances
      printf("<center><h2>Part %u  is found in the following parts"
             "</h2></center>\n", abs(partno));
      int foundInPart = 0;
      for  (BOMIterator i=BOMEntries.begin();  i!=  BOMEntries.end(); i++)
        for  (unsigned int j=0;j<i->second.material.size();j++)
          if (partno == i->second.material[j].partNumber)  {
            foundInPart = 1;
            printf("<li>");
            printPartLink(i->second.partNumber,
                          itoa(abs(i->second.partNumber),  line,  10), partno);
            printf(" ");
            printPartLink(i->second.partNumber,  i->second.description, partno);
            printf("</li>\n");
          }
      if (!foundInPart) printf("None");

      int partQuantity = (int)GetPartQuantity(partno);
      printf("<center><h3>There %s %d of part %d in a complete system</h3></center>\n",
        (partQuantity == 1 ? "is" : "are"), partQuantity, abs(partno));
    }

  }

  //Display BOM explorer image
  printf("<br><center><a href=\"/BOMAnalysis.html\" target=\"_parent\">"
         "<img src=\"/BOMExplorer.JPG\" alt=\"Return to BOM Explorer home\""
         "border=\"0\"></a></center>\n");

  //Provide date of file extraction
  struct _finddata_t c_file;
  long hFile;
  hFile = (long)_findfirst( SourceFileName, &c_file );
  printf("<br><br><br><center>-- BOM pulled from Oracle on %s --</center>\n",
         ctime( &( c_file.time_write ) ));
  _findclose( hFile );

  //Display where the documents are taken from
  printf("<br><center>Documents are pulled from <a target=\"_parent\""
         " href=\"http://%s/%s\">%s/%s</a></center>", ReferenceDocsRoot,
         ReferenceDocsDir, ReferenceDocsRoot, ReferenceDocsDir);

  printf("</body></html>\n");

  fclose(fil);
  gets(line);
  return 0;
}
```

Appendix D - Manufacturing Simulation Tool Source Code

Language: C/C++
Compiled with Microsoft VS.NET C/C++ compiler
Execution: Apache CGI

Source Code:

```
// Manufacturing Environment Model.cpp : Defines the entry point for the application.
//

#include "stdafx.h"

using namespace std;


#define iterations 50000
int _tmain(int argc, _TCHAR* argv[])
{
   Model mod;
   srand(time(NULL));
   mod.RunLength = GetTicks(300, 0, 0, 0);

   map<PARTID, StockSupplier*> StockSuppliers;
   map<PARTID, Supplier*> Suppliers;

   Suppliers[1] = new Supplier();
   …
   Suppliers[13] = new Supplier();

   StockSuppliers[14] = new StockSupplier(14, 35, 0.02);
   …
   StockSuppliers[22] = new StockSupplier(22, 0.5, 0.02);

   strcpy(Suppliers[1]->SupplierName, "Lead CM");
   strcpy(Suppliers[1]->PartName, "SUGV");
   strcpy(Suppliers[2]->SupplierName, "Chassis Assembler");
   strcpy(Suppliers[2]->PartName, "Chassis");
   strcpy(Suppliers[3]->SupplierName, "Head and Neck Assembler");
   strcpy(Suppliers[3]->PartName, "Head and Neck");
   strcpy(Suppliers[4]->SupplierName, "Battery Supplier");
   strcpy(Suppliers[4]->PartName, "Battery");
   strcpy(Suppliers[5]->SupplierName, "Flipper Assembler");
   strcpy(Suppliers[5]->PartName, "Flipper Set");
   strcpy(Suppliers[6]->SupplierName, "Chassis Track Supplier");
   strcpy(Suppliers[6]->PartName, "Chassis Track");
   strcpy(Suppliers[7]->SupplierName, "Side Panel Supplier");
   strcpy(Suppliers[7]->PartName, "Side Panel");
   strcpy(Suppliers[8]->SupplierName, "Electronics Box Supplier");
   strcpy(Suppliers[8]->PartName, "Electronics Box");
   strcpy(Suppliers[9]->SupplierName, "Head Supplier");
   strcpy(Suppliers[9]->PartName, "Head");
   strcpy(Suppliers[10]->SupplierName, "Neck Supplier");
   strcpy(Suppliers[10]->PartName, "Neck");
   strcpy(Suppliers[11]->SupplierName, "Flipper Guide Supplier");
   strcpy(Suppliers[11]->PartName, "Flipper Guide");
   strcpy(Suppliers[12]->SupplierName, "Flipper Hub Supplier");
   strcpy(Suppliers[12]->PartName, "Flipper Hub");
   strcpy(Suppliers[13]->SupplierName, "Flipper Track Supplier");
   strcpy(Suppliers[13]->PartName, "Flipper Track");

   for (map<PARTID, Supplier*>::iterator i = Suppliers.begin(); i != Suppliers.end();
i++)
      i->second->ID = i->first;
```

```
   //Material Name, Cost, Failure Rate
   //Quantity per, Transport Time mean & std,
       //Shipping Fixed, variable, storage cost, initial ROP, EOQ
   Suppliers[1]->AddRMI(2, 1, Suppliers[2], GetTicks(4), 0, 110, 30, 300.0 /
(float)GetTicks(365), 0.01, GetTicks(4), 5, 5);
   Suppliers[1]->AddRMI(3, 1, Suppliers[3], GetTicks(4), 0, 110, 30, 300.0 /
(float)GetTicks(365), 0.01, GetTicks(4), 5, 5);
   Suppliers[1]->AddRMI(4, 4, Suppliers[4], GetTicks(4), 0, 40, 10, 50.0 /
(float)GetTicks(365), 0.04, GetTicks(4), 20, 15);
   Suppliers[1]->AddRMI(5, 1, Suppliers[5], GetTicks(4), 0, 40, 10, 200.0 /
(float)GetTicks(365), 0.01, GetTicks(4), 5, 5);
   Suppliers[2]->AddRMI(6, 2, Suppliers[6], GetTicks(2), 0, 10, 10, 100.0 /
(float)GetTicks(365), 0.01, GetTicks(4), 15, 5);
   Suppliers[2]->AddRMI(7, 1, Suppliers[7], GetTicks(2), 0, 50, 30, 300.0 /
(float)GetTicks(365), 0.01, GetTicks(4), 15, 5);
   Suppliers[2]->AddRMI(8, 1, Suppliers[8], GetTicks(2), 0, 110, 50, 300.0 /
(float)GetTicks(365), 0.01, GetTicks(4), 15, 5);
   Suppliers[3]->AddRMI(9, 1, Suppliers[9],   GetTicks(4), 0, 50, 15, 300.0 /
(float)GetTicks(365), 0.01, GetTicks(4), 15, 5);
   Suppliers[3]->AddRMI(10, 1, Suppliers[10], GetTicks(4), 0, 30, 20, 300.0 /
(float)GetTicks(365), 0.01, GetTicks(4), 15, 5);
   Suppliers[4]->AddRMI(14, 1, StockSuppliers[14], GetTicks(4), 0, 0, 30, 30.0 /
(float)GetTicks(365), 0.01, GetTicks(4), 50, 50);
   Suppliers[5]->AddRMI(11, 1, Suppliers[11], GetTicks(4), 0, 30, 30, 30.0 /
(float)GetTicks(365), 0.01, GetTicks(4), 15, 5);
   Suppliers[5]->AddRMI(12, 1, Suppliers[12], GetTicks(4), 0, 25, 30, 30.0 /
(float)GetTicks(365), 0.01, GetTicks(4), 15, 5);
   Suppliers[5]->AddRMI(13, 1, Suppliers[13], GetTicks(4), 0, 25, 30, 30.0 /
(float)GetTicks(365), 0.01, GetTicks(4), 15, 5);
   Suppliers[6]->AddRMI(15, 1, StockSuppliers[15], GetTicks(4), 0, 25, 30, 30.0 /
(float)GetTicks(365), 0.01, GetTicks(4), 50, 50);
   Suppliers[7]->AddRMI(16, 1, StockSuppliers[16], GetTicks(4), 0, 25, 30, 30.0 /
(float)GetTicks(365), 0.01, GetTicks(4), 50, 50);
   Suppliers[8]->AddRMI(17, 1, StockSuppliers[17], GetTicks(4), 0, 25, 30, 30.0 /
(float)GetTicks(365), 0.01, GetTicks(4), 50, 50);
   Suppliers[9]->AddRMI(18, 1, StockSuppliers[18], GetTicks(4), 0, 25, 30, 30.0 /
(float)GetTicks(365), 0.01, GetTicks(4), 50, 50);
   Suppliers[10]->AddRMI(19, 1, StockSuppliers[19], GetTicks(4), 0, 25, 30, 30.0 /
(float)GetTicks(365), 0.01, GetTicks(4), 50, 50);
   Suppliers[11]->AddRMI(20, 1, StockSuppliers[20], GetTicks(4), 0, 25, 30, 30.0 /
(float)GetTicks(365), 0.01, GetTicks(4), 50, 50);
   Suppliers[12]->AddRMI(21, 1, StockSuppliers[21], GetTicks(4), 0, 25, 30, 30.0 /
(float)GetTicks(365), 0.01, GetTicks(4), 50, 50);
   //Suppliers[13]->AddRMI(22, 1, StockSuppliers[22], GetTicks(4), 0, 25, 30, 30.0 /
(float)GetTicks(365), 0.01, GetTicks(4), 50, 50);
   Suppliers[13]->AddRMI(15, 1, StockSuppliers[15], GetTicks(4), 0, 25, 30, 30.0 /
(float)GetTicks(365), 0.01, GetTicks(4), 50, 50);

   Suppliers[1]->ContractedDeliveryWindow = GetTicks(30);
   …
   Suppliers[13]->ContractedDeliveryWindow = GetTicks(5);

   Suppliers[1]->Line.BatchSize = 1;
   …
   Suppliers[13]->Line.BatchSize = 30;

   for (map<PARTID, Supplier*>::iterator i = Suppliers.begin(); i != Suppliers.end();
i++) {
      i->second->Line.RunIncompleteBatches = true;
      i->second->Cash = 5000000.0;
      i->second->OperatingLevelTimeConsideration = TICKS(DAY*250);
      i->second->EOQPolicyReviewTime = TICKS(DAY*15);
      i->second->ROPPolicyReviewTime = TICKS(DAY*15);
      i->second->FGISafetyLevelReviewTime = TICKS(DAY*15);
      i->second->FGISafetyLevel = 35;
      i->second->Line.InvestedQuality = 0.0;
      i->second->Line.mySupplier = i->second;
      i->second->TrackingOn();
   }

   Suppliers[1]->Line.SchedulingDelayMean = GetTicks(0, 0, 0, 0);
```

```
…
Suppliers[13]->Line.SchedulingDelayMean = GetTicks(0, 10, 0, 0);

Suppliers[1]->Line.SchedulingDelayStd = GetTicks(0, 0, 0, 0);
…
Suppliers[13]->Line.SchedulingDelayStd = GetTicks(0, 2, 0, 0);

Suppliers[1]->Line.TestCoverage = 0.95;
…
Suppliers[13]->Line.TestCoverage = 0.95;

Suppliers[1]->Line.ManufacturingQuality = 0.95;
…
Suppliers[13]->Line.ManufacturingQuality = 0.90;

Suppliers[1]->Line.DesignQuality = 0.99;
…
Suppliers[13]->Line.DesignQuality = 0.99;

Suppliers[1]->Line.Line.FlowTimeMean = GetTicks(0, 12, 0);
…
Suppliers[13]->Line.Line.FlowTimeMean = GetTicks(0, 1, 0);

Suppliers[1]->Line.Line.FlowTimeStd = GetTicks(0, 4, 0);
…
Suppliers[13]->Line.Line.FlowTimeStd = GetTicks(0, 0, 10);

Suppliers[1]->Line.Line.TimeBetweenStarts = GetTicks(0, 3, 0);
…
Suppliers[13]->Line.Line.TimeBetweenStarts = GetTicks(0, 0, 30);

Suppliers[1]->Line.PartsInRework.FlowTimeMean = GetTicks(0, 36, 0);
…
Suppliers[13]->Line.PartsInRework.FlowTimeMean = GetTicks(0, 0, 2);

Suppliers[1]->Line.PartsInRework.FlowTimeStd = GetTicks(0, 40, 0);
…
Suppliers[13]->Line.PartsInRework.FlowTimeStd = GetTicks(0, 0, 5);

Suppliers[1]->WACC = ( 0.15/ 365.0) / (float)TicksPerDay;
…
Suppliers[13]->WACC = ( 0.08/ 365.0) / (float)TicksPerDay;

