Center for Excellence

in

Logistics and Distribution

National Science Foundation sponsored Industry/University Cooperative Research Center

University of Arkansas/Clemson University

Final Research Report

Project #: CDP13-VISOR

Variation Identification System for Operational Risks in Inventory Systems

Center Designated Project

Variation Identification System for Operational Risks in Inventory Systems

Research Team: Manuel D. Rossetti (PI), Kevin Taaffe (PI), Keith Webb (RA), Kyle Lassiter (RA)

Sponsor: **CELDi CDP** | ***Thrust Area:*** **Inventory**

Keywords: reorder point reorder quantity policies, uncertainty, Monte Carlo simulation

Problem in Context: The main goal of this research is to review methods and software to deal with less than perfect information in the supply chain and to develop a software tool to enable analysts to better understand how data uncertainty can lead to risk in the supply chain. Current inventory models assume perfect knowledge about costs and the distribution of future demands and lead time. However, it is usually the case that the level of knowledge is less than perfect due to estimation errors or imperfect data. Failing to account for data uncertainties can lead to failing to meet customer service standards or to excessive supply chain costs.

Technical Approach: The project developed a web application that runs Monte Carlo simulations of (r,Q) inventory models. The application uses random distributions to represent uncertain parameters in the model. Algorithms developed in a previous CELDi project for optimizing and reporting the performance of (r,Q) models were used throughout the application. The Vaadin framework was used to develop the graphical user interface in Java.

Results: A web based software tool implemented in Java that allows the user to easily set up and run Monte Carlo simulations of inventory models with uncertain cost and demand parameters.

Broader Value to CELDi Members: This proposed research has the broader impact of allowing analysts to evaluate if an item poses a significant supply chain risk, estimate the value of improved information in the supply chain, and determine which inventory model parameter is most contributing to performance uncertainties.

Future Research and Potential Extensions: Adding file input and output. Adding modules to allow simulations of periodic review and other types of continuous review inventory models. Improving the visual presentation of results.

Variation Identification System for Operational Risks in Inventory Systems

Manuel D. Rossetti, Ph. D., P. E. (Principal Investigator),

Kevin Taaffe, Ph. D. (Principal Investigator)

Keith Webb (Research Assistant)

Kyle Lassiter (Research Assistant)

This report is intended for distribution within the

Center for Excellence in Logistics & Distribution (CELDi).

Variation Identification System for Operational Risks in Inventory Systems

Manuel D. Rossetti , Ph. D., P. E., Keith Webb
Department of Industrial Engineering
University of Arkansas
4207 Bell Engineering Center
Fayetteville, AR 72701

Kevin Taaffe, Ph. D., Kyle Lassiter
Department of Industrial Engineering
Clemson University
130A Freeman Hall, Clemson University
Clemson, SC 29634-0920

Executive Summary

Supply chain analysts are often faced with missing, incomplete, inconsistent, or contradictory data. Failing to account for less than perfect data can lead to situations where items in the supply chain fail to meet customer service expectations or exceed expected costs.

This report begins with a summary of current methods to incorporate data driven risk into inventory modeling solutions. It specifically addresses data variability, validity, inconsistency, and multiple data sources. It also has an overview of the current state of the art software solutions that address data uncertainty in inventory modeling.

A web application was developed to allow users to quickly and easily run Monte Carlo simulations to determine how parameter uncertainty affects the performance of (r, Q) inventory models. The application has three modules that run different simulations and present the results:

- The Parameter Comparison modules allows a user to compare the effects of different parameter values, different levels of uncertainty, or different reordering policies on inventory performance

- The Rapid Comparison module runs several simulations on different levels of uncertainty. By using this module, the user can quickly determine which uncertain parameter is driving the most uncertainty in model performance.
- The Policy Comparison module optimizes the reordering policy for each simulation run and determines the probability that a particular policy is optimal. This tells the user which policies are most likely to be optimal for a given set of uncertain parameters.

Table of Contents

List of Tables

List of Figures

1 Introduction

Logistics analysts rely on models such as base stock, (r, Q), or economic order quantity models, to set reordering policies. These models assume that the analyst has perfect information about costs and the distribution of future demand. However, this is rarely the case. Supply chain data can be corrupt, uncertain, or completely missing. It can also be overwhelming and complex. Either case introduces a significant risk of failing to meet customer service expectations or of excessive costs in the supply chain.

Current software solutions do not take into account sources of variability. This project identified methodologies to find items that pose a supply chain performance or cost risk and identified gaps in existing software solutions. The project also developed a software tool to help analysts identify items with substantial risk and to determine the value of more certain information.

This report first presents the literature review and gap analysis of existing methodologies and software. Next, the report documents the development of the software tool. Finally, there is a user's guide and a guide for future developers of the software tool.

2 Literature Review and Gap Analysis

A number of factors contribute to supply chain performance, both internally and externally. For this Center Designated Project (CDP), the research focus is on the role of data in supply chain performance. As defined in the CDP proposal, the specific plan was to investigate developing "methodologies and software tools that can help identify inventory items (e.g. spare parts) that pose risks in terms of performance and cost, and help quantify the risks both visually and statistically," via a data analysis approach. The general idea is to help logistics analysts understand: the effect of missing, corrupted, or uncertain data; massive amounts of data from one

or more sources and their role in supply chain decisions; and the identification of high-risk supply chain items from a data analysis approach. The underlying assumption identified in this CDP is that sources of variability that affect the input parameters for inventory stocking decisions are not being adequately considered.

The IAB requested an investigation into the current state of this proposed research area as the first step of this CDP. A report on the topic coverage in relevant academic literature and existing software solutions is compiled here. The findings are organized in two sections corresponding to the first two tasks defined in the CDP. The first section is the literature review summarizes how data-driven risk is incorporated into inventory modeling solutions, and specifically addressing: data variability, data validity, data inconsistency, and multiple data sources. The second section is an overview of the current state of software solutions that address data uncertainty in inventory modeling, and discusses the gap(s) between the desired outcomes of the CDP and the current inventory modeling software. A summary and gap analysis is presented in the conclusions section.

2.1 Literature Review

A thorough review of academic literature was conducted, covering numerous recognized academic and industry journals such as Operations Research, Interfaces, Production and Operations Management and many others. The findings are categorized into the four subtopics defined in the CDP.

2.1.1 Data Variability/Inconsistency

The major topic in this category, and perhaps in all supply chain literature, is the definition and subsequent analysis of the bullwhip effect, and its impact on inventory levels. An article by Chopra and Sodhi (2004) is a classic example that details the dangers of the bullwhip effect on

inventory performance; they also suggest inventory pooling to reduce supply chain risk in high cost situations. Chen et al. (1998) state that the bullwhip effect can be reduced by centralizing customer demand information. There are hundreds of other works published that discuss ways to reduce and quantify the bullwhip effect via a myriad of approaches. One such article by Chatfield et al. (2004) "find that lead-time variability exacerbates variance amplification in a supply chain," and that "information sharing reduces total variance amplification" while better information quality reduces the bullwhip effect. Lee, So, and Tang (2000) agree, and add that information sharing "could provide significant inventory reduction and cost savings to the manufacturer," especially when the demand correlation over time or demand variance within each time period is high, or when lead times are long (like in most modern high-tech products). There are a few authors, however, who question the magnitude of the influence on the bullwhip effect caused by information distortion sources, which are things such as data variability and inconsistency (Niranjan et al. (2011)). One interesting article by Zhang (2005) also proves that, when using minimum mean squared error (MMSE) forecasting techniques, "a delay in demand information…dampens the bullwhip effect"; sometimes timely demand information can cause unintended consequences! Moving on from the bullwhip effect debate, Wan and Evers (2011) found that higher on-hand inventories and lower stockout rates existed at the retailer level in two and three-retailer supply chains, compared to single retailer supply chains in a form of the classic Beer Game model. They also found that implementing last period forecasting in supply chains generally led to worse results in on-hand inventories and stockout rates, than if no forecasting method was used at all.

There is a wealth of literature in the field of inventory forecast modeling under uncertainty as well. Akcay et al.'s paper titled "Overcoming the 'Triple-Threat' in Managing

Inventory with Limited History of Intermittent Demand" has a great literature review that discusses the different types of inventory models built and their corresponding assumptions over the years. Most of the models discussed, however, are built on parameterized distributions from limited historical data. This means that these models assume a demand distribution function is known (often normal), and then the parameters of the distribution function are estimated from the limited historical data. Croston's research in 1972 is one such example; his intermittent-demand forecasting method assumes a normally distributed demand size, sample independence, and constant probabilities. More recently, Ramamurthy et al. (2012) worked on an inventory model with an unknown demand shape parameter, while Chu et al. (2008) worked on a model with unknown demand shape and scale parameters. One of the earliest works in this area can be found in Hayes 1969, which quantifies the inaccuracies in estimated inventory targets via the expected total operating cost (ETOC) concept, assuming normally and exponentially distributed demands.

On the other side, Akcay et al. did discuss some nonparametric approaches to inventory forecasting as well. Nonparametric approaches use the empirical distribution of historical data in their forecasts, such as in Willemain et al. (1994). Huh et al. (2011) goes further and uses a Kaplan-Meier estimator for their sample stochastic inventory control problem, which they formulate using an unknown distribution containing sales data only from their example problem. In a work unrelated to Akcay et al.'s, Ruiz-Torres and Mahmoodi (2010) contend that assuming a normal demand distribution during lead time often leads to higher than necessary inventory levels and safety stocks. They subsequently verify that a different method, such as an expected value reorder point (EVR) method, which "considers demand and lead time variability by focusing on historical data," yields improved supply chain performance.

Solyah et al. (2012) introduce two robust mixed-integer programming (MIP) formulations to tackle an inventory routing problem (IRP), with great results in terms of improved processing time to optimality. They only assume that the demand probability is independent and symmetric, without knowing its distribution, in their research. The results of their research are encouraging to see, especially given the large sample problem used in their work. They reference several other similar research works in their literature review, pertaining to various uncertain demand problems; however none of them cover the topics of data accuracy or multiple (and perhaps conflicting) data sources. Van der Vorst & Buelens (2002) did a series of case studies that found supply chain information system uncertainties (such as multiple/conflicting data sources) to be a significant source of supply chain uncertainty, however. Shukla et al. (2012) tackle the problem of rogue seasonality detections in supply chains in an article by the same name. In it, they devise a method to identify and quantify rogue seasonality in supply chain management via a "rogue seasonality index." Rogue seasonality is a false seasonal forecast trend caused by internal forecasting practices, not by actual demand data, and thus reducing it improves supply chain performance.