Suppliers[1]->Margin = 0.25;
…
Suppliers[13]->Margin = 0.08;

Suppliers[1]->FGIPerTickStorageCost = (500.0/ 365.0) / (float)DAY;
…
Suppliers[13]->FGIPerTickStorageCost = (10.0/ 365.0) / (float)DAY;

Suppliers[1]->ReplacementPartCost = 500;
…
Suppliers[13]->ReplacementPartCost = 30;

Suppliers[1]->LateDeliveryPenalty = 300;
…
Suppliers[13]->LateDeliveryPenalty = 10;

Suppliers[1]->LaborRate = 24.0 / (float)HOUR;
…
Suppliers[13]->LaborRate = 13.0 / (float)HOUR;

Suppliers[1]->EquipmentRate = 30.0 / (float)HOUR;
…
Suppliers[13]->EquipmentRate = 90.0 / (float)HOUR;

Suppliers[1]->FixedBuildLabor = 0 * HOUR;
…
Suppliers[13]->FixedBuildLabor = 0 * HOUR;
```

```
   Suppliers[1]->VariableBuildLabor = 12 * HOUR;
   …
   Suppliers[13]->VariableBuildLabor = 0.3 * HOUR;

   Suppliers[1]->FixedBuildEquipment = 0 * HOUR;
   …
   Suppliers[13]->FixedBuildEquipment = 0 * HOUR;

   Suppliers[1]->VariableBuildEquipment = 6 * HOUR;
   …
   Suppliers[13]->VariableBuildEquipment = 0.1 * HOUR;

   Suppliers[1]->TeardownAndRebuildLabor = 24 * HOUR;
   …
   Suppliers[13]->TeardownAndRebuildLabor = 0.1 * HOUR;

   Suppliers[1]->TeardownAndRebuildEquipment = 3 * HOUR;
   …
   Suppliers[13]->TeardownAndRebuildEquipment = 0.2 * HOUR;

   OEM oem(GetTicks(30), GetTicks(2), 3);

   for (map<PARTID, Supplier*>::iterator i = Suppliers.begin(); i != Suppliers.end();
i++)
      oem.Suppliers[i->first] = i->second;

   oem.Demand[1] = 64;
   oem.DemandIncrease[1] = 0.267;
   …
   oem.Demand[13] = 0;
   oem.DemandIncrease[13] = 13.87;

   for (map<PARTID, Supplier*>::iterator i = Suppliers.begin(); i != Suppliers.end();
i++)
      mod.AddModelComponent(i->second);
   for (map<PARTID, StockSupplier*>::iterator i = StockSuppliers.begin(); i !=
StockSuppliers.end(); i++)
      mod.AddModelComponent(i->second);

   mod.AddModelComponent(&oem);

   do {

      if (!(CurrentTick % TicksPerDay))
         printf("Simulating... Ticks: %d, day %d, %02d:%02d\r", CurrentTick,
GetDay(CurrentTick), GetHour(CurrentTick), GetMinute(CurrentTick));

   } while (mod++);


   printf("Saving tracking information to disk...");
   fflush(stdout);

   char FileName[54];
   for (map<PARTID, Supplier*>::iterator i = Suppliers.begin(); i != Suppliers.end();
i++) {
      sprintf(FileName, "%s.txt", i->second->SupplierName);
      i->second->SaveReport(FileName, '\t');
   }
   oem.SaveOrderReport("OEM.txt", '\t');
   printf("done\n");

   printf("(%d) Simulation ended at day %d, %02d:%02d\n"/*:%02d\n"*/, CurrentTick,
GetDay(CurrentTick), GetHour(CurrentTick), GetMinute(CurrentTick),
GetSecond(CurrentTick));

   oem.PrintOrderReport();

   for (map<PARTID, StockSupplier*>::iterator i = StockSuppliers.begin(); i !=
StockSuppliers.end(); i++)
      delete i->second;
```

```
   for (map<PARTID, Supplier*>::iterator i = Suppliers.begin(); i != Suppliers.end();
i++)
      delete i->second;

   char tmp[30];
   gets(tmp);


   return 0;
}
```

```
// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently
//

#pragma once

#define WIN32_LEAN_AND_MEAN          // Exclude rarely-used stuff from Windows headers
#include <stdio.h>
#include <tchar.h>
#include "math.h"
#include "time.h"
#include <stdlib.h>
#include "StatisticBox.h"
#include "ModelTypes.h"
#include "../mySQLTest/myString.h"
#include <stdarg.h>
#include <time.h>
#include <string.h>
#include <ctype.h>
```

```
//File: ModelTypes.h

#include <map>
#include <vector>

#define DefName(__Name) virtual string ComponentName() { return # __Name;}
#define UNLIMITED -1
#define HISTORICBUFFERSIZE 1000

using namespace std;

typedef int TICKS;
typedef unsigned int PARTID;
#define NOPART -1
typedef unsigned int ORDERID;
#define NOORDER -1

extern TICKS CurrentTick;
extern TICKS TicksPerDay;
extern ORDERID NextOrderID;

unsigned int GetDay(TICKS t);
unsigned int GetHour(TICKS t) ;
unsigned int GetMinute(TICKS t) ;
unsigned int GetSecond(TICKS t) ;
TICKS GetTicks(unsigned int Days, unsigned int Hours = 0, unsigned int Minutes = 0,
unsigned int Seconds = 0);

#define HOUR (GetTicks(0, 1))
#define DAY (TicksPerDay)
#define MINUTE (GetTicks(0, 0, 1))
#define SECOND (GetTicks(0, 0, 0, 1))

class ModelComponent {
public:
   virtual void operator++ () {};
   virtual void iterate() { this->operator++(); }
   DefName("ModelComponent");
};

class Model {
public:
   TICKS StartingTick;
   TICKS RunLength; //Number of ticks to run to
   vector <ModelComponent*> Components;
   Model () {RunLength = GetTicks(1); StartingTick=0;}
   void AddModelComponent(ModelComponent *mc) { Components.push_back(mc); }
   bool operator ++();
   void Run() { while (this->operator++()) ; }
};

class Part : public ModelComponent {
public:
   PARTID ID;
   TICKS CreatedTime;
   bool MfgFailure;
   unsigned int BadConstituentParts;
    vector<Part> SourceParts;

   DefName("Part");
};

class PartContainer : public ModelComponent {
public:
   DefName("PartContainer");
   vector <Part> Contents;
   bool CollectVolume;
   DynamicArray *Volume;
   unsigned int Capacity;
```

```
   PartContainer () { Contents.clear(); CollectVolume = false; Volume = NULL; Capacity =
UNLIMITED; }
   virtual void operator ++() ;
   virtual bool AddPart(Part part) { if ((Capacity == UNLIMITED) || (Capacity >
Contents.size())) { Contents.push_back(part); return true; } else return false; }
   virtual unsigned int PartCount() { return (unsigned int) Contents.size(); }
};

class Inventory : public PartContainer {
public:
   bool FIFO; //Indicates whether the Inventory buffer is a FIFO or LIFO
   Inventory() { PartContainer::PartContainer(); FIFO = true; }
   DefName("Inventory");
   Part RemovePart();
};

class Channel : public PartContainer  {
public:
   DefName("Channel");
   PartContainer *destination;
   TICKS FlowTimeMean, FlowTimeStd;
   TICKS TicksAtLastInsert;
   TICKS LastAssignedFlowTime;
   TICKS TimeBetweenStarts; //also known as Tact Time in serial processes

   vector <TICKS> TimeToDelivery; //Synced up with Contents on iteration "I hope"
   bool Queue; //Indicates if items added after others can be completed before the
others.

   Channel() { PartContainer::PartContainer(); TimeToDelivery.clear(); TicksAtLastInsert
= -1; LastAssignedFlowTime = 0; TimeBetweenStarts = 0; }
   Channel(PartContainer *Destination, bool queue, TICKS flowTimeMean, TICKS flowTimeStd
= 0) :
    destination(Destination), Queue(queue), FlowTimeMean(flowTimeMean),
          FlowTimeStd(flowTimeStd) { Channel(); }
   virtual bool SpaceAvailable(void);
   virtual bool AddPart(Part part);
   virtual void operator++();
};

class Supplier;

class ProductionLine : public ModelComponent {
   unsigned int RunLength; //Number of units left to produce in run
public:
   Channel Line; //Production Line

   Supplier *mySupplier;

   DynamicArray DemandHistory;

   unsigned int BatchSize;
   bool RunIncompleteBatches;

   TICKS SchedulingDelayMean, SchedulingDelayStd;
   TICKS TicksBeforeRun; //Number of ticks that must expire before the first item can be
processed

   Inventory TestArticles; //Articles ready for testing

   //Quality Variables for the Line
   float TestCoverage;

   float ManufacturingQuality;
   float InvestedQuality;
   float DesignQuality;

   Channel PartsInRework;  //Articles being reworked

   ProductionLine();
   void Run(unsigned int UnitsToProduce);
```

```
   unsigned int wip(); //returns the work in progress
   unsigned int PartsRemainingInRun() { return RunLength; }
   void operator++ ();
};

class SupplierInterface;

typedef struct {
   ORDERID OrderID;
   PARTID PartID;
   TICKS OrderedTime;
   bool BadPartReplacement;
   unsigned int Quantity;
   vector <TICKS> DeliveryTime;
   vector <unsigned int> DeliveryQuantities;
   unsigned int QuantityOutstanding;
   SupplierInterface *Customer;
} OrderRecord;

class SupplierInterface : public ModelComponent{
public:
   virtual void DeliverParts(ORDERID orderID, Inventory inv, bool BadPartReplacement) {}
//Deliver parts from orderID
   virtual void Order(ORDERID orderID, unsigned int UnitCount, SupplierInterface
*Customer, bool BadPartReplacement) {}
   virtual float Price() { return 0.0; }
};

class Supplier : public SupplierInterface {
   ORDERID GetOldestOutStandingOrder();

   bool ExternalChange;
   //Called by this supplier
   void DeliverFGI(unsigned int UnitCount);
   unsigned int RMIOnOrder(PARTID PartID);
   unsigned int RMIOrdered(PARTID PartID);
   unsigned int RMIReceived(PARTID PartID);
   float RMIFailureRate(PARTID PartID);
   float RMIDeliveryTime(PARTID PartID);
   float FGIUnitCost();
   void ReviewPolicies();
   float __Cost;

   TICKS EOQPolicyElapseTime, ROPPolicyElapseTime, FGISafetyLevelElapseTime;

public:
   PARTID ID;
   char SupplierName[50];
   char PartName[50];
   void OrderRMI(PARTID PartID, unsigned int UnitCount, bool BadPartReplacement = false);

   float DemandMean(TICKS PerTimeFrame, unsigned int TimeFramesToConsider);
   float DemandStd(TICKS PerTimeFrame, unsigned int TimeFramesToConsider);

   TICKS EOQPolicyReviewTime; //Time inbetween review of the EOQ policies
   TICKS ROPPolicyReviewTime; //Time inbetween review of the ROP policies
   TICKS FGISafetyLevelReviewTime; //Time inbetween review of the FGI safety policy

   map <PARTID, Inventory> RMI;
   map <PARTID, Channel> RMIDeliveryChannels;
   map <PARTID, SupplierInterface*> RMISuppliers;
   map <PARTID, unsigned int /*Part Count*/> PartComposition;
   Inventory FGI;
   Inventory BadParts;

   map <PARTID, DynamicArray> RMIDeliveryLeadTime;
   map <PARTID, DynamicArray> RMIFailureHistory;
   map <ORDERID, OrderRecord> OrdersReceived;
   map <ORDERID, OrderRecord> OrdersMade;

   ProductionLine Line;
```

```
   //Cost Data

   map <PARTID, float> RMIShipmentFixedCost;
   map <PARTID, float> RMIShipmentVariableCost;
   map <PARTID, float> RMIPerTickStorageCost;
   float LateDeliveryPenalty;  //Should cover the cost of shipment as it will not be
accounted by the receiver
   float FGIPerTickStorageCost;
   float WACC; //Weighted average cost of capital (Discount rate) per tick
   virtual float Price(); //Price charged for produced part
   float Margin; //Margin over cost needed for operations
   float InvestedQualityCostPerImprovement; //Cost per 1% improvement
   float ReplacementPartCost; //Cost of shipping a failed part
   float Cash; //Money to fund operations  (Not sure if we're going to use this)
   DynamicArray CashTrack;
   DynamicArray FGICostTrack;

   float LaborRate; //cost per TICK for labor
   TICKS FixedBuildLabor; //Amount of labor to build each batch
   TICKS VariableBuildLabor; //Amount of labor for each unit
   float EquipmentRate; //Cost per TICK for machinery
   TICKS FixedBuildEquipment; //Amount of equipment to build each batch
   TICKS VariableBuildEquipment; //Amount of equipment for each unit
   TICKS TeardownAndRebuildLabor; //Avergae labor to tear down and rebuild a failed part
   TICKS TeardownAndRebuildEquipment; //Avergae equipment usage to tear down and rebuild
a failed part