There are many other works that study other areas of uncertainty in supply chain systems. Lim (2013) develops a robust optimization model (ROM) that determines "a joint optimal bundle of price and order quantity for a retailer in a two-stage supply chain under uncertainty of parameters in demand and purchase cost functions." These uncertain parameters are characterized by ellipsoids in the ROM model, which is subsequently converted into a convex optimization problem for solving. According to Lim, this problem is solved "efficiently and effectively" to global optimality via interior-point methods, using "high-quality convex optimization software like CVX." CVX is a Matlab-based software used for disciplined convex

programming, and while high-quality, is not a user-friendly "off-the-shelf" software that can be used to immediately understand complex uncertainties in inventory management problems.

Fuzzy set theory has also been applied to model supply chain uncertainty. Fuzzy sets are sets of objects with an associated property that indicates the degree to which the object belongs to the set ("Fuzzy Set"). In other words, they are another way to represent probability distributions in mathematics, since all the objects' set memberships must total to 1. There are several pieces of research studying this. One is by Wang and Shu (2005), who developed a fuzzy decision model "that allows decision makers to express their risk attitudes to evaluate SC (supply chain) performances and select suitable SC inventory strategies." In other words, a fuzzy decision model is useful to decision makers who are trying to examine a supply chain with a lack of data certainty and/or historical data, as well as input their opinion of the supply chain risk into the model. This risk can be qualified as pessimistic to optimistic, and supply chain decision makers can model and/or make strategy decisions in a corresponding manner. The authors used supply chain parameters demand, lead time, material transit time, and production time in their model, and used six-point fuzzy numbers in their model formulation. In another study, Handfield, Warsing, and Wu (2009) create a fuzzy set model as well, using triangular fuzzy sets and an object function that is defuzzified. The uncertainties they use in their model are demand, lead time, supplier yield, and penalty cost; comparable to Wang and Shu (2005). The authors do point out that fuzzy set models have an advantage over some other model techniques, since the former can be used with nearly any empirical demand or lead time distribution.

One article provided some measures for supply chain performance under data uncertainty. Fang, Zhang, Robb, and Blackburn (2013) develop analytical expressions for marginal values of lead time mean, lead time variance, and demand variance in a supply chain, from which the

effectiveness of the supply chain is determined via the ratio of the marginal value of lead time mean to the marginal value of lead time variance. They find that this effectiveness is strongly dependent on whether they are independent or correlated. When lead time mean and variance are independent, they found that "the ratio of the marginal values of lead time mean and lead time variance is determined by the coefficient of variation of demand." On the other hand, when lead time mean and variance are correlated, the ratio "is a function of the coefficient of variation of the lead time." Finally, they reference Paknejad et al. (1992) and their conclusion that lead time variance was more costly than demand variance to the supply chain.

Data inconsistency and variability can also be an issue in parameters other than lead time demand. Many researchers have focused on the deterministic Economic Order Quantity model and its sensitivity to unknown or changing parameters. This discussion presents some of this research and how it can be compared to stochastic inventory models. Lowe and Schwartz (1983) examined the Economic Order Quantity model with parameters whose precise values are unknown, but fall within certain ranges. They used two approaches when determining an optimal ordering quantity, minimize the maximum error or minimize the expected error. The paper presents formulas for the optimal order quantity with uncertain parameters. The paper also shows that EOQ models are very robust to errors in parameter estimation. Large errors in estimation result in small errors in total cost. Zheng (1992) conducted a sensitivity analysis of the (r, Q) model with fixed lead times and compared it to the EOQ. He found that the optimal order quantity of the (r,Q) model is always higher than the EOQ for the same cost parameters, although at large order quantities, the difference is small. The cost function of the (r, Q) model as a function of Q is flatter than the EOQ cost function, and the relative cost increase caused by using the EOQ order quantity is no more than 12.5%. Federgruen and Wang (2013) derived

monotonicity conditions for the optimal reorder point and quantity and the optimal cost. If the functions of these values are monotonic, the model user can determine how under or overestimating the model parameters will affect the optimal ordering policy or total cost.

One final item worth mentioning is the Supply Chain Operations Reference (SCOR) model; a major downstream supply chain performance measurement product, with many major companies listed as members of the Supply Chain Council (SCC), who is the owner of the SCOR model. A study by Cirtita and Glaser-Segura (2012) surveyed the use and effectiveness of the SCOR model's performance metrics in downstream supply chains. The metrics they specifically looked at were supply chain delivery reliability, supply chain responsiveness (i.e. order fulfillment), supply chain flexibility, supply chain costs, and supply chain asset management efficiency (i.e. asset turns). They found that many supply chains use some form of a SCOR model, and that many of them used the standard form of SCOR for their performance metric system. The latest SCOR software edition is SCOR 11.0 (http://supply-chain.org/scor).

2.1.2 Multiple Data Sources

Another major topic of interest to VISOR is the subject of multiple, conflicting, and/or missing data sources in a supply chain. A general term for the process of combining these multiple sources (which sometimes have conflicting or missing data amongst the ensemble data set members) is data fusion. A technical article by Bleiholder and Naumann (2009) details many different data fusion methods for computerized databases, as well as many other things related to managing the database(s) before and after any data fusion processes. This includes data transformation (matching data formats & layouts), duplicate detection (identifying and modifying/deleting duplicate data records after fusion), and data conflict management. Data conflicts are defined as either schematic (formatting), identity, or value/parameter conflicts, such

as uncertainties (null vs. non-null values) and contradictions (differing non-null values). The authors then detail nine different strategies for handling data conflicts; a table from their work summarizing this is shown below. The classification column describes their approach to handling the data conflict.

Table 1 – Data Conflict Strategies in Data Fusion

Strategy	Classification	Short Description
PASS IT ON	ignoring	escalates conflicts to user or application
CONSIDER ALL POSSIBILITIES	ignoring	creates all possible value combinations
TAKE THE INFORMATION	avoiding, instance based	prefers values over null values
NO GOSSIPING	avoiding, instance based	returns only consistent tuples
TRUST YOUR FRIENDS	avoiding, metadata based	takes the value of a preferred source
CRY WITH THE WOLVES	resolution, instance based, deciding	takes the most often occurring value
ROLL THE DICE	resolution, instance based, deciding	takes a random value
MEET IN THE MIDDLE	resolution, instance based, mediating	takes an average value
KEEP UP TO DATE	resolution, metadata based, deciding	takes the most recent value

Bleiholder and Naumann also offer the option for uncertain data to be explicitly represented in the databases. They state that "in an OLAP scenario Burdick et al. (2005) use probability distribution functions to model both uncertainty on the value, as well as imprecision according to a dimension hierarchy," in order to present the one highest probability value to the supply chain manager, or to allow managers to choose what values to use based on the probability of each value. This differs from the techniques listed in Table 1, which are an automated process.

In Table 2, Bleiholder and Naumann also describe the data conflict-handling characteristics of some of the different computer systems in database management. More follow-up research can be done on the articles referenced in the "data conflict awareness" column in the portion called "Table X."

In addition, the authors also mention that "commercially available database management systems from vendors such as IBM, Oracle, or Microsoft, to name just a few, can usually be used to integrate data from different sources," instead of the generally lesser known systems

mentioned in Tables X and XI. They also state that "most integration systems only cope with schematic conflicts, as this has been the field with the longest research history." This is a very thorough article with much useful information therein.

Table 2 – Data Conflict-Handling Characteristics across Many Systems

Table X. General Conflict-Handling Properties of Systems (conflict types and data conflict awareness)

System	Conflict Types	Data Conflict Awareness
Multibase	schematic, data	yes, [Dayal 1983]
Hermes	schematic, data	yes, [Subrahmanian et al. 1995]
Fusionplex	schematic, object, data	yes, [Motro and Anokhin 2006]
HumMer	schematic, object, data	yes, [Bleiholder and Naumann 2006]
Ajax	schematic, object, data	yes
TSIMMIS	schematic, data	yes, [Papakonstantinou et al. 1996]
SIMS/Ariadne	schematic, data	yes, partly, [Ambite et al. 2001]
Infomix	schematic, data	yes, [Leone et al. 2005]
Hippo	schematic, object, data	yes
ConQuer	schematic, object, data	yes
Rainbow	schematic, object, data	yes
Pegasus	schematic, data	no
Nimble	unknown	unknown
Carnot	schematic	no
InfoSleuth	schematic	no
Potter's Wheel	schematic	no

Table XI. Solutions to Schema Matching/Mapping and Duplicate Detection in Integration Systems

System	Schema Matching/Schema Mapping	Duplicate Detection
Multibase	hand crafted	assumes global id
Hermes	hand crafted	assumes global id
Fusionplex	hand crafted, learned in Autoplex	assumes global id
HumMer	semi-automatically	global id generated by duplicate detection
Ajax	hand crafted	global id generated by similarity measure
TSIMMIS	wrapper generation	assumes global id
SIMS/Ariadne	wrapper generation	mapping table
Infomix	hand crafted	assumes global id
Hippo	hand crafted	assumes global id
ConQuer	hand crafted	assumes global id
Rainbow	hand crafted	assumes global id
Pegasus	hand crafted	assumes given mapping table
Nimble	unknown	mapping table, generated
Carnot	hand crafted	assumes global id
InfoSleuth	hand crafted	assumes global id
Potter's Wheel	n/a	n/a

A second source focusing on data fusion is a dissertation by Wang (2010), in which the focus is modeling supply chain dynamics via simulation with data fusion. Wang inputted real time supply chain data into the simulations, then applied Kalman Filtering, a "well-known tool in the control community for stochastic state and parameter estimation…from noisy (uncertain) measurements." This "makes an overall best estimate of a state based on all information"

available. This method is primarily applied for control sensing purposes; i.e. continuous monitoring to identify when a system is out of tolerance, or if a serious supply chain disruption is beginning to occur. The term used to describe solutions that "sense" these supply chain disruptions or demand fluctuations in near real time is "supply network event management." For systems with uncertain data sources, Ensemble Kalman Filtering (EnKF) is applied, which "uses Monte Carlo simulation to approximate the nonlinear state transition function." In short, the multiple or uncertain data source values are estimated using EnKF, then used in the simulation model. When combining data from multiple sources in the supply chain simulation model, it is inserted into the model in the form of a vector, representing the possible values of the parameter based on the data sources. Wang also finds that for noiseless (certain) shared demand information, a minimum-mean square error (MMSE) forecast is able to be explicitly derived "in a multi-echelon supply chain facing non-stationary end-consumer demand." However, it of course gets more complicated for noisy demand data, and Wang discusses an alternate forecasting approach in those situations in his work. Wang also mentions that, "as pointed out by Treharne and Sox (2002), research that considers both non-stationary demand and partial information is relative (sic) limited in the literature compared to the study of full information and stationary supply chain." Perhaps this is an area that modern research is still lacking, as Wang's dissertation was written in December 2010.

In a related article, Dey (2003) created a decision model to effectively match records in a data warehouse (an integrated set of databases) during data consolidation. Dey used a heuristic solving approach with a cost minimization objective function, with cost components of user error, cost of human effort, and closeness between two records from two different data sources (Dey 2003). The model was tested with real data examples and determined to be effective at

identifying identical entities from separate databases, even with varying degrees of formatting differences and/or data errors. The author was thus confident that the model would perform well in a real-world environment, although it had not actually been tested that way in the research.

2.1.3 Data Validity

In situations where perhaps complete or accurate data fusion is not possible, then a quantitative measure of a data source's quality could be useful. Dey and Kumar (2013) research the problem of quantitatively determining the data quality of a database source query (or multiple sources and queries, including conjunction and disjunction operations). They restrict their domain to only identifying random errors in the database, and thus assume that systematic data recording/reporting errors were already being properly dealt with by the organization. However, they relax previous researchers' assumption that the underlying query data distribution is not affected by random errors. They consider three metrics in their data quality determination formulation: data accuracy, mismembership, and incompleteness. Data accuracy is defined as the probability that an attribute stored in the data is an accurate representation of the corresponding real-world value. Mismembership is defined as the number of spurious objects in the data set (expressed as a fraction of the total objects in the data set), and incompleteness is defined as the number of real-world objects missing from the data set (expressed as a fraction as well). The researchers define type I (when a tuple is missing from the query results) and type II (when a tuple is incorrectly included in the query results) errors as well, and calculate the expected numbers and corresponding probabilities of each error type to then use in the metric calculations. Several heuristics and sampling methods are employed to estimate some parameters used in the metric calculations. Ultimately, the person who queries the database can then have some measure

of the quality of the data based on these three defined metrics; this could be very helpful to a supply chain manager who is uncertain as to the quality of his inventory database.

2.1.4 Literature Review Observations

Data fusion may not always be applicable, particularly if the multiple data sources are of uncertain accuracy. Relevant research focuses only on combining or fusing data sources, not generating a methodology to select a superior data source as the primary one for decision making purposes.

2.2 Software Review

The second task outlined in the VISOR CDP proposal was a review of existing software solutions that address issues of data uncertainty in inventory modeling. A web search of relevant software was conducted initially. Requests for further details on the more advanced and successful software solutions were delivered via email, but no responses were ever received. A survey was distributed to CELDi members soliciting their knowledge on existing supply chain software solutions, and one reply was received from Boeing. The key questions investigated during this phase were:

- Can existing software tools identify and measure sources of variability?
- Can existing software tools quantify the effect of variability on the decision making process?
- Do existing software tools enable analysts to easily communicate the risks of uncertainties in a manner that less technically sophisticated decision makers can understand?

Modern day technological advances are allowing for unprecedented amounts of computing power to be available to tackle some immense, complicated supply chain inventory problems. At the same time, the advent of modern technology has allowed for equally unprecedented amounts of data collection and storage for analysis. Thus, the subject of “Big

Data" and software-based analytics has become very popular in the last 10 years. However, the intersection between this subject and supply chain inventory management might be surprisingly small. In an editorial published just this past June (2013), Waller & Fawcett claim there are a "myriad of opportunities for research where supply chain management (SCM) intersects with data science, predictive analytics, and big data" since a "pure data mining approach often leads to an endless search for what the data really say." The authors then call for research to be conducted related to supply chain management and big data, claiming there is a dearth of work in this area. VISOR would fall squarely in this dearth.

2.2.1 Big Data

Stadtler & Kilger in their book "Supply Chain Management and Advanced Planning: Concepts, Models, Software, and Case Studies" discuss collaborative supply chain software present in SAP programs. These include SAP APO (Advanced Planner and Optimizer) and SAP ICS (Inventory Collaboration Hub), the latter of which supports Supplier Managed Inventory (SMI) and Release Processing (buyer-driven replenishment via SAP R/3). Supply chain performance monitoring is also available via KPI's (key performance indicators), via SAP APO Plan Monitor or SAPBW (SAP Business Information Warehouse). For businesses familiar with SAP, these programs may already be in use.

However, with big data needs growing rapidly, several other relatively new software systems and their companies have recently sprung up to handle big data analytics. In a recently published book by the International Institute for Analytics (Davenport 2012), some of these major entities are listed: SAS, Intel, Accenture, Teradata, IBM, Dell, and SAP.

SAS is arguably the leader in big data analytics, as evidenced by their products advertised publically on their website as well as their wide range of software options. Relevant to VISOR,

these include: risk management software, manufacturing applications software, demand forecasting software, supply chain intelligence software, and visual analytics software. One software advertised even works with SAP APO, combining SAS forecasting with SAP planning, for allegedly superior supply chain planning results.

Accenture is another large provider of big data analytics software. Their relevant products listed on their website include analytics software, information management software, data quality services, and risk management services. The goal of their information management software is to "improve data quality through a comprehensive approach to data management, including capturing required information, improving data accuracy, and optimizing all data management areas."

Not surprisingly, the details of these software options were not publicly available, and thus their software capabilities are only understood in general terms. However, it is a safe assumption to make that these premier supply chain software providers could likely provide a customized solution to any supply chain problem. Another supply chain software provider who appears to have superficially similar products is Teradata and their suite of big data analytics softwares for transportation, distribution, and logistics applications. Intel should also be mentioned as well as the distributor of the Apache Hadoop Software for "Big Data," used by customers such as Google and Facebook with immense volumes of data to process each minute. Many other companies employ Hadoop software as the basis for their software applications as well.

2.2.2 Supply Chain Analytics Software and other useful software tools

The first part of this section is a summary of Boeing's survey response, and subsequent investigation into the software tools they mentioned. First, Llamasoft's Supply Chain Guru is a

major supply chain analytics software they use. Their website highlights three abilities for their product. First, it can find hidden inefficiencies in a supply chain, utilizing data from multiple sources. Second, it can run "what if" scenario analyses, by optimizing for different scenarios based on variations in assumptions about demand, costs, lead times, and availability. Third, it can test outcomes using simulation technology. Llamasoft also states that its software is capable of discrete event simulation (which provides probability distributions for quantity variables like demand, lead time, etc.) and data integration with error checking.

IBM's Cognos software is their version of a supply chain analytics software. Their website boasts that Cognos can "consolidate information from multiple data sources to measure supplier performance across a range of key performance indicators (KPIs)," and that it "provides insight into procurement spending, supplier performance, contract management and operational efficiency." Thus, both Supply Chain Guru and Cognos are apparently capable of handling data from multiple sources, but the exact details and methodology behind these claims is not publicly described.

Other supply chain software tools referenced by Boeing in their VISOR survey are VMetric XL, @Risk, and IBM Infosphere. VMetric XL is produced by TDFG, and is a spare parts optimization modeling software (http://www.tfdg.com/pdf/VMetric.pdf). Boeing states that this software contains a data quality attribute to characterize data risk; however, this statement cannot be verified from publicly available information about the software. @RISK is a basic risk simulation tool plug-in for Microsoft Excel, with its primary feature being its ability for the user to input data variability distributions. IBM InfoSphere is a software product that according to Boeing allows for the "integration of disparate sources of data." This matches the description

provided by the IBM website as well, which states that InfoSphere has data integration, data warehousing, master data management, big data, and information governance capabilities.

2.2.3 Data Visualization

When investigating the idea of whether existing software tools enable analysts to easily communicate the risks of uncertainties, Tableau quickly surfaced. Tableau is also a relatively young company, which specializes in data visualization software. The sample products available on their website are indeed very impressive. VizQL is the programming language at the basis of their software. It "is a visual query language that translates drag-and-drop actions into data queries and then expresses that data visually. VizQL delivers dramatic gains in people's ability to see and understand data by abstracting the underlying complexities of query and analysis." The key feature they believe has made their software so successful is the drag-and-drop, user-friendly aspect to their data visualization programs. They've based their products on five principles: easy interface, data exploration, expressiveness, visualization best practices, and database independence, and it appears to be working for them as evidenced by their rapid growth in the last three years.

Other features promoted by Tableau are data blending, which has "the ability to pull information from disparate sources into one console"; supply chain forecasting models ("Aggressive", "Conservative", "Deterministic", "Stochastic", seasonal, "Risk Optimization", or "Cost Optimization"); and "ShowMe" data visualization assistance function, which selects the most appropriate data visualization based on the requested data. All of these features, along with the very pleasant graphics and ease-of-use, and it is no surprise that Tableau software is growing rapidly in the data visualization field. However, no specifics on the forecasting model calculations were discovered in the public domain.

From an academic perspective, Kreye, Goh, Newnes, and Goodwin (2012) tested three different approaches for displaying cost-forecasting information with data uncertainty: three point trend forecast, bar chart, and fan-diagram. They included different levels of contextual information in their analysis as well, since more tended to make decision makers more aware of uncertainty in their supply chain. They concluded that the fan-diagram was "the most suitable method of graphical information display for encouraging decision makers to consider the uncertainty in cost forecasting"; 75% of test participants considered uncertainty in their decisions when using it, even if they had not used a fan-diagram before. Their example of a fan-diagram is shown below:

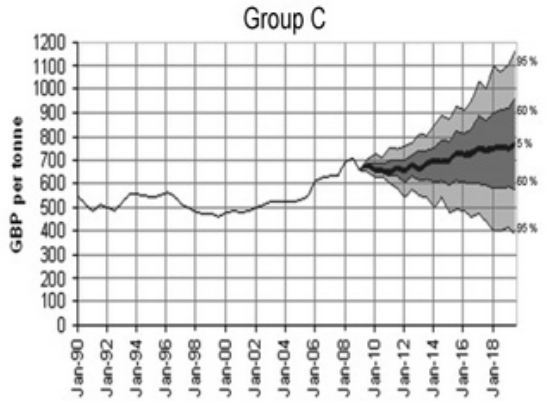


Figure 1 – Fan Diagram

2.2.4 Software Review Observations

In short, there appears to be a wealth of software available that can likely handle at least some of the questions posed by VISOR. However, there are likely gaps as well, like the one identified within the literature review. To summarize:

- Can existing software tools identify and measure sources of variability?

Yes, but public details and specifics of the software tools are insufficient to know how these tasks are done due to lack of software provider inquiry responses.

- Can existing software tools quantify the effect of variability on the decision making process?

Yes, but again things like specific statistics are not publically available.

- Do existing software tools enable analysts to easily communicate the risks of uncertainties in a manner that less technically sophisticated decision makers can understand?

Yes. Software providers like Tableau have very popular software that easily displays data and communicates information to its users.