   TICKS ContractedDeliveryWindow;
   //Operating Levels
   TICKS OperatingLevelTimeConsideration;
   map <PARTID, unsigned int> RMIReorderPoint;
   map <PARTID, unsigned int> RMIReorderQuantities;
   unsigned int FGISafetyLevel;

   //Interfacing functions
   Supplier();
   Supplier(const Supplier&);

   //Called by Remote Suppliers
   virtual void DeliverParts(ORDERID orderID, Inventory inv, bool BadPartReplacement);
//Deliver parts from orderID
   virtual void Order(ORDERID orderID, unsigned int UnitCount, SupplierInterface
*Customer, bool BadPartReplacement);

   virtual unsigned int OutstandingOrders(void);
   virtual void operator++ ();
   void AddRMI(PARTID PartID, unsigned int UnitCount, SupplierInterface *Source, float
DeliveryTimeMean,
            float DeliveryTimeStd, float ShipmentFixedCost,
            float ShipmentVariableCost, float PerTickStorageCost, float
EstimatedFailureRate,
            TICKS EstimatedDeliveryLeadTime, unsigned int InitialReorderPoint, unsigned
int InitialReorderQuantity);
   void PrintReport(bool full = false);

   void TrackingOn();
   bool Tracking;
   void SaveReport(char *FileName, char Separator, TICKS AveragedTime = 1);
};

class StockSupplier : public SupplierInterface {
public:
   PARTID ID;
   float price;
   float PartFailureRate;
   StockSupplier(PARTID PartID, float Price, float FailureRate = 0.0) : price(Price),
ID(PartID), PartFailureRate(FailureRate) { }
   virtual void DeliverParts(ORDERID orderID, Inventory inv, bool BadPartReplacement) {
printf("Error: Parts delivered to StockSupplier\n"); /*Nobody should be deliverying parts
to this entity*/ }
```

```
   virtual void Order(ORDERID orderID, unsigned int UnitCount, SupplierInterface
*Customer, bool BadPartReplacement);
   virtual float Price() { return price; }
   void operator ++() {}
};

class OEM : public SupplierInterface {
   TICKS TicksUntilNextOrder;
   void OrderRMI(PARTID PartID, unsigned int Quantity, bool BadPartReplacement = false);
public:
   TICKS OrderCycleTime, OrderCycleTimeStd;
   unsigned int OrderQuantityStd;
   Inventory FGI;

   map<PARTID, Supplier*> Suppliers;
   map<PARTID, float> Demand;
   map<PARTID, float> DemandIncrease;

   map <ORDERID, OrderRecord> OrdersMade;

   OEM(TICKS orderCycleTime, TICKS orderCycleTimeStd, unsigned int orderQuantityStd);
   virtual void DeliverParts(ORDERID orderID, Inventory inv, bool BadPartReplacement);
   virtual void Order(ORDERID orderID, unsigned int UnitCount, SupplierInterface
*Customer, bool BadPartReplacement) { printf("Error: Parts requested from OEM\n");
/*Nobody should be deliverying parts to this entity*/}
   virtual float price() { return 0; }
   void operator ++();
   void SaveOrderReport(char *FileName, char Separator);
};
```

```
//File: ModelTypes.cpp

#include "stdafx.h"

TICKS CurrentTick = 0;
TICKS TicksPerDay = 24;
ORDERID NextOrderID = 1;

unsigned int GetDay(TICKS t) { return t / TicksPerDay; }
unsigned int GetHour(TICKS t) { return ((t % TicksPerDay) * 24) / TicksPerDay; }
unsigned int GetMinute(TICKS t) { if (TicksPerDay / (24) == 0) return 0; else return ((t
% (TicksPerDay / 24)) * 60*24) / TicksPerDay; }
unsigned int GetSecond(TICKS t) { if (TicksPerDay / (24*60) == 0) return 0; else return
((t % (TicksPerDay / (24*60))) * 60*60*24) / TicksPerDay; }

TICKS GetTicks(unsigned int Days, unsigned int Hours, unsigned int Minutes, unsigned int
Seconds) {
   TICKS ret;
   ret = Days*TicksPerDay;
   ret += Hours * TicksPerDay / 24;
   ret += Minutes * TicksPerDay / (24*60);
   ret += Seconds * TicksPerDay / (24*60*60);
   return ret;
}

bool Model::operator ++() {
   if (CurrentTick < StartingTick + RunLength) {
      for (unsigned int i=0; i<Components.size(); i++)
         (*Components[i])++;
      CurrentTick++;
   }
   return CurrentTick < StartingTick + RunLength;
}

void PartContainer::operator ++ () {

   if (CollectVolume) {
      if (Volume == NULL) Volume = new DynamicArray();
      Volume->AddData(PartCount());
   }
}

Part Inventory::RemovePart() {
   vector<Part>::iterator i;
   if (FIFO)
      i = Contents.begin();
   else
      i = Contents.end()--;
   Part p = *i;
   Contents.erase(i);
   return p;
}

bool Channel::SpaceAvailable(void) {
   if ((Capacity != UNLIMITED)   && (Contents.size() >= Capacity)) return false;
   if ((CurrentTick != TicksAtLastInsert) && (CurrentTick - TicksAtLastInsert <
TimeBetweenStarts)) return false;
   return true;
}

//Add a part to the channel and assign its time to delivery
bool Channel::AddPart(Part part) {
   if (!SpaceAvailable()) return false;
   Contents.push_back(part);
   if (CurrentTick != TicksAtLastInsert) {
      TICKS newFlowTime = abs((TICKS)normal(FlowTimeMean, FlowTimeStd));

      if ((Queue) && (LastAssignedFlowTime - (CurrentTick - TicksAtLastInsert) > 0) &&
(newFlowTime < (LastAssignedFlowTime - (CurrentTick - TicksAtLastInsert))))
         LastAssignedFlowTime -= (CurrentTick - TicksAtLastInsert);
      else
```

```
         LastAssignedFlowTime = newFlowTime;

      TicksAtLastInsert = CurrentTick;
   }
   TimeToDelivery.push_back(LastAssignedFlowTime);
   return true;
}

//Progress the Items Through the channel.  If the time arrives, add the part to the
destination parts container and remove the reference.
void Channel::operator++() {

   PartContainer::operator++();

   vector<TICKS>::iterator TimeToDeliveryIterator = TimeToDelivery.begin();
   vector<Part>::iterator ContentsIterator = Contents.begin();

   while (TimeToDeliveryIterator != TimeToDelivery.end()) {
      if (*TimeToDeliveryIterator > 0) (*TimeToDeliveryIterator)--;
      if ((*TimeToDeliveryIterator == 0) && (destination->AddPart(*ContentsIterator))) {
            ContentsIterator = Contents.erase(ContentsIterator);
            TimeToDeliveryIterator = TimeToDelivery.erase(TimeToDeliveryIterator);
      } else {
         TimeToDeliveryIterator++;
         ContentsIterator++;
      }
   }
}

ProductionLine::ProductionLine() {
   PartsInRework.destination = &TestArticles;
   Line.destination = &TestArticles;
   RunLength = 0;
   TicksBeforeRun = 0;
}

void ProductionLine::Run(unsigned int UnitsToProduce) {
   if ((!RunLength) && (!wip())) {
      TicksBeforeRun = normal(SchedulingDelayMean, SchedulingDelayStd);
      if (TicksBeforeRun < 0) TicksBeforeRun = 0;
   }
   RunLength+=UnitsToProduce;
}

unsigned int ProductionLine::wip() {
   return Line.PartCount() + PartsInRework.PartCount() + TestArticles.PartCount();
}

void ProductionLine::operator ++ () {
   if (mySupplier->Tracking) DemandHistory.AddData(RunLength);

   //Testing...
   Inventory BlockedPartsForRework;  //Holder for parts that are waiting on RMI to be
completed.
   while (TestArticles.PartCount()) {
      Part p = TestArticles.RemovePart();
      if (((p.MfgFailure) || (p.BadConstituentParts)) && (((float)rand() /
(float)RAND_MAX <= TestCoverage))) {
         //Failed part caught
         //If it failed due to Mfg then discard and order a new one
         //Otherwise look for first failed RMI and order a new one.  Place the part in
the Rework channel
         if (p.MfgFailure) {
            mySupplier->BadParts.AddPart(p);
            Run(1);
         } else { //Look through the part's source parts for the bad one.  when found
replace.
            for (unsigned int i=0;i<p.SourceParts.size();i++) {
               if ((p.SourceParts[i].MfgFailure) ||
(p.SourceParts[i].BadConstituentParts)) { //This is the bad part
                  mySupplier->OrderRMI(p.SourceParts[i].ID, 1, true); //Order a new part
```

```
                  //This assumes that no RMI sources have been added since the part was
created
                  //Assumes that the ordering of sources coincides with ordering of RMI[]
                  if (mySupplier->RMI[p.SourceParts[i].ID].PartCount()) {
                     mySupplier->BadParts.AddPart(p.SourceParts[i]);
                     mySupplier->RMIFailureHistory[p.SourceParts[i].ID].AddData(1);
                     p.SourceParts[i] = mySupplier-
>RMI[p.SourceParts[i].ID].RemovePart();
                     p.BadConstituentParts--;
                     PartsInRework.AddPart(p);
                  } else {
                     //Part is blocked
                     BlockedPartsForRework.AddPart(p);
                  }
                  break; //Stop looking for bad parts
               }
            }
         }
      } else {
         mySupplier->FGI.AddPart(p);

         //Track the RMI failure rates
         for (map <PARTID, unsigned int>::iterator i=mySupplier->PartComposition.begin();
i!= mySupplier->PartComposition.end(); i++)
            for (unsigned int x=0;x<i->second;x++)
               mySupplier->RMIFailureHistory[i->first].AddData(0);
      }
   }

   while (BlockedPartsForRework.PartCount())
      TestArticles.AddPart(BlockedPartsForRework.RemovePart());

   Line++;
   PartsInRework++;

   //Line scheduling...
   if (TicksBeforeRun) { //If there's time to wait for the line to get setup, then wait
      TicksBeforeRun--;
      return;
   }

   //If run length > BatchSize, then run the number
   //Processing...
    if (RunLength) {
      unsigned int PotentialUnitsToProcess = (RunLength < BatchSize ? RunLength :
BatchSize);
      for (map <PARTID, unsigned int>::iterator i=mySupplier->PartComposition.begin();i
!= mySupplier->PartComposition.end(); i++)
         if ((mySupplier->RMI[i->first].PartCount() / i->second) <
PotentialUnitsToProcess)
            PotentialUnitsToProcess = (mySupplier->RMI[i->first].PartCount() / i-
>second);

      if ((Line.Capacity != UNLIMITED) && ((Line.Capacity - Line.PartCount()) <
PotentialUnitsToProcess))
         PotentialUnitsToProcess = Line.Capacity - Line.PartCount();

      if (!Line.SpaceAvailable()) PotentialUnitsToProcess = 0; //No room because of
tacttime

      if ((RunIncompleteBatches) || (PotentialUnitsToProcess == BatchSize) ||
(PotentialUnitsToProcess == RunLength)) {
         for (unsigned int i=0;i<PotentialUnitsToProcess;i++) {
            Part newPart;
            newPart.CreatedTime = CurrentTick;
            newPart.ID = mySupplier->ID;
            newPart.BadConstituentParts = 0;
            for (map <PARTID, unsigned int>::iterator i=mySupplier-
>PartComposition.begin();i != mySupplier->PartComposition.end(); i++)
               for (unsigned int x=0;x<i->second;x++) {
                  Part p = mySupplier->RMI[i->first].RemovePart();
```

```
                if ((p.BadConstituentParts) || (p.MfgFailure))
newPart.BadConstituentParts++;
                newPart.SourceParts.push_back(p);
              }
            if ((float)rand() / (float)RAND_MAX > DesignQuality * (ManufacturingQuality +
InvestedQuality))
               newPart.MfgFailure = true;   else newPart.MfgFailure = false;