2.3 Summary

There is a wealth of research conducted in supply chain topics, making the task of finding relevant information for this CDP somewhat challenging. It is clear that there is plenty of research done in topics like data fusion, big data, supply chain modeling, the bullwhip effect, etc. However, there were topics that had less, including the aforementioned idea of choosing one data source out of several to use in supply chain decision making, which appears to be an original concept. Also, relatively limited research is available on data quality quantification; Dey and Kumar (2013) referenced within was restricted to database query applications only. The clearest gap, however, is the idea of creating a technique to quantitatively choose one data source over others in order to make supply chain decisions. This appears to be a promising research path for VISOR going forward.

3 Web Application

Current inventory modeling software solutions do not adequately take into account data uncertainty, particularly in regards to quantifying output uncertainty. The web application was

developed to assist analysts in identifying items that pose a supply chain risk. This section details the background and development of the web application.

3.1 (r, Q) Inventory Models

The deterministic EOQ model has closed form solutions for the optimal ordering quantity. Because of this, the effects of uncertain parameters are well understood. There are not closed form solutions for the optimal reordering policy in a (r, Q) model. The optimal must be found algorithmically. For this reason, the web application implements the more complicated (r, Q) model. A summary of the (r, Q) model, how the performance of the model is determined, and how the optimal ordering policy is determined follows.

The reorder point, lot size or (r, Q) model is a continuous review inventory model. In this model, an order of size Q is placed every time the inventory position reaches the reorder point r. Demand is assumed to arrive one at a time according to some stationary stochastic process. Several costs are used in the model. Each unit purchased has an item cost. A holding charge is applied for every time period a unit is held in inventory, an ordering cost for every time an order is placed, and a backorder cost for every time period a unit is backordered. The final piece of information the model needs is the distribution of demand during lead time. The lead time demand is described using the mean and variance of both the demand and lead time. These values can be combined. If μ_D and σ_D^2 are the mean and variance of the demand distribution and μ_{LT} and σ_{LT}^2 are the mean and variance of the lead time, then the mean and variance of the combined lead time demand distribution are:

$$\mu_{LTD} = \mu_D \mu_{LT}$$

$$\sigma_{LTD}^2 = \mu_{LT}\sigma_D^2 + \mu_D^2\sigma_{LT}^2$$

These values are used to fit a probability distribution, typically a Gamma or negative binomial distribution, to describe lead time demand.

3.1.1 Evaluating (r,Q) models

Evaluating the performance of (r, Q) models requires the use of the first and second order loss functions of the lead time demand distribution. The first order loss function is

$$F^1(x) = E[\max\{0, X - x\}]$$

The second order loss function is

$$F^2(x) = \frac{1}{2}E[\max\{0, X - x\} * \max\{0, X - x - 1\}]$$

The key performance metrics for (r, Q) models are summarized in Table 3.

Table 3 – (r,Q) Performance Metrics

Name	Description	Formula
On Hand Inventory	Long run average number of units in stock	$\bar{I}(r,Q) = \frac{Q+1}{2} + r - \mu_{LTD} + \bar{B}(r,Q)$
Backorders	Long run average number of backorders	$\bar{B}(r,Q) = \frac{1}{Q}\left(F^2_{LTD}(r) - F^2_{LTD}(r+Q)\right)$
Order frequency	Long run average number of orders per time period	$\overline{OF}(r,Q) = \frac{\mu_D}{Q}$
Cost	Long run average cost	$C(r,Q) = k\overline{OF} + h\bar{I} + b\bar{B}$
Ready Rate	Probability of demand being filled immediately	$\overline{RR} = 1 - \frac{F^1_{LTD}(r) - F^1_{LTD}(r+Q)}{Q}$

3.1.2 Optimizing (r,Q) policies

The problem of finding the optimal reordering policy for the (r, Q) model can be formulated as follows,

$$minimize\ C(r,Q)$$

$$s.t.\quad r + Q \geq 0$$

$$Q \geq 0$$

$$r, Q\ are\ integer$$

The constraint $r + Q \geq 0$ ensures that when an order is placed the stock level is equal to or above zero. This means that we cannot have a permanent out of stock condition. There is not an exact, closed form solution to this problem, so a solution must be found algorithmically. The web application uses a two stage algorithm as implemented in the Inventory Web Service application. The first stage uses the algorithm developed by Federgruen and Zheng. The second stage conducts a neighborhood search around the solution of the first stage.

3.2 Monte Carlo Simulation

Extensive use of Monte Carlo simulation is made within the application. Monte Carlo methods generate a large number of random inputs to estimate the probability distribution of uncertain outputs. This is especially useful when exact values of inputs are uncertain or a model is too mathematically complex for closed form solutions. Monte Carlo simulation can be broken up into six steps:

1. Develop a deterministic model
2. Define the probability distribution of possible inputs
3. Draw a set of input values from the random distributions
4. Evaluate the deterministic model using the inputs
5. Repeat steps 3 and 4 for n replications
6. Analyze the results

3.3 The VISOR Web Application

The application allows a user to quickly run Monte Carlo simulations of (r, Q) inventory models with uncertain cost, demand, or lead time parameters. The user supplies the random distributions or constants to be used as parameters, and the application takes care of collecting the results and

reporting the results as summary statistics and charts. The application is entirely web based, and the user runs it from their browser by visiting the URL for the module they want to use.

The application is organized into three separate modules that use a common input interface. Each module runs a slightly different simulation and presents different results. The three modules are the Parameter Comparison module, the Optimal Policy Comparison module, and the Rapid Comparison module. The Parameter Comparison module is highly flexible to allow the user to examine every parameter. The Optimal Policy Comparison module finds a range of optimal policies for a given set of uncertain parameters. The Rapid Comparison module rapidly runs several simulations to determine if an uncertain parameter is contributing to uncertain performance. The following sections describe in detail the function and usage of the input interface and each module.

3.3.1 Input Interface

The user interface for setting up the simulation and defining distributions is similar for each of the three modules of the application. The application uses three windows, the Build Item Window, Optimal Policy Window, and the Normal Distribution Window, in each of the modules. The Build Item Window allows the user to describe an item using constants or random distributions, the Optimal Policy Window finds a cost minimizing policy for a deterministic item, and the Normal Distribution Window fits a normal distribution to user input.

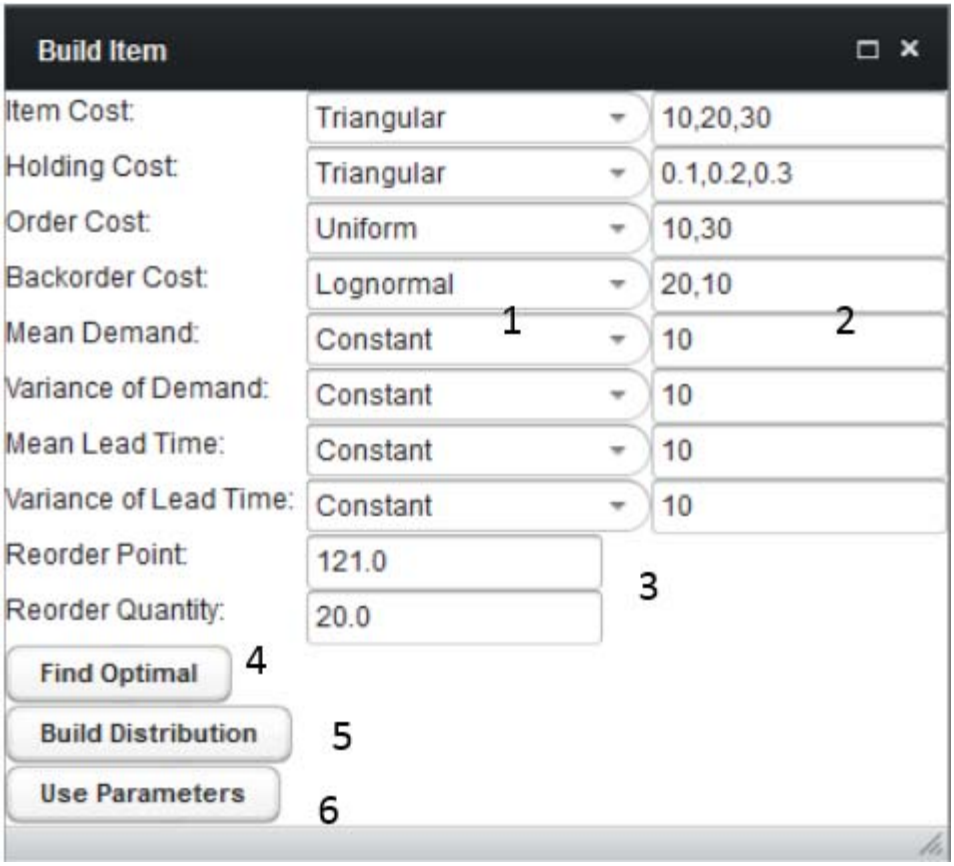


Figure 2 -- Build Item Window

All three modules of the web application use the Build Item Window for defining the parameters for the (r, Q) model. For each of the model parameters, select the type of distribution (or constant) from the drop down box (#1, Figure 4) and enter the appropriate parameters in the neighboring text box. (#2, Figure 4)

Table 4 -- Supported Random Distributions

Distribution	Parameters	Example
Constant	Value	27
Lognormal	Mean, Variance	10,20
Triangular	Minimum, Mode, Maximum	10,25,30
TNormal (Truncated Normal)	Mean, Variance	10, 20
Uniform	Minimum, Maximum	10, 20

Enter the reorder point and reorder quantity in the appropriate text boxes (#3, Figure 4). Clicking the Find Optimal button (#4, Figure 4) will open a new window to find an optimal policy. The Build distribution button (#5, Figure 4) opens a new window to allow the user to build a Normal distribution. The Use Parameters button (#6, Figure 4) loads the parameters for the application to use.

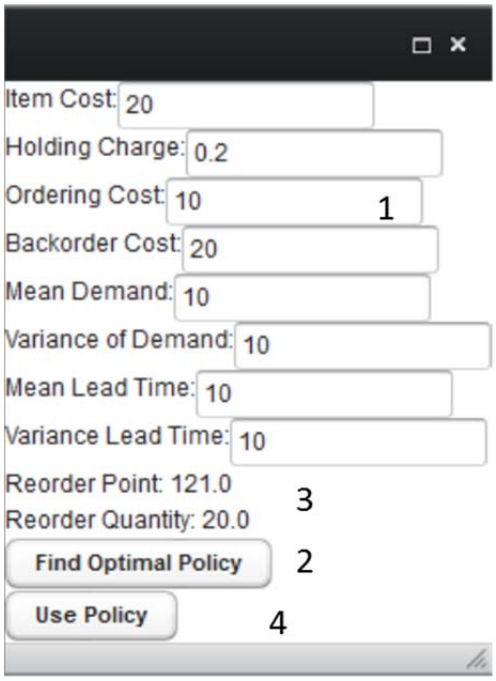


Figure 3 -- Find Optimal Policy Window

The Find Optimal utility will find an optimal reordering policy for a set of deterministic parameters. To use the utility, first enter values for each of the parameters (#1, Figure 5). Next, click the Find Optimal Policy button (#2, Figure 5). The resulting policy will be displayed (#3, Figure 5). Clicking the Use Policy button (#4, Figure 5) will copy the policy to the Build Item window.

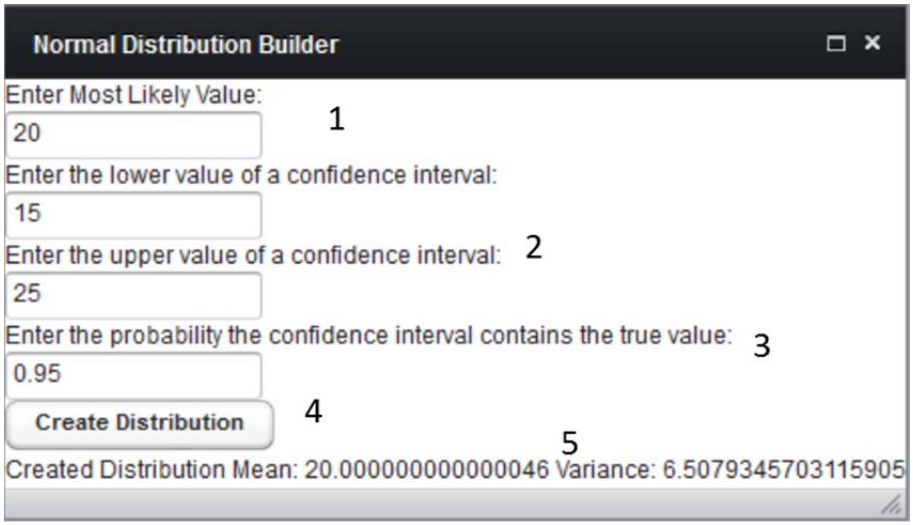


Figure 4 -- Normal Distribution Builder Window

The application provides a utility that will fit a normal distribution. The utility needs a modal value (#1, Figure 5), an interval (#2, Figure 5), and the probability of a random value falling within that interval (#3, Figure 5). Clicking the Create Distribution button (#4, Figure 5) will find a distribution that fits the values. It will display (#5, Figure 5) the mean and variance of a truncated normal distribution that can be used as the parameters for the TNormal distribution. The truncated normal ensures that generated parameter values cannot be negative. These values can be copied into the appropriate text box in the Build Item window.

3.3.2 Parameter Comparison

The Parameter Comparison application is the most flexible part of the application. It allows a user to closely examine one or more items under different levels of uncertainty or ordering policies.

This module requires the user to input parameters, either constants or random distributions, for the cost and lead time demand parameters, as well as a reordering policy. For each replication of the simulation, new random parameters are drawn. When all replications are

run, summary statistics and a histogram of the performance metric the user has chosen are displayed.

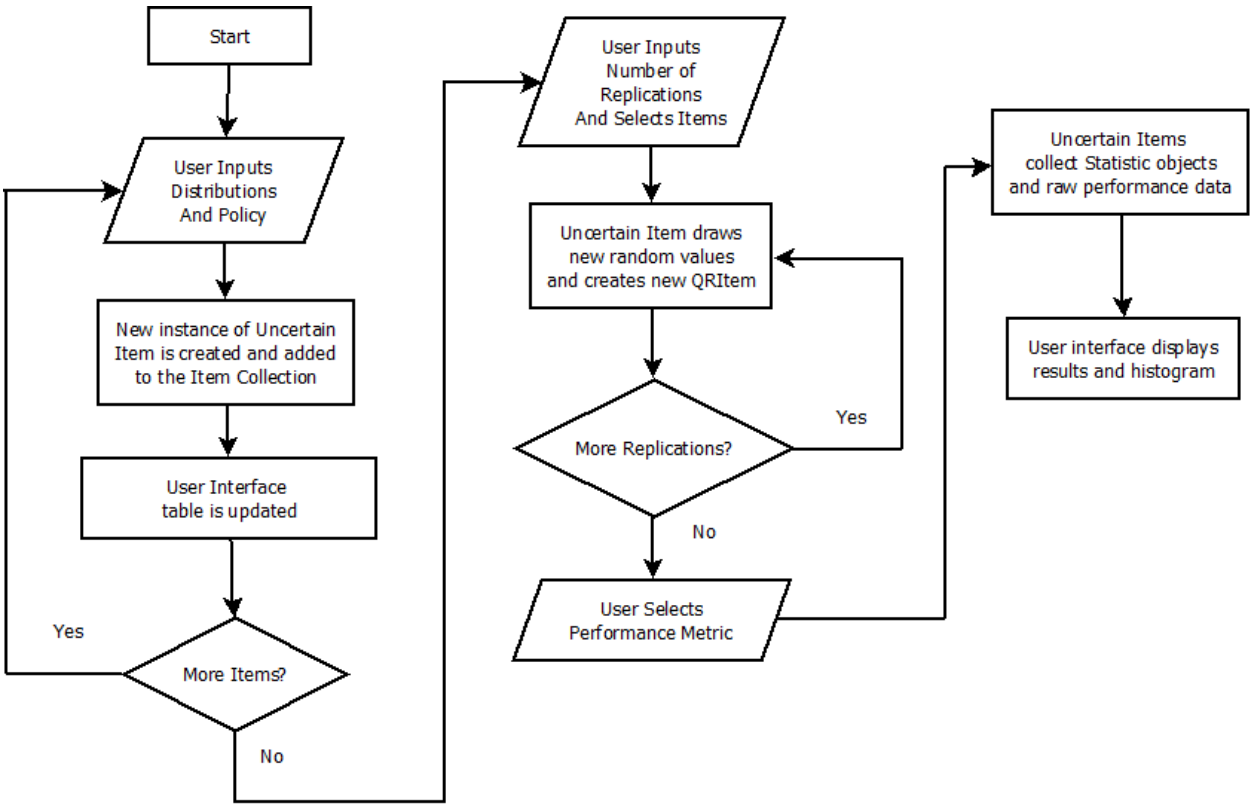


Figure 5 -- Parameter Comparison Module Flow Chart

3.3.2.1 Using the Parameter Comparison Module

The Parameter Comparison module is the most flexible part of the application. In this module, comparisons between different levels of uncertainty, reordering policies, or cost parameters can be made. The module displays summary statistics and a histogram of the results of the simulation.

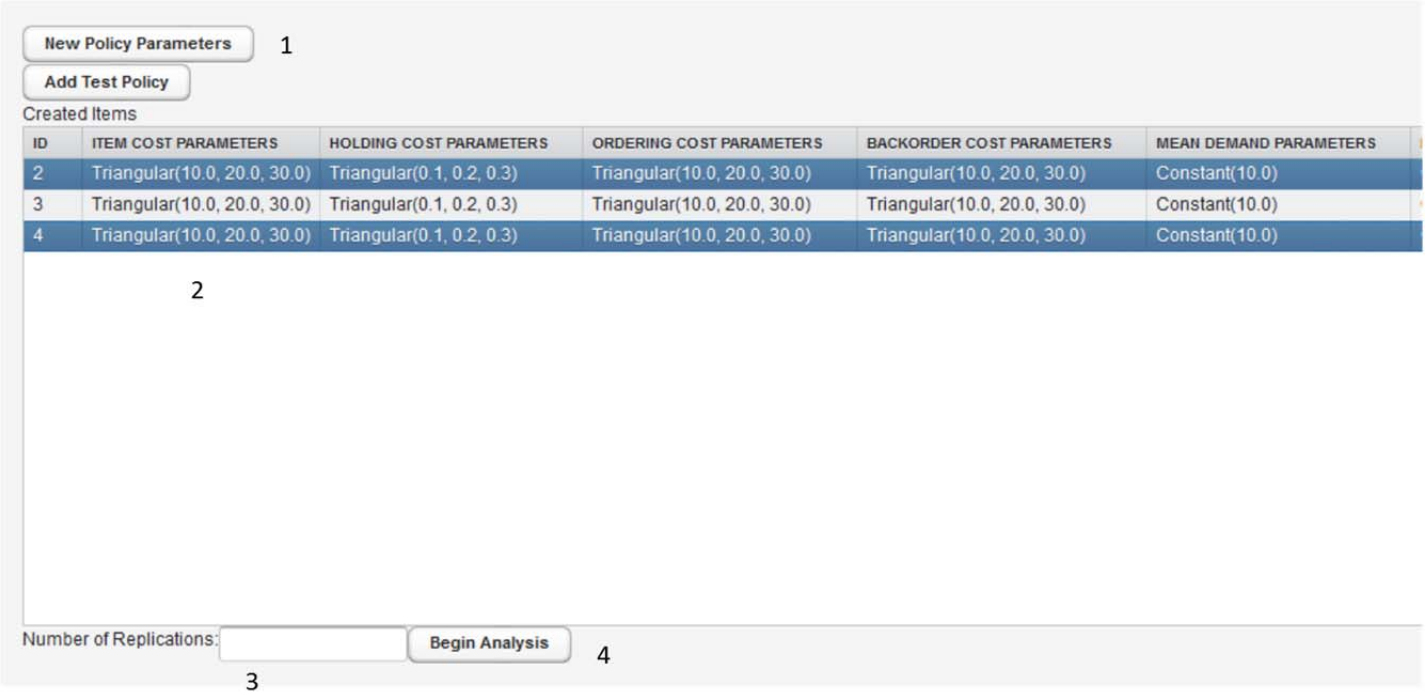


Figure 6 -- Parameter Comparison Home Screen

Begin by pressing the New Policy Parameters button (#1, Figure 8) to add a set of parameters. Multiple sets can be added at once. Once finished, close the Build Item window and select the desired items from the list (#2, Figure 8). Multiple items can be selected by holding down the Control key. Enter a number of replications to run (#3, Figure 8) and press the Begin Analysis button (4) to view the results.

Pressing the Begin Analysis button opens a new window. The parameters for each selected item are displayed on the left (#1, Figure 9). The Number of Bins field (#2, Figure 9) controls how many bins the histogram displays. Select a performance metric to display and press the Refresh button (#3, Figure 9). This will update the display to show the summary statistics (#4,

Figure 9) of the performance metric and a histogram (#5, Figure 9) showing the relative frequency of the performance values.

The histogram can be used as a probability mass function for the performance metric.