            Line.AddPart(newPart);
            RunLength--;
         }
      }
   }
}

Supplier::Supplier() {
   Line.RunIncompleteBatches = false;
   Line.mySupplier = this;
   EOQPolicyElapseTime = 0;
   ROPPolicyElapseTime = 0;
   FGISafetyLevelElapseTime = 0;
   Tracking = false;
   strcpy(SupplierName, "");
   strcpy(PartName, "");
}

Supplier::Supplier(const Supplier& s) {
   //Copy constructor being executed
   printf(" copy const \n");
}

void Supplier::AddRMI(PARTID PartID, unsigned int UnitCount, SupplierInterface *Source,
float DeliveryTimeMean,
                float DeliveryTimeStd, float ShipmentFixedCost, float
ShipmentVariableCost,
                float PerTickStorageCost, float EstimatedFailureRate, TICKS
EstimatedDeliveryLeadTime,
                unsigned int InitialReorderPoint, unsigned int InitialReorderQuantity) {
   RMI[PartID].FIFO = true;
   RMI[PartID].Capacity = UNLIMITED;
   RMIDeliveryChannels[PartID].destination = &RMI[PartID];
   RMIDeliveryChannels[PartID].FlowTimeMean = DeliveryTimeMean;
   RMIDeliveryChannels[PartID].FlowTimeStd = DeliveryTimeStd;

   //Cost Data
   RMISuppliers[PartID] = Source;
   RMIShipmentFixedCost[PartID] = ShipmentFixedCost;
   RMIShipmentVariableCost[PartID] = ShipmentVariableCost;
   RMIPerTickStorageCost[PartID] = PerTickStorageCost;

   PartComposition[PartID] = UnitCount;

   RMIReorderPoint[PartID] = InitialReorderPoint;
   RMIReorderQuantities[PartID] = InitialReorderQuantity;

   //Allocate and fill the Dynamic Arrays for Failure and Delivery Tracking
   RMIDeliveryLeadTime[PartID].FixLength(HISTORICBUFFERSIZE);
   RMIFailureHistory[PartID].FixLength(HISTORICBUFFERSIZE);
   for (unsigned int i=0;i<HISTORICBUFFERSIZE;i++) {
      if ((float)rand() / (float)RAND_MAX > EstimatedFailureRate)
RMIFailureHistory[PartID].AddData(0); else RMIFailureHistory[PartID].AddData(1);
      RMIDeliveryLeadTime[PartID].AddData(EstimatedDeliveryLeadTime);
   }
}

unsigned int Supplier::OutstandingOrders(void) {
   unsigned int Quantity = 0;
   for (map<ORDERID, OrderRecord>::iterator i = OrdersReceived.begin(); i !=
OrdersReceived.end(); i++)
      Quantity += i->second.QuantityOutstanding;
   return Quantity;
```

```
}

void Supplier::Order(ORDERID orderID, unsigned int UnitCount, SupplierInterface
*Customer, bool BadPartReplacement) {
   OrdersReceived[orderID].OrderID = orderID;
   OrdersReceived[orderID].PartID = ID;
   OrdersReceived[orderID].BadPartReplacement = BadPartReplacement;
   OrdersReceived[orderID].OrderedTime = CurrentTick;
   OrdersReceived[orderID].Quantity = UnitCount;
   OrdersReceived[orderID].QuantityOutstanding = UnitCount;
   OrdersReceived[orderID].DeliveryTime.clear();
   OrdersReceived[orderID].DeliveryQuantities.clear();
   OrdersReceived[orderID].Customer = Customer;

}

unsigned int Supplier::RMIOnOrder(PARTID PartID) {
   unsigned int OutstandingRMI = 0;
   for (map <ORDERID, OrderRecord>::iterator OMi = OrdersMade.begin(); OMi !=
OrdersMade.end(); OMi++)
      if (OMi->second.PartID == PartID)
         OutstandingRMI += OMi->second.QuantityOutstanding;

   OutstandingRMI += RMIDeliveryChannels[PartID].PartCount();

   return OutstandingRMI;
}

unsigned int Supplier::RMIOrdered(PARTID PartID) {
   unsigned int OrderedCount = 0;
   for (map <ORDERID, OrderRecord>::iterator OMi = OrdersMade.begin(); OMi !=
OrdersMade.end(); OMi++)
      if (OMi->second.PartID == PartID)
         OrderedCount += OMi->second.Quantity;
   return OrderedCount;
}

unsigned int Supplier::RMIReceived(PARTID PartID) {
   unsigned int ReceivedCount = 0;
   for (map <ORDERID, OrderRecord>::iterator OMi = OrdersMade.begin(); OMi !=
OrdersMade.end(); OMi++)
      if (OMi->second.PartID == PartID)
         ReceivedCount += (OMi->second.Quantity - OMi->second.QuantityOutstanding);
   return ReceivedCount;
}

float Supplier::DemandMean(TICKS PerTimeFrame, unsigned int TimeFramesToConsider) {
   //Demand on average within each PerTimeFrame
   //For example if I wanted Average Demand per month over the last 12 months then:
DemandMean(MONTH, 12)
   //                                    per week over 40 weeks then:
DemandMean(DAY*7, 40)
   DynamicArray Demand;
   TICKS EarliestTimeToConsider = CurrentTick - (PerTimeFrame * TimeFramesToConsider);
   if ((EarliestTimeToConsider < 0) || (TimeFramesToConsider == -1)) {
      EarliestTimeToConsider = 0;
      TimeFramesToConsider = 1 + (CurrentTick / PerTimeFrame);
   }

   unsigned int QuantitiesOrderedInTimeFrame = 0;

   for (map <ORDERID, OrderRecord>::iterator OMi = OrdersReceived.begin(); OMi !=
OrdersReceived.end(); OMi++)
      if (OMi->second.OrderedTime >= EarliestTimeToConsider)
         QuantitiesOrderedInTimeFrame += OMi->second.Quantity;

   return (float)QuantitiesOrderedInTimeFrame / (float)TimeFramesToConsider;
}

float Supplier::DemandStd(TICKS PerTimeFrame, unsigned int TimeFramesToConsider) {
   DynamicArray Demand;
```

```
   TICKS EarliestTimeToConsider = CurrentTick - (PerTimeFrame * TimeFramesToConsider);
   if ((EarliestTimeToConsider < 0) || (TimeFramesToConsider == -1)) {
      EarliestTimeToConsider = 0;
      TimeFramesToConsider = 1 + (CurrentTick / PerTimeFrame);
   }

   for (TICKS Time = EarliestTimeToConsider; Time <= CurrentTick; Time += PerTimeFrame) {
      unsigned int QuantitiesOrderedInTimeFrame;

      QuantitiesOrderedInTimeFrame = 0;
      unsigned int size = OrdersMade.size();

      for (map <ORDERID, OrderRecord>::iterator OMi = OrdersReceived.begin(); OMi !=
OrdersReceived.end(); OMi++) {
         if ((OMi->second.OrderedTime >= Time) && (OMi->second.OrderedTime < Time +
PerTimeFrame))
            QuantitiesOrderedInTimeFrame += OMi->second.Quantity;
      }

      Demand.AddData(QuantitiesOrderedInTimeFrame);
   }
   return Demand.Std();
}


void Supplier::DeliverParts(ORDERID orderID, Inventory inv, bool BadPartReplacement) {

   if (OrdersMade.find(orderID) == OrdersMade.end() ) {
      printf("Recieved an order of %d units without ording the said units, orderID:
%d\n", inv.PartCount(), orderID);
      return;
   }
   unsigned int PartCount = inv.PartCount();

   OrdersMade[orderID].DeliveryQuantities.push_back(PartCount);
   OrdersMade[orderID].DeliveryTime.push_back(CurrentTick -
OrdersMade[orderID].OrderedTime);

   //Account for shipping cost
   if (!BadPartReplacement) { //assumes the cost of shippng a replacement is paid by
supplier, no cost to replace
      Cash -= RMIShipmentFixedCost[OrdersMade[orderID].PartID];
      Cash -= RMIShipmentVariableCost[OrdersMade[orderID].PartID] * PartCount;
      Cash -= RMISuppliers[OrdersMade[orderID].PartID]->Price();
   }
   //End accounting

   for (;(OrdersMade[orderID].QuantityOutstanding && inv.PartCount());
OrdersMade[orderID].QuantityOutstanding--) {
      Part p = inv.RemovePart();
      RMIDeliveryChannels[OrdersMade[orderID].PartID].AddPart(p);
      RMIDeliveryLeadTime[OrdersMade[orderID].PartID].AddData((CurrentTick -
OrdersMade[orderID].OrderedTime) +
RMIDeliveryChannels[OrdersMade[orderID].PartID].LastAssignedFlowTime);
   }

   if (inv.PartCount())
      printf("%d extra parts received in order id: %d, parts discarded\n",
inv.PartCount(), orderID);

   //ExternalChange = true;
}

void Supplier::OrderRMI(PARTID PartID, unsigned int UnitCount, bool BadPartReplacement) {
   ORDERID OrderID = NextOrderID++;
   OrdersMade[OrderID].OrderID = OrderID;
   OrdersMade[OrderID].PartID = PartID;
   OrdersMade[OrderID].BadPartReplacement = true;
   OrdersMade[OrderID].Quantity = UnitCount;
   OrdersMade[OrderID].QuantityOutstanding = UnitCount;
   OrdersMade[OrderID].DeliveryQuantities.clear();
```

```
   OrdersMade[OrderID].DeliveryTime.clear();
   OrdersMade[OrderID].OrderedTime = CurrentTick;
   RMISuppliers[PartID]->Order(OrderID, UnitCount, this, BadPartReplacement);
}

ORDERID Supplier::GetOldestOutStandingOrder() {
   TICKS OldestOrderTime = CurrentTick;
   ORDERID OldestOrderID = NOORDER;
   for (map <ORDERID, OrderRecord>::iterator ORi = OrdersReceived.begin(); ORi !=
OrdersReceived.end(); ORi++)
      if ((ORi->second.QuantityOutstanding) && (ORi->second.OrderedTime <
OldestOrderTime)) {
         OldestOrderTime = ORi->second.OrderedTime;
         OldestOrderID = ORi->second.OrderID;
      }
   return OldestOrderID;
}

void Supplier::DeliverFGI(unsigned int UnitCount) {
   //Find the oldest order that's still active and deliver some units
   while ((UnitCount) && (FGI.PartCount())) {
      ORDERID OrderID = GetOldestOutStandingOrder();
      if (OrderID == NOORDER) { printf("Attempted to deliver unordered goods\n"); return;
}
      unsigned int UnitsToDeliverUnderThisOrder = min(UnitCount,
OrdersReceived[OrderID].QuantityOutstanding);
      Inventory inv;
      for (unsigned int i=0;i<UnitsToDeliverUnderThisOrder;i++) {
         Part p = FGI.RemovePart();
         inv.AddPart(p);
      }
      OrdersReceived[OrderID].Customer->DeliverParts(OrderID, inv,
OrdersReceived[OrderID].BadPartReplacement);
      OrdersReceived[OrderID].QuantityOutstanding -= UnitsToDeliverUnderThisOrder;
      OrdersReceived[OrderID].DeliveryQuantities.push_back(UnitsToDeliverUnderThisOrder);
      OrdersReceived[OrderID].DeliveryTime.push_back(CurrentTick -
OrdersReceived[OrderID].OrderedTime);
      UnitCount -= UnitsToDeliverUnderThisOrder;

      //Account for revenues received
      if (OrdersReceived[OrderID].BadPartReplacement)
         Cash -= UnitsToDeliverUnderThisOrder * ReplacementPartCost;
      else
         Cash += UnitsToDeliverUnderThisOrder * Price();

      //If it's late take away the penalty
      if (CurrentTick - OrdersReceived[OrderID].OrderedTime > ContractedDeliveryWindow)
         Cash -= LateDeliveryPenalty * UnitsToDeliverUnderThisOrder;
      //End accounting
   }
   if (UnitCount) printf("Unable to deliver all requested units due to empty FGI\n");
}


float Supplier::RMIFailureRate(PARTID PartID) {
   return RMIFailureHistory[PartID].Mean();
}

float Supplier::RMIDeliveryTime(PARTID PartID) {
   return RMIDeliveryLeadTime[PartID].Mean();
}

float Supplier::FGIUnitCost() {

   if (CurrentTick % GetTicks(1)) //Only recalculate once a day
      return __Cost;

   float FGICost = ((FixedBuildLabor / Line.BatchSize) + VariableBuildLabor) * LaborRate;
   FGICost += ((FixedBuildEquipment / Line.BatchSize) + VariableBuildEquipment) *
EquipmentRate;
```