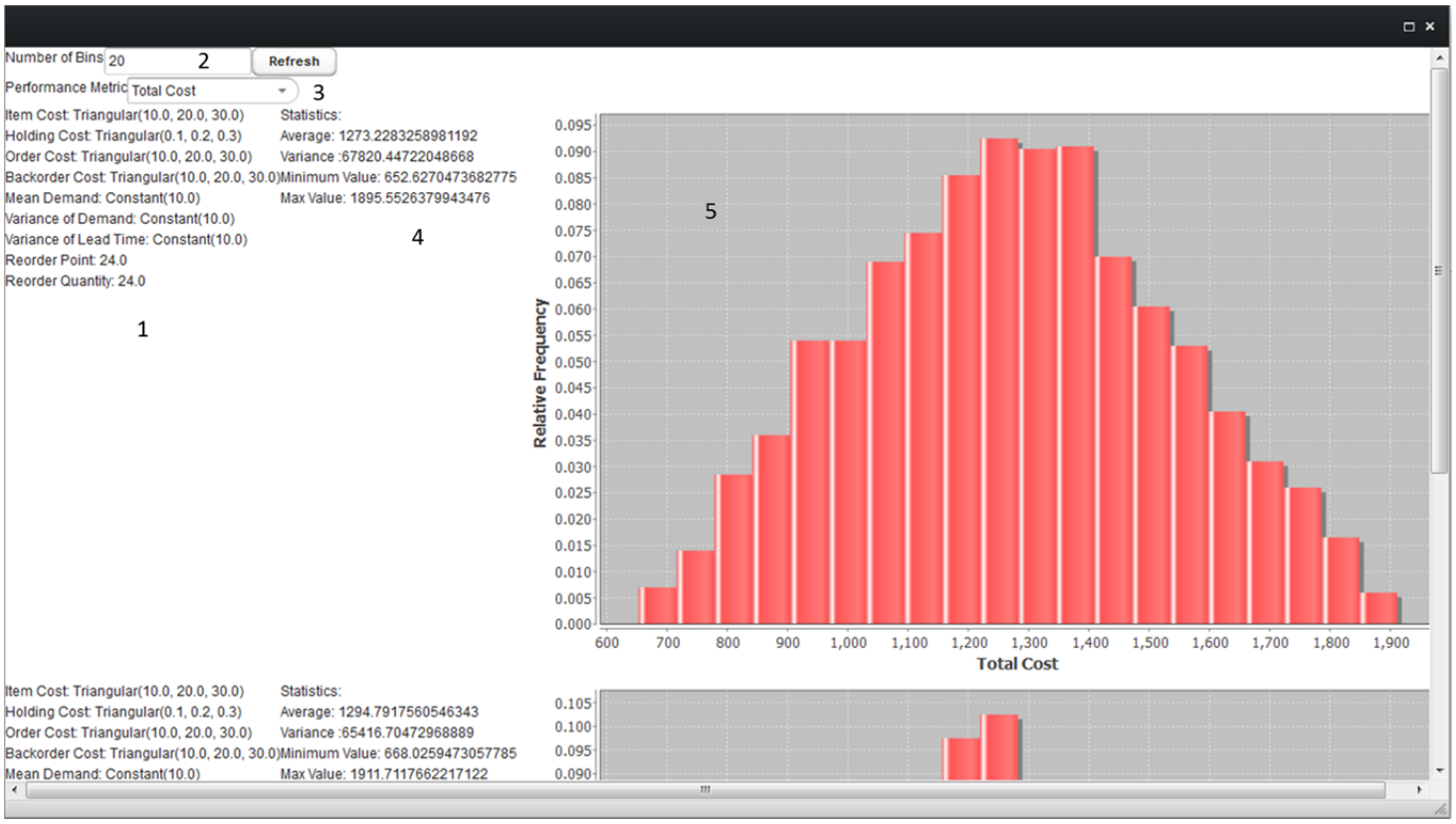


Figure 7 -- Parameter Comparison Results Window

Comparing different sets can inform the user about how different costs, levels of uncertainty, or reordering policies can affect the model performance.

3.3.2.2 Example 1 – Finding a less risky ordering policy

An analyst is examining an item with the following estimated parameters:

Table 5 -- Example 1 Parameters

Item Cost	$5000
Holding Charge	20%/year
Order Cost	$50/order
Backorder Cost	$4500/unit/year
Mean Demand	2.5 units/year
Variance of Demand	0.5 units^2
Mean Lead Time	0.75 years
Variance of Lead Time	0.25 years^2 or 3 days^2

The analyst is concerned that the backorder cost, mean demand, and lead time are significantly higher than the estimates indicates. The analyst would like to discover if a change in reordering policy might eliminate some of the risk of excessive costs. He can use the Parameter Comparison module to do this.

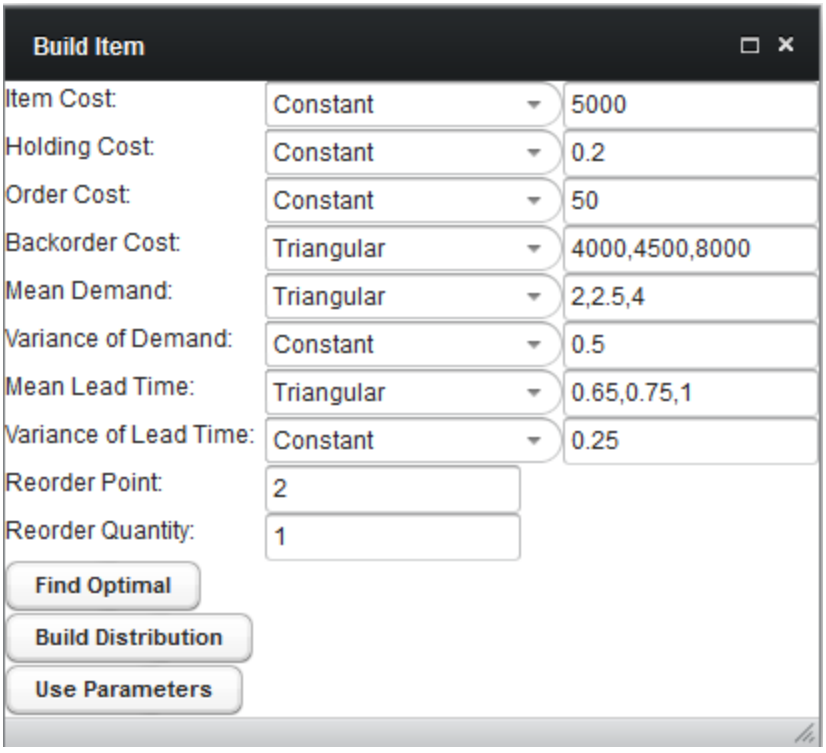


Figure 8 -- Example 1 Distribution Inputs

The analyst begins by loading the parameters in Figure 10 into the application's build item window. The analyst will use different reordering points and quantities, but the first policy

will be the optimal policy, r = 2 and Q = 1. The analyst enters several additional policies with higher and lower reordering points and quantities. The analyst wants to know the probability that total cost exceeds $3,500 and sets the condition bar accordingly. The policies the analyst used and the probability that total cost exceeded $3,500 are summarized in Table 6.

Table 6 -- Example 1 Probability of High Costs

Reordering Point	Reordering Quantity	Pr(Total Cost >= 3500)
2	1	0.295
2	2	0.176
3	1	0.087
3	2	0.056

In this example, the original optimal policy is a very risky policy, and changing to a (3, 2) reordering policy greatly reduces the risk of excessive costs.

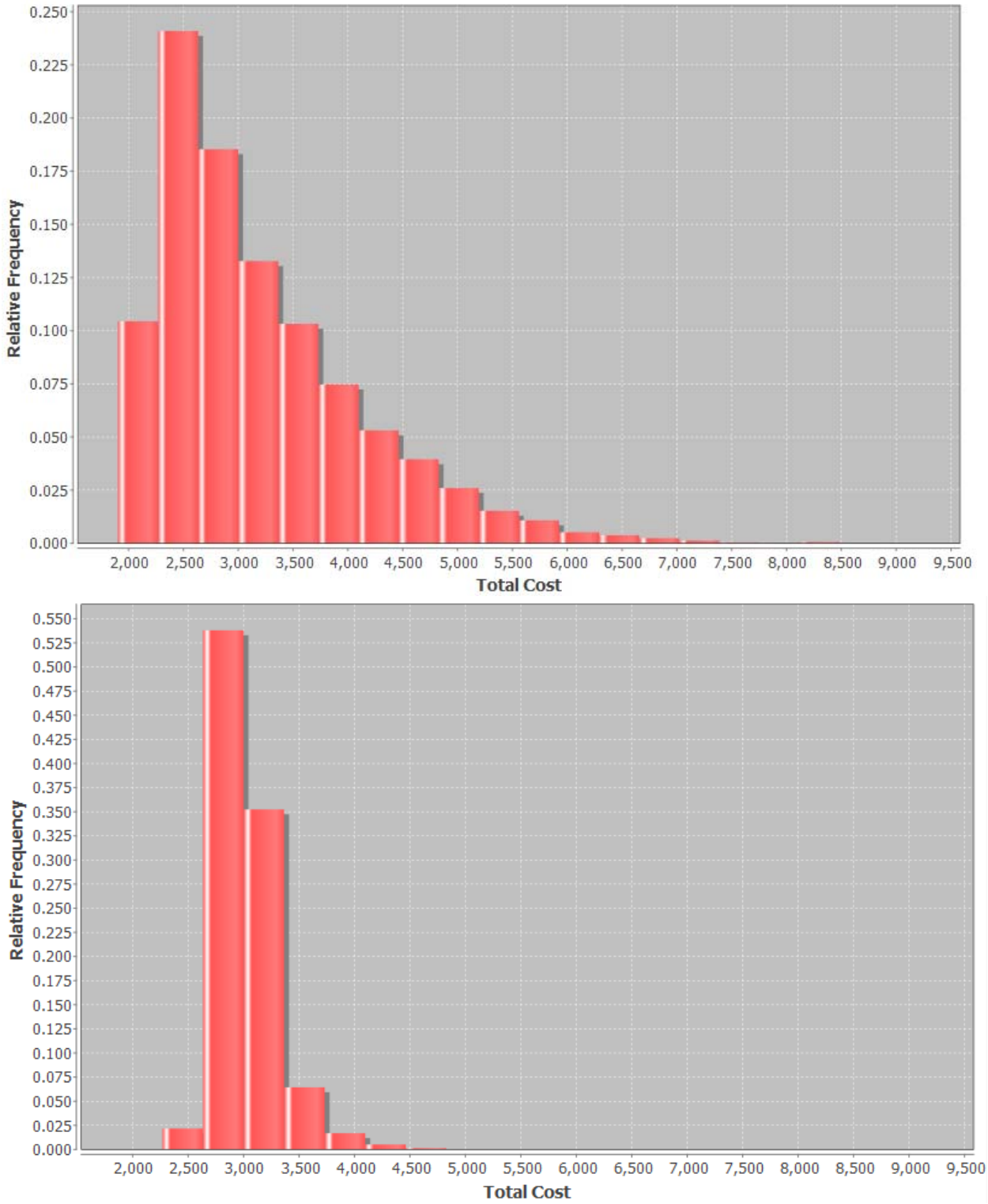


Figure 9 - Histogram for (2,1) policy (top), and (3,2) policy (bottom)

3.3.3 Optimal Policy Comparison Module

The Optimal Policy Comparison application module allows the user to examine the range of optimal policies given a set of parameters as well as which policies are most likely to be optimal. This module requires (r, Q) model parameters, but does not require the user to submit a reordering policy. Like the previous module, new random parameters are drawn for every replication. After new values are drawn, an optimization algorithm is run to determine the reordering policy that minimizes total cost. The application displays the results of the simulation as a tally of all policies and as a scatterplot.