```
   for (map <PARTID, unsigned int>::iterator Parti = PartComposition.begin(); Parti !=
PartComposition.end(); Parti++)
      FGICost += Parti->second * RMISuppliers[Parti->first]->Price();

   //Cost of failures
   float Pf = 1; //Probability of a failure occurring
   for (map <PARTID, unsigned int>::iterator Parti = PartComposition.begin(); Parti !=
PartComposition.end(); Parti++)
      Pf *= powf(1 - RMIFailureHistory[Parti->first].Mean(), Parti->second);

   Pf = 1 - Pf;

   float ReworkTimes = Line.TestCoverage * (Pf + powf(Pf, 2) + powf(Pf, 3));  //Number of
times the part will be reworked
   float ReworkCost = ReworkTimes * (TeardownAndRebuildLabor * LaborRate +
TeardownAndRebuildEquipment * EquipmentRate);

   FGICost += ReworkCost;

   //Inventory Cost
   for (map <PARTID, unsigned int>::iterator Parti = PartComposition.begin(); Parti !=
PartComposition.end(); Parti++) {
      float Demand = (float)DemandMean(1, OperatingLevelTimeConsideration)*Parti->second;
      if (Demand < 1) Demand = 1;
      float AverageShelfTime = ((float)RMISafetyStock[Parti->first] +
(float)RMIReorderQuantities[Parti->first] / 2.0) / Demand;
      FGICost += AverageShelfTime * (RMIPerTickStorageCost[Parti->first] + WACC *
RMISuppliers[Parti->first]->Price());
   }

   __Cost = FGICost;
   return FGICost;
}

float Supplier::Price(void) {
   return FGIUnitCost() * (1+Margin);
}

void Supplier::ReviewPolicies() {

   if (++EOQPolicyElapseTime > EOQPolicyReviewTime) {
      EOQPolicyElapseTime = 0;
      for (map<PARTID, unsigned int>::iterator PCi = PartComposition.begin(); PCi !=
PartComposition.end(); PCi++) {
         float D = DemandMean(1, OperatingLevelTimeConsideration)*PCi->second;
         float ShippingCost = RMIShipmentFixedCost[PCi->first];
         float PurchasePrice = RMISuppliers[PCi->first]->Price();
         float UnitStorageCost = RMIPerTickStorageCost[PCi->first];
         float a = sqrt(2*D*ShippingCost / (PurchasePrice * WACC + UnitStorageCost));
         RMIReorderQuantities[PCi->first] = a;
      }
   }

   if (++ROPPolicyElapseTime > ROPPolicyReviewTime) {
      ROPPolicyElapseTime = 0;

      float ServiceLevel = 0.9997;

      float Demand = DemandMean(1, OperatingLevelTimeConsideration);
      for (map<PARTID, unsigned int>::iterator PCi = PartComposition.begin(); PCi !=
PartComposition.end(); PCi++) {
         unsigned LeadTimeDemand = RMIDeliveryLeadTime[PCi->first].Mean() * Demand * PCi-
>second;
         float LeadTimeDemandVar = pow(RMIDeliveryLeadTime[PCi->first].Std() * Demand *
PCi->second, 2) +
            pow((DemandStd(DAY*30, OperatingLevelTimeConsideration) / (DAY*30)) * PCi-
>second * RMIDeliveryLeadTime[PCi->first].Mean(), 2);
         float t = cdfinv(ServiceLevel);
         unsigned int SafetyStock = t * sqrt(LeadTimeDemandVar);
         RMIReorderPoint[PCi->first] = LeadTimeDemand + SafetyStock;
```

```
        if (RMIReorderQuantities[PCi->first] < LeadTimeDemand) RMIReorderQuantities[PCi-
>first] = LeadTimeDemand;
      }


   }

   if (++FGISafetyLevelElapseTime > FGISafetyLevelReviewTime) {
      FGISafetyLevelElapseTime = 0;

      float ServiceLevel = 0.9997;

      FGISafetyLevel = 0;
   }
}

void Supplier::operator ++() {

   ReviewPolicies();
   //Account for costs
   if (Tracking) CashTrack.AddData(Cash);
   if (Tracking) FGICostTrack.AddData(FGIUnitCost());
   Cash -= (Line.wip() + FGI.PartCount()) * (FGIUnitCost() * WACC +
FGIPerTickStorageCost);
   for (map <PARTID, Inventory>::iterator RMIi = RMI.begin(); RMIi != RMI.end(); RMIi++)
      Cash -= (RMIi->second.PartCount() + RMIDeliveryChannels[RMIi->first].PartCount()) *
(RMISuppliers[RMIi->first]->Price() * WACC + RMIPerTickStorageCost[RMIi->first]);
   //End cost accounting


   ORDERID OrderID;
   //If there are FGI parts avail and outstanding orders, try to fill them
   while (( FGI.PartCount()) && ((OrderID = GetOldestOutStandingOrder()) != NOORDER)) {
      //If we can't fill the entire order and the order is not late don't do anything
      if ((FGI.PartCount() < OrdersReceived[OrderID].QuantityOutstanding) && (CurrentTick
- OrdersReceived[OrderID].OrderedTime < ContractedDeliveryWindow))
         break;
      //If we can will the entire order or the order is late then send what's required or
available
      DeliverFGI(min(FGI.PartCount(), OrdersReceived[OrderID].QuantityOutstanding));
   }

   if (Tracking) FGI++;
   if (Tracking) BadParts++;
   Line++;

   for (map <PARTID, Inventory>::iterator RMIi = RMI.begin(); RMIi != RMI.end(); RMIi++)
      RMIi->second.operator++();
   for (map <PARTID, Channel>::iterator DCi = RMIDeliveryChannels.begin(); DCi !=
RMIDeliveryChannels.end(); DCi++)
      DCi->second.operator++();

   //Decide what to do now
   //Total Parts available is FGI, in the works is Line WIP.

   //Check RMI Levels, Reorder if necessary
   //Check FGI, WIP, and Scheduled production for enough parts to cover demand

   //Basic...  Schedule the run
   //Check for RMI needs

   unsigned int o = OutstandingOrders() + FGISafetyLevel;
   unsigned int Scheduled = Line.PartsRemainingInRun() + Line.wip() + FGI.PartCount();

   if (o > Scheduled)
      Line.Run(o - Scheduled);

   unsigned int RMINeeds = Line.PartsRemainingInRun();
   //Determine the RMI Needs
   for (map <PARTID, Inventory>::iterator RMIi = RMI.begin(); RMIi != RMI.end(); RMIi++)
{
```

```
      unsigned int RMIOnHandOrOnOrder = RMIi->second.PartCount() + RMIOnOrder(RMIi-
>first);

      if (RMIOnHandOrOnOrder < RMIReorderPoint[RMIi->first]) {
         //Need to make an order
         if (RMIOnHandOrOnOrder + RMIReorderQuantities[RMIi->first] <
RMIReorderPoint[RMIi->first])
            OrderRMI(RMIi->first, RMIReorderPoint[RMIi->first] - RMIOnHandOrOnOrder);
         else
            OrderRMI(RMIi->first, RMIReorderQuantities[RMIi->first]);
      }
   }
}

void Supplier::PrintReport(bool full) {

   //Raw material report
   for (map <PARTID, DynamicArray>::iterator RMDTi = RMIDeliveryLeadTime.begin(); RMDTi
!= RMIDeliveryLeadTime.end(); RMDTi++) {
      TICKS AvgTime = RMDTi->second.Mean();
      TICKS StdTime = RMDTi->second.Std();

      printf("RMI %d Ordered: %d, Received: %d, On hand: %d, Failure rate: %.2f%%\n",
RMDTi->first, RMIOrdered(RMDTi->first),
         RMIReceived(RMDTi->first), RMI[RMDTi->first].PartCount() +
RMIDeliveryChannels[RMDTi->first].PartCount(), RMIFailureHistory[RMDTi
>first].Mean()*100);

      printf("  Delivery Time Mean: %d days %02d:%02d:%02d", GetDay(AvgTime),
GetHour(AvgTime), GetMinute(AvgTime), GetSecond(AvgTime));
      printf(", Std: %d days %02d:%02d:%02d\n", GetDay(StdTime), GetHour(StdTime),
GetMinute(StdTime), GetSecond(StdTime));

      printf("  Reorder point: %d, Economical Order Quantity: %d\n\n",
RMIReorderPoint[RMDTi->first], RMIReorderQuantities[RMDTi->first]);
   }
}

void StockSupplier::Order(ORDERID orderID, unsigned int UnitCount, SupplierInterface
*Customer, bool BadPartReplacement) {
   Inventory NewParts;
   for (unsigned int i=0;i<UnitCount;i++) {
      Part p;
      p.ID = ID;
      p.BadConstituentParts = false;
      if ((float)rand() / (float)RAND_MAX > PartFailureRate) p.MfgFailure = false; else
p.MfgFailure = true;
      p.CreatedTime = CurrentTick;
      p.SourceParts.clear();
      NewParts.AddPart(p);
   }
   Customer->DeliverParts(orderID, NewParts, BadPartReplacement);
}

void Supplier::TrackingOn() {
   Tracking = true;
   //Enable tracking features
   Line.Line.CollectVolume = true;
   Line.PartsInRework.CollectVolume = true;
   FGI.CollectVolume = true;
   for (map<PARTID, unsigned int>::iterator PCi = PartComposition.begin(); PCi !=
PartComposition.end(); PCi++) {
      RMI[PCi->first].CollectVolume = true;
      RMIDeliveryChannels[PCi->first].CollectVolume = true;
   }
}

void Supplier::SaveReport(char *FileName, char Separator, TICKS AveragedTime) {
   FILE *fil;
   fil = fopen(FileName, "w+");
```

```
   if (!fil) {
      printf("Unable to open file %s for writing\n", FileName);
      return;
   }

   time_t tim;
   time(&tim);
   fprintf(fil, "Data Collected from Supplier %s producing part %d on %s", SupplierName,
ID, ctime(&tim));
   fprintf(fil, "Ticks per day: %d, therefore one hour = %.2f ticks\n", TicksPerDay,
(float)TicksPerDay/24.0);
   fprintf(fil, "Data averaged over every %d ticks or %.2f hours\n\n", AveragedTime,
((float)AveragedTime / (float)TicksPerDay) * 24.0);

   fprintf(fil, "\n\n");
   //Demand Report
   fprintf(fil, "Order ID%cReplacement%cQuantity Ordered%cAverage Delivery Time%cOn Time
Deliveries%cPercent on Time%cQuantity Undelivered\n",
         Separator, Separator, Separator, Separator, Separator, Separator);
   for (map <ORDERID, OrderRecord>::iterator i = OrdersReceived.begin(); i !=
OrdersReceived.end(); i++) {
      fprintf(fil, "%d%c%s%c%d%c", i->second.OrderID, Separator, i-
>second.BadPartReplacement ? "YES" : "NO", Separator, i->second.Quantity, Separator);
      TICKS AverageTimeSum = 0;
      for (unsigned int x=0;x<i->second.DeliveryTime.size();x++)
         AverageTimeSum += i->second.DeliveryTime[x]*i->second.DeliveryQuantities[x];
      TICKS AverageDeliveryTime = 0;
      if (i->second.Quantity - i->second.QuantityOutstanding)
         AverageDeliveryTime = AverageTimeSum / (i->second.Quantity - i-
>second.QuantityOutstanding);
      fprintf(fil, "%d days %02d:%02d:%02d%c", GetDay(AverageDeliveryTime),
GetHour(AverageDeliveryTime), GetMinute(AverageDeliveryTime),
GetSecond(AverageDeliveryTime), Separator);

      unsigned int OnTimeDeliveries = 0;
      for (unsigned int x=0;x<i->second.DeliveryTime.size();x++)
         if (i->second.DeliveryTime[x] <= ContractedDeliveryWindow)
            OnTimeDeliveries += i->second.DeliveryQuantities[x];

      fprintf(fil, "%d%c%2.0f%%%c%d\n", OnTimeDeliveries, Separator,
(float)OnTimeDeliveries*100.0/(float)i->second.Quantity, Separator, i-
>second.QuantityOutstanding);
   }

   fprintf(fil, "\n\n");

   //RMI Report
   fprintf(fil, "RMI Part ID%cQuantity Ordered%cQuantity Received%cOn hand%cFailure
rate%c",Separator,Separator,Separator,Separator,Separator);
   fprintf(fil, "Delivery Time Mean%cDelivery Time Std%cReorder point%cEconomical Order
Quantity\n", Separator, Separator, Separator);
   for (map <PARTID, DynamicArray>::iterator RMDTi = RMIDeliveryLeadTime.begin(); RMDTi
!= RMIDeliveryLeadTime.end(); RMDTi++) {
      TICKS AvgTime = RMDTi->second.Mean();
      TICKS StdTime = RMDTi->second.Std();
      fprintf(fil, "%d%c", RMDTi->first, Separator);
      fprintf(fil, "%d%c%d%c%d%c%.2f%%%c", RMIOrdered(RMDTi->first), Separator,
RMIReceived(RMDTi->first), Separator, RMI[RMDTi->first].PartCount() +
RMIDeliveryChannels[RMDTi->first].PartCount(),
               Separator, RMIFailureHistory[RMDTi->first].Mean()*100, Separator);
      fprintf(fil, "%d days %02d:%02d:%02d%c", GetDay(AvgTime), GetHour(AvgTime),
GetMinute(AvgTime), GetSecond(AvgTime), Separator);
      fprintf(fil, "%d days %02d:%02d:%02d%c", GetDay(StdTime), GetHour(StdTime),
GetMinute(StdTime), GetSecond(StdTime), Separator);
      fprintf(fil, "%d%c%d\n", RMIReorderPoint[RMDTi->first], Separator,
RMIReorderQuantities[RMDTi->first]);
   }

   //Run-time Tracking Report
   if (Tracking) {
      fprintf(fil, "\n\n");
```

```
      fprintf(fil, "Relative Tick%c", Separator);
      fprintf(fil, "Time%c", Separator);
      fprintf(fil, "Production Line Volume%c", Separator);
      fprintf(fil, "Parts scheduled for Production%c", Separator);
      fprintf(fil, "Rework Queue%c", Separator);
      fprintf(fil, "Finished Goods Inventory%c", Separator);
      for (map<PARTID, unsigned int>::iterator PCi = PartComposition.begin(); PCi !=
PartComposition.end(); PCi++) {
         fprintf(fil, "RMI [%d] Inventory%c", PCi->first, Separator);
         fprintf(fil, "RMI [%d] Delivery Channel%c", PCi->first, Separator);
      }
      fprintf(fil, "%cCash%c", Separator, Separator);
      fprintf(fil, "Cost Per Unit%c", Separator);
      fprintf(fil, "\n");
      for (unsigned int i=0;i<CurrentTick;i+=AveragedTime) { //Not all will have begun at
CurrentTick = 0 so don't presume.  CurrentTick is just the Maximum.
         //Assume we're only condiering the autoexpand ones
         fprintf(fil, "%d%c", i, Separator);
         fprintf(fil, "Day %d, %02d:%02d:%02d%c", GetDay(i), GetHour(i), GetMinute(i),
GetSecond(i), Separator);
         fprintf(fil, "%g%c", (*Line.Line.Volume).SubMean(i, i+AveragedTime-1),
Separator);
         fprintf(fil, "%g%c", Line.DemandHistory.SubMean(i, i+AveragedTime-1),
Separator);
         fprintf(fil, "%g%c", (*Line.PartsInRework.Volume).SubMean(i, i+AveragedTime-1),
Separator);
         fprintf(fil, "%g%c", (*FGI.Volume).SubMean(i, i+AveragedTime-1), Separator);
         for (map<PARTID, unsigned int>::iterator PCi = PartComposition.begin(); PCi !=
PartComposition.end(); PCi++) {
            fprintf(fil, "%g%c", (*RMI[PCi->first].Volume).SubMean(i, i+AveragedTime-1),
Separator);
            fprintf(fil, "%g%c", (*RMIDeliveryChannels[PCi->first].Volume).SubMean(i,
i+AveragedTime-1), Separator);
         }
         fprintf(fil, "%c%g%c", Separator, CashTrack.SubMean(i, i+AveragedTime-1),
Separator);
         fprintf(fil, "%g%c", FGICostTrack.SubMean(i, i+AveragedTime-1), Separator);
         fprintf(fil, "\n");
      }
   }
   fclose(fil);
}

OEM::OEM(TICKS orderCycleTime, TICKS orderCycleTimeStd, unsigned int orderQuantityStd) :
   OrderCycleTime(orderCycleTime), OrderCycleTimeStd(orderCycleTimeStd),
OrderQuantityStd(orderQuantityStd) {
   TicksUntilNextOrder = normal(OrderCycleTime, OrderCycleTimeStd);
}

void OEM::OrderRMI(PARTID PartID, unsigned int Quantity, bool BadPartReplacement) {
   ORDERID OrderID = NextOrderID++;
   OrdersMade[OrderID].OrderID = OrderID;
   OrdersMade[OrderID].PartID = PartID;
   OrdersMade[OrderID].Quantity = Quantity;
   OrdersMade[OrderID].QuantityOutstanding = OrdersMade[OrderID].Quantity;
   OrdersMade[OrderID].DeliveryQuantities.clear();
   OrdersMade[OrderID].DeliveryTime.clear();
   OrdersMade[OrderID].OrderedTime = CurrentTick;
   OrdersMade[OrderID].BadPartReplacement = BadPartReplacement;

   Suppliers[PartID]->Order(OrderID, OrdersMade[OrderID].Quantity, this,
OrdersMade[OrderID].BadPartReplacement);
}

void OEM::DeliverParts(ORDERID orderID, Inventory inv, bool BadPartReplacement) {

   if (OrdersMade.find(orderID) == OrdersMade.end() ) {
      printf("Recieved an order of %d units without ording the said units, orderID:
%d\n", inv.PartCount(), orderID);
      return;
   }
```

```
   OrdersMade[orderID].DeliveryQuantities.push_back(inv.PartCount());
   OrdersMade[orderID].DeliveryTime.push_back(CurrentTick -
OrdersMade[orderID].OrderedTime);

   for (;(OrdersMade[orderID].QuantityOutstanding && inv.PartCount());
OrdersMade[orderID].QuantityOutstanding--) {
      Part p = inv.RemovePart();
      FGI.AddPart(p);
      if ((p.BadConstituentParts) || (p.MfgFailure))
         OrderRMI(p.ID, 1, true);
   }

   if (inv.PartCount())
      printf("%d extra parts received in order id: %d, parts discarded\n",
inv.PartCount(), orderID);
}

void OEM::operator ++() {
   if (!(--TicksUntilNextOrder)) {

      for (map<PARTID, float>::iterator i = Demand.begin(); i != Demand.end(); i++) {
         int QuantityToOrder = normal(i->second, OrderQuantityStd);
         if (QuantityToOrder > 0)
            OrderRMI(i->first, QuantityToOrder);
         i->second += DemandIncrease[i->first];
      }
      TicksUntilNextOrder = normal(OrderCycleTime, OrderCycleTimeStd);
   }
}

void OEM::SaveOrderReport(char *FileName, char Separator) {
   FILE *fil = fopen(FileName, "w+");
   if (!fil) {
      printf("Unable to open %s for OEM reporting ", FileName);
      perror("");
      return;
   }

   time_t tim;
   time(&tim);
   fprintf(fil, "Data Collected from OEM on %s", ctime(&tim));
   fprintf(fil, "Ticks per day: %d, therefore one hour = %.2f ticks\n", TicksPerDay,
(float)TicksPerDay/24.0);
   fprintf(fil, "Complete Shipment report\n");
   fprintf(fil, "Order ID%cPart ID%cPart Name%cReplacement%cQuantity Ordered%cAverage
Delivery Time%cOn time Deliveries%cOn Time Delivery Percent%cQuantity Undelivered\n",
               Separator, Separator, Separator, Separator, Separator, Separator,
Separator, Separator);
   for   (map <ORDERID, OrderRecord>::iterator i   = OrdersMade.begin(); i   !=
OrdersMade.end(); i++) {
      fprintf(fil, "%d%c%d%c%s%c%s%c%d%c", i->second.OrderID, Separator, i-
>second.PartID, Separator,
            Suppliers[i->second.PartID]->PartName, Separator, i-
>second.BadPartReplacement ? "YES" : "NO", Separator, i->second.Quantity, Separator);
      TICKS AverageTimeSum = 0;
      for   (unsigned int x=0;x<i->second.DeliveryTime.size();x++)
         AverageTimeSum += i->second.DeliveryTime[x]*i->second.DeliveryQuantities[x];
      TICKS AverageDeliveryTime =   0;
      if (i->second.Quantity - i->second.QuantityOutstanding)
         AverageDeliveryTime   = AverageTimeSum / (i->second.Quantity - i-
>second.QuantityOutstanding);
      fprintf(fil, "%d days %02d:%02d:%02d%c", GetDay(AverageDeliveryTime),
GetHour(AverageDeliveryTime), GetMinute(AverageDeliveryTime),
GetSecond(AverageDeliveryTime), Separator);

      unsigned int OnTimeDeliveries =   0;
      for   (unsigned int x=0;x<i->second.DeliveryTime.size();x++)
         if (i->second.DeliveryTime[x] <= Suppliers[i->second.PartID]-
>ContractedDeliveryWindow)
            OnTimeDeliveries +=   i->second.DeliveryQuantities[x];
```

```
      fprintf(fil, "%d%c%2.0f%%%c%d\n", OnTimeDeliveries, Separator,
(float)OnTimeDeliveries*100.0/(float)i->second.Quantity,   Separator, i-
>second.QuantityOutstanding);
   }

}
```

```
//File: StatisticsBox.h

#define PI (3.141592653589793)

float cdfinv(float p  );
float normal(float mean, float stdev);

class DynamicArray {
   float *Data;
   unsigned int Size;
   unsigned int BufferSize; //Number of floats that can be stored
   unsigned int GrowthRate; //Rate at which to grow the Databuffer
   bool AutoExpand;
   unsigned int SamplePosition; //Position of next sample (Size independant)
public:
   DynamicArray();
   DynamicArray(const DynamicArray& a);
   DynamicArray(unsigned int size); //Fixed Length Buffer
   const DynamicArray& operator=(const DynamicArray& a);
   void FixLength(unsigned int NewSize); //Switch to a fixed length buffer

   ~DynamicArray() { if (Data) free(Data); }

   unsigned int AddData(float dataPoint);
   float operator [] (unsigned int index) { if (index < Size) return Data[index]; else
return sqrt(-1); }
   operator unsigned int () { return Size; }
   unsigned int size() {return Size; }
   void Plot(bool Bars = false, int Width = 78, int Height = 18, bool Labels = false,
bool Border = true, char *Title = NULL);
   void Histogram(int Width = 78, int Height = 18, bool Labels = false, bool Border =
true, char *Title = NULL);

   void SetGrowthRate(unsigned int rate) { GrowthRate = rate; }

   float SubMean(unsigned int LowIndex, unsigned int HighIndex);
   float SubMax(unsigned int LowIndex, unsigned int HighIndex);
   float SubMin(unsigned int LowIndex, unsigned int HighIndex);
   float SubStd(unsigned int LowIndex, unsigned int HighIndex);

   float Mean(void) {return SubMean(0, Size-1); }
   float Max(void)  {return SubMax (0, Size-1); }
   float Min(void)  {return SubMin (0, Size-1); }
   float Std(void)  {return SubStd (0, Size-1); }

   void WriteToFile(FILE *fil, char Separator);

};
```

```
//File: StatisticsBox.cpp

#include "stdafx.h"

float getMean(float values[], unsigned int count);
float getStdev(float values[], unsigned int count);
float min(float values[], unsigned int count);
float max(float values[], unsigned int count);
int min(int values[], unsigned int count);
int max(int values[], unsigned int count);
inline int max(int a, int b) { return (a > b ? a : b); }
inline int min(int a, int b) { return (a < b ? a : b); }



int __compare( const void *arg1, const void *arg2 )
{
   int retval;
   float a, b;
   a = *(float*)arg1;
   b = *(float*)arg2;
   return (int)(a - b);
}


DynamicArray::DynamicArray() {
   GrowthRate = BufferSize = 500;
   Size = 0;
   Data = (float*)malloc(BufferSize*sizeof(float));
   AutoExpand = true;
}

DynamicArray::DynamicArray(const DynamicArray& a) {
   Data = NULL;
   this->operator=(a);
}

DynamicArray::DynamicArray(unsigned int size) {
   GrowthRate = 0;
   BufferSize = size;
   Size = 0;
   Data = (float*)malloc(BufferSize*sizeof(float));
   AutoExpand = false;
   SamplePosition = 0;
}

void DynamicArray::FixLength(unsigned int NewSize) {
   float *OldData = Data;
   Data = (float*)malloc(NewSize*sizeof(float));

   unsigned int Trim = (Size > NewSize ? Size - NewSize : 0);
   unsigned int Length = min(Size, NewSize);
   if ((AutoExpand) || (Size < BufferSize)) {
      memcpy(Data, &OldData[Trim], Length*sizeof(float));
   } else { //Two part copy required
      unsigned int Start = SamplePosition + Trim;
      if (Start >= BufferSize) Start -= BufferSize;
      memcpy(Data, &OldData[Start], min(Length, BufferSize - Start)*sizeof(float));
      memcpy(&Data[min(Length, BufferSize - Start)], OldData, (Length - min(Length,
BufferSize - Start))*sizeof(float));
   }
   free(OldData);
   GrowthRate = 0;
   BufferSize = NewSize;
   Size = min(Size, NewSize);
   AutoExpand = false;
   SamplePosition = Size;
   if (SamplePosition >= BufferSize) SamplePosition = 0;
}
```

```
const DynamicArray& DynamicArray::operator =(const DynamicArray& a) {
   if (Data) free(Data);
   Size = a.Size;
   BufferSize = a.BufferSize;
   GrowthRate = a.GrowthRate;
   Data = (float*)malloc(BufferSize*sizeof(float));
   memcpy(Data, a.Data, BufferSize*sizeof(float));
   AutoExpand = a.AutoExpand;
   SamplePosition = a.SamplePosition;
   return *this;
}

unsigned int DynamicArray::AddData(float dataPoint) { //adds a data item, return position
if added, -1 if there's not enough memory
   if (AutoExpand) {
      if (Size == BufferSize) { //Need to grow the buffer
         float *oldData = Data;
         Data = (float*)malloc(sizeof(float)*(BufferSize+GrowthRate));
         if (!Data) { Data = oldData; return -1; }
         memcpy(Data, oldData, BufferSize*sizeof(float));
         free(oldData);
         BufferSize += GrowthRate;
      }
      Data[Size] = dataPoint;
      return Size++;
   } else {
      Data[SamplePosition++] = dataPoint;
      if (Size < BufferSize) Size++;
      if (SamplePosition >= BufferSize) SamplePosition = 0;
      return SamplePosition;
   }
}

void DynamicArray::Histogram(int Width, int Height, bool Labels, bool Border, char
*Title) {

   float min_value = Min();
   float max_value = Max();
   unsigned BarWidth = Width;
   unsigned BarHeight = Height;
   float Scaler = (float)BarWidth/(max_value-min_value);
   float Offset = 0 - min_value;
   unsigned int *Counts = (unsigned int*)malloc(sizeof(unsigned int) * BarWidth);
   memset(Counts, 0, BarWidth*sizeof(unsigned int));
   for (unsigned int i=0;i<Size;i++)
      Counts[(unsigned int)((Data[i] + Offset) * Scaler)] ++;

   if (Border) { //Top Border
      printf("%c", 218);
      for (unsigned int x=0;x<BarWidth;x++) {
         if (Title != NULL) {
            if ((Width + 2 - strlen(Title)) / 2 == x) {
               printf("%s", Title);
               x += strlen(Title);
            }
         }
         printf("%c", 196);
      }
      printf("%c", 191);
      if (BarWidth < 80 - Border * 2) printf("\n");
   }

   float HeightScaler = (float)BarHeight / max((int *)Counts, BarWidth);
   for (int y=BarHeight;y>0;y--) {

      if (Border) printf("%c", 179);

      for (int x=0;x<BarWidth;x++)
         printf("%c", (Counts[x] * HeightScaler >= y ? 219 : ' '));

      if (Border) printf("%c", 179);
```

```
      if (BarWidth < 80 - Border * 2) printf("\n");
   }

   if (Border) { //Bottom Border
      printf("%c", 192);
      for (unsigned int x=0;x<BarWidth;x++)
         printf("%c", 196);
      printf("%c", 217);
      if (BarWidth < 80 - Border * 2) printf("\n");
   }

   float std = Std();

   if (Labels) {
      for (unsigned int y=0;y<3;y++) {
         for (unsigned int x=0;x<BarWidth;x++)
            if (x == (unsigned int)((Mean() + Offset) * Scaler)) {
               if (y == 0) printf("^"); else if (y==1) printf("|"); else if (y==2)
printf("Mean: %.5g ", Mean());
            } else if ((( x == (unsigned int)((Mean() + std + Offset) * Scaler)) || ( x
== (unsigned int)((Mean() - std + Offset) * Scaler))) && (y == 0)) {
               printf("|");
            } else
               printf(" ");
         printf("\n");
      }
   }
}

void DynamicArray::Plot(bool Bars, int Width, int Height, bool Labels, bool Border, char
*Title) {

   //Account for fixed length data
   if ((!AutoExpand) && (SamplePosition)) {
      DynamicArray Sorted = *this;
      Sorted.FixLength(Size);
      Sorted.Plot(Bars, Width, Height, Labels, Border, Title);
      return;
   }

   float min_value = Min();
   float max_value = Max();
   unsigned BarWidth = Width;
   unsigned BarHeight = Height;

   DynamicArray MeanValues, MaxValues, MinValues;
   float Offset = -min_value;
   float HeightScaler = (float)BarHeight / (max_value - min_value);
   for (unsigned int x=0;x<BarWidth;x++) {
      unsigned int LowIndex = (Size * x) / BarWidth;
      unsigned int HighIndex = (Size * (x+1)) / BarWidth - 1;
      MeanValues.AddData((int)((SubMean(LowIndex, HighIndex) + Offset) * HeightScaler));
      if (Bars) {
         MaxValues.AddData((int)((SubMax(LowIndex, HighIndex) + Offset) * HeightScaler));
         MinValues.AddData((int)((SubMin(LowIndex, HighIndex) + Offset) * HeightScaler));
      }
   }

   if (Border) { //Top Border
      printf("%c", 218);
      for (unsigned int x=0;x<BarWidth;x++) {
         if (Title != NULL) {
            if ((Width + 2 - strlen(Title)) / 2 == x) {
               printf("%s", Title);
               x += strlen(Title);
            }
         }
         printf("%c", 196);
      }
      printf("%c", 191);
      if (Labels) printf("<--Max: %.5g", Max());
```

```
      if (BarWidth < 80 - Border * 2) printf("\n");
   }

   int GlobalMean=Mean() * HeightScaler;
   for (int y=BarHeight;y>0;y--) {
      if (Border) printf("%c", 179);

      for (int x=0;x<BarWidth;x++)
         if (MeanValues[x] == y) {
            if (Bars) {
               if ((MeanValues[x] == MaxValues[x]) && (MeanValues[x] == MinValues[x]))
printf("%c", 248);
               else if (MeanValues[x] == MaxValues[x]) printf("%c", 194 /*24*/);
               else if (MeanValues[x] == MinValues[x]) printf("%c", 193 /*25*/);
               else printf("%c", 249);
            } else printf("%c", 249);
         }
         else if ((Bars) && (MaxValues[x] == y)) printf("%c", 194);
         else if ((Bars) && (MinValues[x] == y)) printf("%c", 193);
         else printf(" ");

      if (Border) printf("%c", 179);
      if ((Labels) && (GlobalMean == y)) printf("<--Mean: %.5g", Mean());
      if (BarWidth < 80 - Border * 2) printf("\n");
   }

   if (Border) { //Bottom Border
      printf("%c", 192);
      for (unsigned int x=0;x<BarWidth;x++)
         printf("%c", 196);
      printf("%c", 217);
      if (Labels) printf("<--Min: %.5g", Min());
      if (BarWidth < 80 - Border * 2) printf("\n");
   }

}

float DynamicArray::SubMean(unsigned int LowIndex, unsigned int HighIndex) {
   if ((LowIndex < Size) && (HighIndex < Size))
      return getMean(&Data[LowIndex], (HighIndex-LowIndex) + 1);
   else return sqrt(-1);
}

float DynamicArray::SubMax(unsigned int LowIndex, unsigned int HighIndex) {
   if ((LowIndex < Size) && (HighIndex < Size))
      return max(&Data[LowIndex], (HighIndex-LowIndex) + 1);
   else return sqrt(-1);
}

float DynamicArray::SubMin(unsigned int LowIndex, unsigned int HighIndex) {
   if ((LowIndex < Size) && (HighIndex < Size))
      return min(&Data[LowIndex], (HighIndex-LowIndex) + 1);
   else return sqrt(-1);
}

float DynamicArray::SubStd(unsigned int LowIndex, unsigned int HighIndex) {
   if ((LowIndex < Size) && (HighIndex < Size))
      return getStdev(&Data[LowIndex], (HighIndex-LowIndex) + 1);
   else return sqrt(-1);
}

void DynamicArray::WriteToFile(FILE *fil, char Separator) {
   if (!Size) return;

   //Account for fixed length data
   if ((!AutoExpand) && (SamplePosition)) {
      DynamicArray Sorted = *this;
      Sorted.FixLength(Size);
      Sorted.WriteToFile(fil, Separator);
      return;
   }
```

```
    for (unsigned int i=0;i<Size-1;i++)
      fprintf(fil, "%g%c", Data[i], Separator);
   fprintf(fil, "%g", Data[Size-1]);
}


double CsubK(unsigned int k) {

   static double CSubK[3000];
   static int ckLimit = -1;
   if (k >= 3000) return CsubK(2999);
   if ((ckLimit != -1) && (ckLimit >= k)) return CSubK[k];

   double result = 0;
   if (k == 0) {
      result = 1.0;
   } else {
      for (unsigned int m=0;m < k; m++) {
         result += CsubK(m)*CsubK(k-1-m)/((m+1) * (2*m+1));
      }
   }
   CSubK[k] = result;
   ckLimit = k;
   return result;
}

double erfinv(float x) {
   double result = 0;
   double SqrtOfPi = sqrt(PI);
   double xOver2 = x/2;
   double SqrtOfPiTimesxOver2 = SqrtOfPi * xOver2;
   for (unsigned int k=0;k<1000;k++) {
      result += (CsubK(k)/(2*k + 1)) * pow(SqrtOfPiTimesxOver2, (2*k+1));
   }
   return result;
}

float cdfinv(float p  ) {
   return sqrt(2)*erfinv(2*p - 1);
}

float normal(float mean, float stdev) {
   float value;
   value = mean + stdev*sqrt(-2*log((float)(abs(rand()-
1)+1)/RAND_MAX))*cos(2*3.14159*(float)(abs(rand()-1)+1)/RAND_MAX);
   return value;
}


float getMean(float values[], unsigned int count) {
   double total = 0;
   for (int x=0;x<count;x++) {
      total += values[x];
   }
   return total / (float)count;
}

float getStdev(float values[], unsigned int count) {
   double total = 0;
   float mean = getMean(values, count);
   for (int x=0;x<count;x++) {
      total += (values[x] - mean)*(values[x] - mean);
   }
   total /= (count - 1);
   return sqrt(total);
}

float min(float values[], unsigned int count) {
   if (!count) return 0;
   float minval = values[0];
```

```
   for (unsigned int x=1;x<count;x++)
      if (values[x] < minval) minval = values[x];
   return minval;
}

float max(float values[], unsigned int count) {
   if (!count) return 0;
   float maxval = values[0];
   for (unsigned int x=1;x<count;x++)
      if (values[x] > maxval) maxval = values[x];
   return maxval;
}

int min(int values[], unsigned int count) {
   if (!count) return 0;
   int minval = values[0];
   for (unsigned int x=1;x<count;x++)
      if (values[x] < minval) minval = values[x];
   return minval;
}

int max(int values[], unsigned int count) {
   if (!count) return 0;
   int maxval = values[0];
   for (unsigned int x=1;x<count;x++)
      if (values[x] > maxval) maxval = values[x];
   return maxval;
}


int compare( const void *arg1, const void *arg2 )
{
   int retval;
   float a, b;
   a = *(float*)arg1;
   b = *(float*)arg2;
   return (int)(a - b);
}


unsigned int trim(float values[], unsigned int count, float percent) {
   qsort(values, count, sizeof(float), compare);
   unsigned int newSize = ((float)count * (1-percent));
   memmove(values, &values[(unsigned int)(count*(percent/2))], newSize);
   return newSize;
}
```

Appendix E - Supplier Network Model Parameters

Model Parameter	Lead CM	Chassis Assembler	Head and Neck Assembler	Battery Supplier	Flipper Assembler	Chassis Track Supplier	Side Panel Supplier	Electronics Box Supplier	Head Supplier	Neck Supplier	Flipper Guide Supplier	Flipper Hub Supplier	Flipper Track Supplier
Contracted Delivery Window (Days)	30	15	15	15	10	10	10	20	10	10	10	10	5
Batch Size (units)	1	1	1	5	3	30	2	10	3	3	10	1	30
Scheduling Delay Mean (hours)	0	0	0	0	0	13	12	22	0	0	10	10	10
Scheduling Delay Std (hours)	0	0	0	0	0	5	5	5	0	0	2	2	2
Test Coverage (%)	95	75	75	99	95	99	95	70	75	95	95	95	95
Manufacturing Quality (%)	95	97	98	80	99	90	80	80	95	95	95	95	90
Design Quality (%)	99	99	99	99	99	99	99	90	99	99	99	99	99
Line Flow Time Mean (hours)	12	24	4	3	3	1	17	20	6	12	3	0.5	1
Line Flow Time Std (hours)	4	6	0.4	0.1	0.2	0.2	0.4	6	2	2	0.3	0.1	0.1
Tact Time (hours)	3	4	1	0.4	1	0.7	4	0.4	1	1	1	0.5	0.5
Rework Flow Time Mean (hours)	36	40	3	1	1	0.3	5	15	10	4	0.4	0.3	0.1
Rework Flow Time Std (hours)	40	5	0.2	0.1	0.1	0.1	2	3	4	1	0.1	0.1	0.1
WACC (%)	15	15	10	12	8	8	10	13	10	10	12	13	8
Margin (%)	25	10	10	10	12	12	8	30	15	10	8	8	8
FGI Storage Cost (year)	500	300	100	35	30	5	30	500	100	30	10	10	10
Replacement Part Cost ($)	500	450	300	40	20	10	100	300	140	130	30	30	30

Model Parameter	Lead CM	Chassis Assembler	Head and Neck Assembler	Battery Supplier	Flipper Assembler	Chassis Track Supplier	Side Panel Supplier	Electronics Box Supplier	Head Supplier	Neck Supplier	Flipper Guide Supplier	Flipper Hub Supplier	Flipper Track Supplier
Late Penalty ($ / unit)	300	300	120	30	10	10	100	30	50	50	20	15	10
Starting Cash ($)	5M	5M	5M	5M	5M	5M	5M	5M	5M	5M	5M	5M	5M
Labor Rate ($/hour)	24	24	24	15	15	13	25	30	25	25	13	13	13
Equipment Rate ($/hour)	30	30	20	50	10	90	250	750	10	10	10	90	90
Fixed Labor Time (hours / batch)	0	0	0	0	0	0	0	0	0	0	0	0	0
Variable Labor Time (hours / unit)	12	40	3	3	1	0.3	0.3	12	8	4	1	1	0.3
Fixed Equipment Time (hours / batch)	0	0	0	0	0	0	0	0	0	0	0	0	0
Variable Equipment Time (hours / unit)	6	6	1	1	0.2	0.1	1	1	2	0.2	0.2	0.3	0.1
Average tear down and rebuild labor (hours)	24	40	3	1	1	0.3	5	15	10	4	0.4	0.3	0.1
Average tear down and rebuild equip. (hours)	3	0	0	0	0	0.1	1	2	0	0	0.2	0.2	0.2

Shipping times, costs, and initial quantities:

Material	Source	Destination	Quantity (units)	Shipping Fixed Cost ($/ship)	Shipping Variable Cost ($/unit)	Travel Time Mean (hours)	Travel Time Std (hours)	Storage Cost ($ / year)	Initial ROP (units)	Initial EOQ (units)
Head and Neck	Head and Neck Assembler	Lead CM	1	110	30	4	0	300	5	5
Chassis	Chassis Supplier	Lead CM	1	110	30	4	0	300	5	5
Flippers	Flipper Assembler	Lead CM	1	40	10	4	0	200	5	5
Battery	Battery Supplier	Lead CM	4	40	10	4	0	50	20	15
Chassis Track	Chassis Track Supplier	Chassis Assembler	2	10	10	2	0	100	15	5
Chassis Side Panels	Side Panel Supplier	Chassis Assembler	1	50	30	2	0	300	15	5
Electronics Box	Electronics Box Supplier	Chassis Assembler	1	110	50	2	0	300	15	5
Head	Head Supplier	Head and Neck Assembler	1	50	15	4	0	300	15	5
Neck	Neck Supplier	Head and Neck Assembler	1	30	20	4	0	300	15	5
Flipper Guide	Flipper Guide Supplier	Flipper Assembler	1	30	30	4	0	30	15	5
Flipper Hubs	Flipper Hub Supplier	Flipper Assembler	1	25	30	4	0	30	15	5
Flipper Track	Flipper Track Supplier	Flipper Assembler	1	25	30	4	0	30	15	5
Battery Components	Battery Stock Components	Battery Supplier	1	0	30	4	0	30	50	50
Track Rubber	Track Rubber Stock	Chassis Track Supplier	1	25	40	4	0	30	50	50
Aluminum	Aluminum Stock	Side Panel Supplier	1	25	40	4	0	30	50	50
Electronics Components	Electronics Stock Components	Electronics Box Supplier	1	25	40	4	0	30	50	50

Material	Source	Destination	Quantity (units)	Shipping Fixed Cost ($/ship)	Shipping Variable Cost ($/unit)	Travel Time Mean (hours)	Travel Time Std (hours)	Storage Cost ($ / year)	Initial ROP (units)	Initial EOQ (units)
Head Components	Head Component Stock	Head Supplier	1	25	40	4	0	30	50	50
Neck Components	Neck Component Stock	Neck Supplier	1	25	40	4	0	30	50	50
Flipper Guide Plastic	Flipper Guide Stock Plastic	Flipper Guide Supplier	1	25	40	4	0	30	50	50
Flipper Hub Plastic	Flipper Hub Stock Plastic	Flipper Hub Supplier	1	25	40	4	0	30	50	50
Track Rubber	Track Rubber Stock	Flipper Track Supplier	1	25	40	4	0	30	50	50

References

A.T. Kearney. "Selecting a Country for Offshore Business Processing: Where to Locate," 2003

Abu-Khalil, Ramy, "Developing a Unified Manufacturing and Sourcing Strategy in a Multi-Business Unit Engineering Firm", MIT 2005

Anupindi, Ravi, "Managing Business Process Flows," Prentice Hall, 1999.

Bureau of Economic Analysis, Summary Statistics 1997-2005 available from http://www.bls.gov/

Bureau of Labor Statistics, Summary Statistics 2005 and Summary Statistics 2001-2005 Available from www.bls.gov

Byrnes, J.L., "Dell Manages Profitability, Not Inventory", HBSWK, Jun 2003

Byrnes, J.L., Roy D. Shapiro, "Intercompany Operating Ties: Unlocking the Value in Channel Restructuring", Harvard Business School Working Paper Series, No. 92-058

Christensen, Clayton. Raynor, Michael, *The Innovator's Solution: Creating and Sustaining Successful Growth.* Harvard Business Press, 2003

Christensen, Clayton M., Raynor, Michael E., "Why Hard-Nosed Executives Should Care About Management Theory", Harvard Business Review, September 2003

Fine, Charles H., *Clock Speed; Winning Industry Control in the Age of Temporary Advantage.* Perseus Books, 1998.

Fine, Charles H., Daniel E. Whitney, "Is the Make Buy Decision a Core Competence", Feb 1996

Hayes, Robert H., Pisano, Gary P., "Beyond World-Class: The New Manufacturing Strategy", Harvard Business Review, Jan-Feb 1994

Herper, Matthew. 2005. "Robots of War, Future Tech." Available from http://www.frobes.com

Hochberg, Peter von, Rodrigues, Marcello, Grenon, Georgiana, "Who Manages Manufacturing?" BAH Strategy and Business, Summer 2006

iRobot Press Release 18 May 2005, case 05-068 Available from http://www.irobot.com/sp.cfm?pageid=86&id=147&referrer=169

Kaplan, Robert S., Norton, David P., "The Balanced Scorecard: Measures That Drive Performance", Harvard Business Review, Jul-Aug 2005

Khosravani, Soheila S., "Providing a Framework for Virtual Manufacturing: An Assessment of OutSourcing in the Computer Industry", MIT 1995

Kimber , Michael A., "Definition and Implementation of a Visual Inventory Management System" MIT 1999

Kotz, Nick, "The Chesapeake Bay goose hunt, the beautiful secretary, and other ways the defense lobby got the B-1," Washington Monthly, 1988. Available from http://www.encyclopedia.com/doc/1G1-6351543.html

Luthje, Boy. "Electronics contract manufacturing: Global production and the international division of labor in the age of the Internet," Industry and Innovation, Dec. 2002

M.D. Springer, Dale, Melvin, *The Algebra of Random Variables*, Wiley, 1979.

McDonald, Ian A., "Using an Extended Enterprise Model to Increase Responsiveness", MIT, 2006

Narayanan, V. G., Raman, A., "Aligning Incentives in Supply Chains", Harvard Business Review, Nov 2004

National Center for Education, Digest of Education Statistics, 2005 Available from http://www.nces.ed.gov/programs/digest/d05/tables/dt05_112.asp

Prahalad, C.K., Gary Hamel, "The Core Competence of the Corporation" HBR, May – June 1990.

Reardon, Elaine, *The Department of Defense and its Use of Small Businesses: An Economic and Industry Analysis*, RAND 2005

Rigoni, Jennifer A., "Assessing Materials Risk in Purchasing Electronic Components During Product Design", MIT 2002

Ross, Sheldon M., *Introduction to Probability Models.* Academic Press, 1972

Rundquist, Barry S., *Congress and Defense Spending: the Distributive Politics of Defense Spending*, University of Oklahoma Press, 2002.

Russell, Paul G., Kroph, William C., "Hewlett-Packard's Packaging Supplier Evaluation Process and Criteria", HP, Feb 1999

Shao, Min, "Digital Factory: Real Time Information System Implementation in a Traditional Manufacturing Environment", MIT 2006

Skroggs, Steven. *Army Relations with Congress: Thick Armor, Dull Sword, Slow Horse.* Praeger, 2000

Socolovski, Alberto, "When Your Customers Become Your Competitors. (Contract Manufacturing)" Electronic Business Buyer, Aug. 1993

Spear, S., H. Kent Bowen, "Decoding the DNA of the Toyota Production System", Harvard Business Review, Sep-Oct 1999

Tax Foundation, "State Business Tax Climate Index 2006," Available from http://www.taxfoundation.org/taxdata/show/1371.html

Tully, S., "Modular Corporation", Fortune, Feb 1993

US Census, Financial Data, 2005 available from http://www.census.gov/

US Census, Education Attainment available from http://www.census.gov/population/socdemo/education/cps2005/tab01-01.xls

Vasovski, Steven. "A Global Sourcing Strategy for Durable Tooling." MIT, 2004

Watson, Kevin J., Blackstone, John H., Gardiner, Stanley C., "The evolution of a management philosophy: The theory of constraints", Journal of Operations Management, 2006

Wickham, Skinner, "Manufacturing – Missing link in corporate strategy", Harvard Business Review, May-Jun 1969

Wu, Qian, "A Supply Chain Strategy for Digital Camera Products", MIT 2001

Government Documents

GAO-05-428T, "Defense Acquisitions, Future Combat Systems Challenges and Prospects for Success", 2005

GAO-04-715, "Opportunities to Enhance the Implementation of Performance-Based Logistics", 2004

GAO-06-839, "Preliminary Observations on DOD's Acquisition of Technical Data to Support Weapons Systems", 2006

H. CON. RES. 113, 110th Congress, 1st Session

H.R.5408.IH, Sec. 220(a), 106th Congress

H.R.5803.ENR, "Department of Defense Appropriations Act", 1991

H. Report. 109-119, "House Defense Appropriate Act Committee Report", 2006

Tyszakiewicz, Mary T., Daggett, Stephen, "A Defense Budget Primer." CRS Report RL30002

Web Sites

[1] http://www.time.com/time/magazine/article/0,9171,919040-1,00.html

[2] http://www.army.mil/fcs/factfiles/sugv.html

[3] http://www.defenselink.mil/admin/about.html

[4] http://www.dnb.com

[5] http://www.house.gov/house/CommitteeWWW.shtml

[6] http://armedservices.house.gov/subcommittee.shtml

[7] http://www.globalsecurity.org/wmd/systems/b-1a.htm

[8] http://www.madehow.com/Volume-2/Ammunition.html

[9] http://home.howstuffworks.com/revolver2.htm

[10] http://www.recguns.com/Sources/VIIA2.html

[11] http://www.cartridgecollectors.org/glossary.htm

[12] http://www.rohs.gov.uk

[13] http://www.proximityone.com

[14] http://www.army.mil/fcs

[15] http://wrds.wharton.upenn.edu