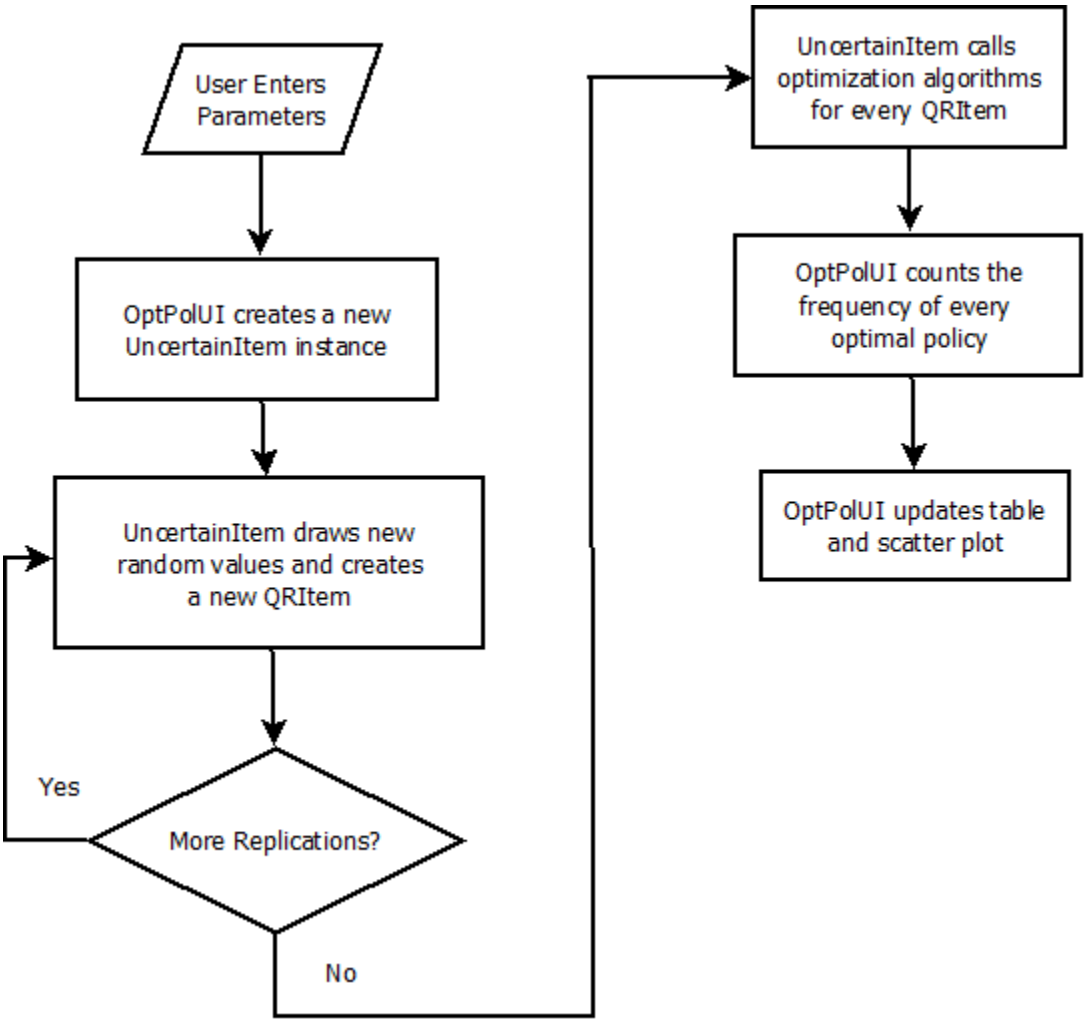


Figure 10 -- Optimal Policy Comparison Flow Chart

3.3.3.1 Using Optimal Policy Comparison

The Optimal Policy Comparison module allows the user to find a range of reordering policies that fit the parameter distributions the user provides.

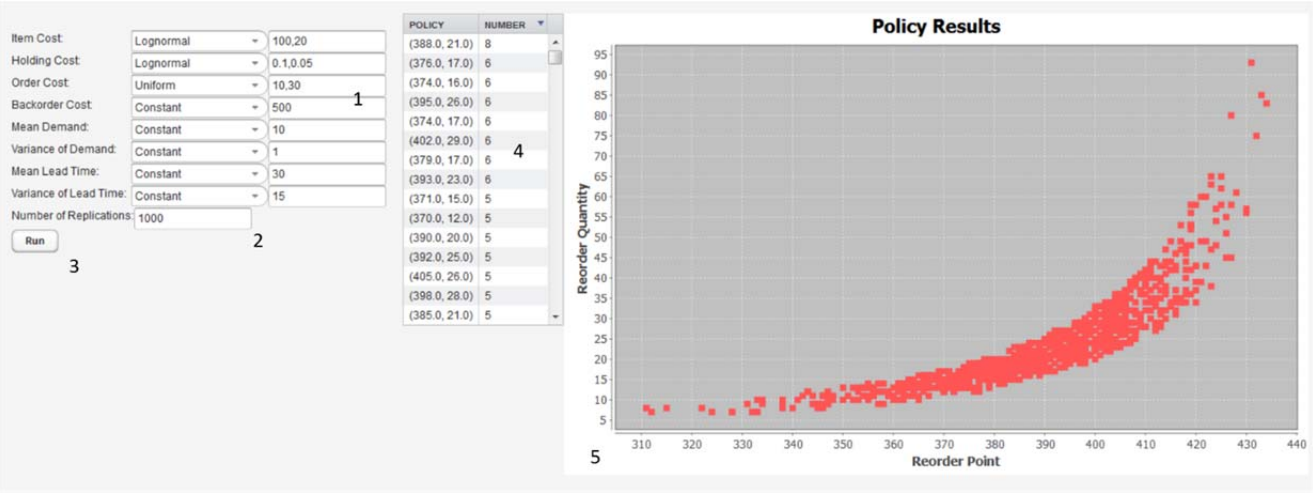


Figure 11 -- Optimal Policy Comparison Screen

Using this module is straightforward. First enter cost, demand, and lead time parameters (#1, Figure 12). No reordering policy is needed. Then enter a number of replications to run (#2, Figure 12). This module calls a potentially time consuming optimization algorithm for each replication, so running an excessively large number can take a long time. First choose a smaller number, fifty for example, to get an idea of how long the simulation will take. Larger lead time demands require more optimization time, so if a simulation will take too long, converting demands from individual units to case quantity can reduce the computation time. For example, replace a demand of 10,000 units with 200 cases of 50. Once all of the parameters are set, click the run button and wait until the simulation finishes (#3, Figure 12).

The results of the simulation are displayed in a table (#4, Figure 12) and a scatter plot (#5, Figure 12). The table displays each of the optimal policies generated by the simulation and ranks

them by frequency. The scatter plot displays each of the policies visually. The most frequent occurrences are those policies that are most likely to be optimal for the set of parameters. These policies will warrant closer examination using the Parameter Comparison Module.

3.3.3.2 Example 2 – Finding alternative optimal policies

The analyst is examining a new item with the following parameters:

Table 7 -- Example 2 Parameters

Item Cost	$1000
Holding Charge	15%/year
Order Cost	$20/order
Backorder Cost	$2850/unit/year
Mean Demand	125 units/year
Variance of Demand	54 units^2
Mean Lead Time	0.0822 years or 30 days
Variance of Lead Time	0.0082 years^2 or 3 days^2

The analyst is very certain about all of these parameters except for the holding charge, which he feels may be much higher. He will represent the holding charge as a lognormal distribution with mean 0.15 and variance 0.1. He would like to find reordering policies that are most likely to be optimal given the uncertainty about the holding charge. In order to do so, he will use the Optimal Policy Examiner.

The analysts loads the Optimal Policy Examiner module and enters the parameters as follows:

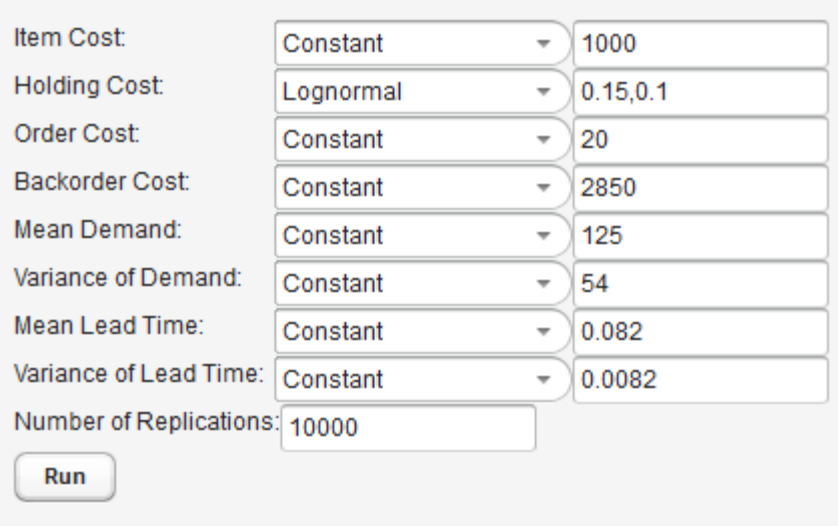


Figure 12 -- Example 2 Distribution Inputs

After running the simulation, the analyst receives the following results:

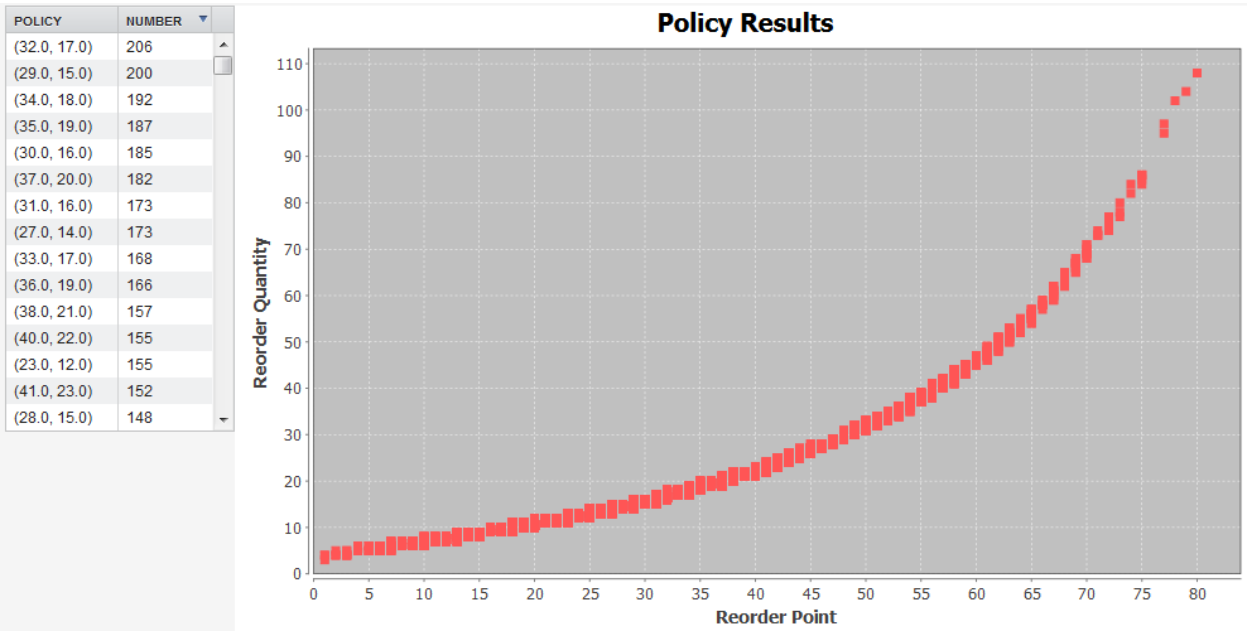


Figure 13 -- Example 2 Results

While there is no one policy that has a very high likelihood of occurring, reordering policies with a reordering point between 27 and 40 are the most common. Sorting the table by policy will show which reordering quantities are most common within the 27-40 range. The analyst can pick several policies from the list for further examination in the Parameter Comparison module.

3.3.4 Rapid Comparison

The Rapid Comparison application allows the user to determine which uncertain parameter is driving uncertainty in performance. It requires the user to input parameters and a reordering policy. It also requires the user to select a parameter they wish to examine. Several simulation runs are made. Each run varies the variance of the parameter the user selected. Each simulation replication is the same as in the Parameter Comparison module. The user is presented with the variance of the performance metric by simulation run, as well as the probability that the performance metric falls within a user defined range.

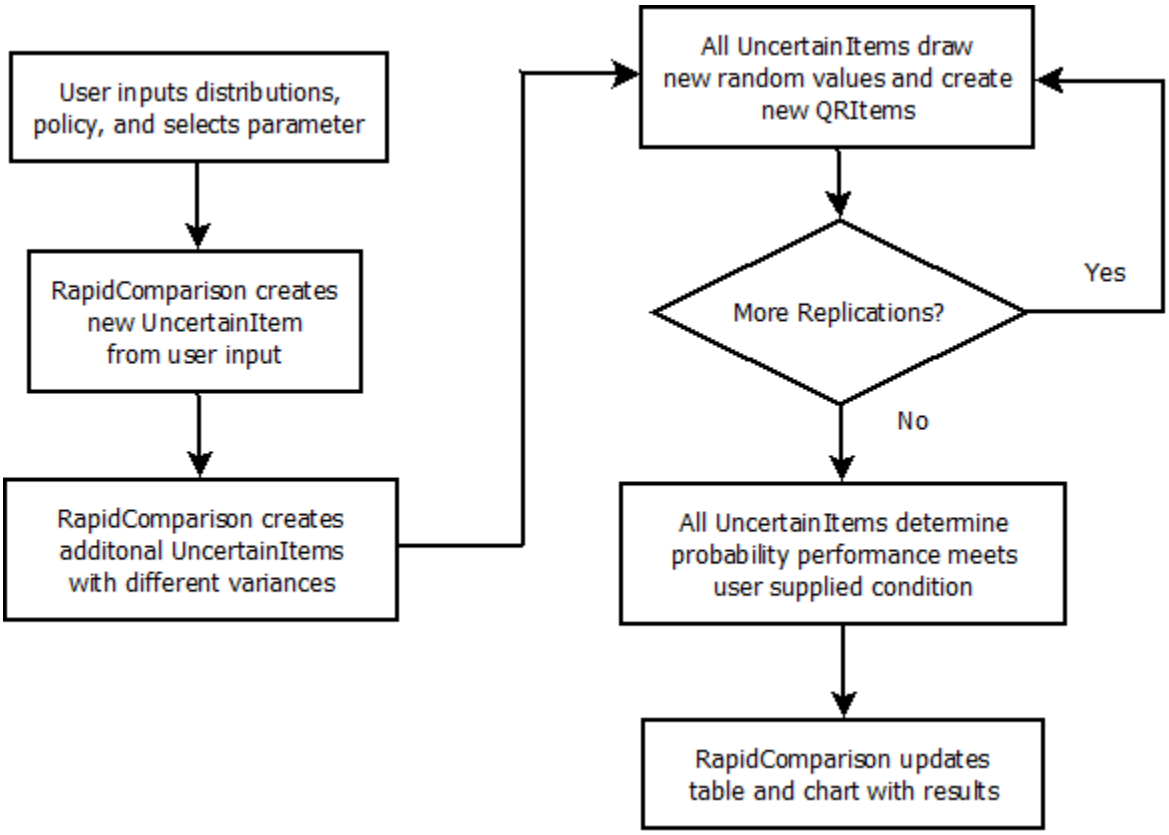


Figure 14 -- Rapid Comparison Flow Chart

3.3.4.1 Using the Rapid Comparison Module

The Rapid Comparison module allows a user to quickly compare the performance effects of different levels of parameter uncertainty. To model higher levels of uncertainty, the application

increases the variance of a distribution. Given a set of model parameters and a performance range, the module will calculate the probability that a performance metric will fall within the range as the variance of one of the input parameters changes.

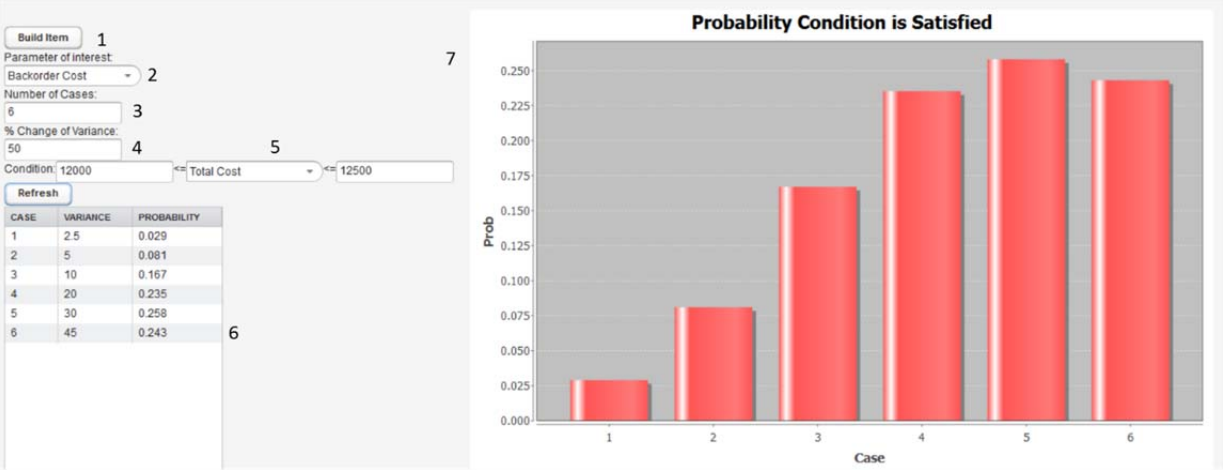


Figure 15 -- Rapid Comparison Screen

Begin by pressing the Build Item button (#1, Figure 16) and entering the parameters for an item and a reordering policy. The application can only change the variance of truncated normal distributions, but will accept any other distributions at this point. Once the parameters and reordering policy are entered, click the Use Parameters button and close the window.

Next select a parameter from the Parameter of Interest box (#2, Figure 16). Only select a parameter that is described with a truncated normal distribution. This will be the parameter whose variance will be changed by the application. Next, enter the number of different cases (#3, Figure 16). This is the number of different levels of uncertainty the application will create. The default is six. The % Change of Variance box (#4, Figure 16) controls the size of the step between different levels of uncertainty. The default is 50%. The last step is to enter a condition (#5, Figure 16). Select a performance metric and enter an upper and lower bound, for example 1400 <= Total Cost <= 1600. In addition to numerical values, the text fields will accept “Infinity”

and "-Infinity." Using the Parameter Comparison module is helpful in finding an appropriate performance range. Finally, pressing the Refresh button will update the table and graph.

The table (#6, Figure 16) will display each of the cases, the variance of the parameter of interest for a particular case, and the probability that the performance falls within the user supplied range. The chart (#7, Figure 16) displays the same information visually. If the uncertainty of a particular parameter is contributing to the uncertainty of the performance metric, the probability of the performance metric falls within the range should be changing. If the probability is the same across all cases, it is likely that the parameter is not significantly contributing to performance uncertainty.

3.3.4.2 Example 3 -- Determining contributors to performance uncertainty

The analyst is examining a third item with the following parameters.

Table 8 - Example 3 Item

Item Cost	$2.80
Holding Charge	22%/year
Order Cost	$5.50/order
Backorder Cost	$30.20/unit/year
Mean Demand	312 units/year
Variance of Demand	78 units^2
Mean Lead Time	0.0384 years or 14 days
Variance of Lead Time	0.0082 years^2 or 3 days^2

The costs for stocking this item have been higher than expected, and the analyst would like to know which parameter is the most likely cause for higher costs. To find out, the analyst will use both the Parameter Comparison and Rapid Comparison module.

The analyst first uses the Parameter Comparison module to gain a basic understanding of the total cost. The analyst is 95% confident that the actual value for all of the parameters lies

within 10% of the estimate. He uses the Normal Distribution Build window to fit a truncated normal distribution to each of the parameters and fills in the parameters.

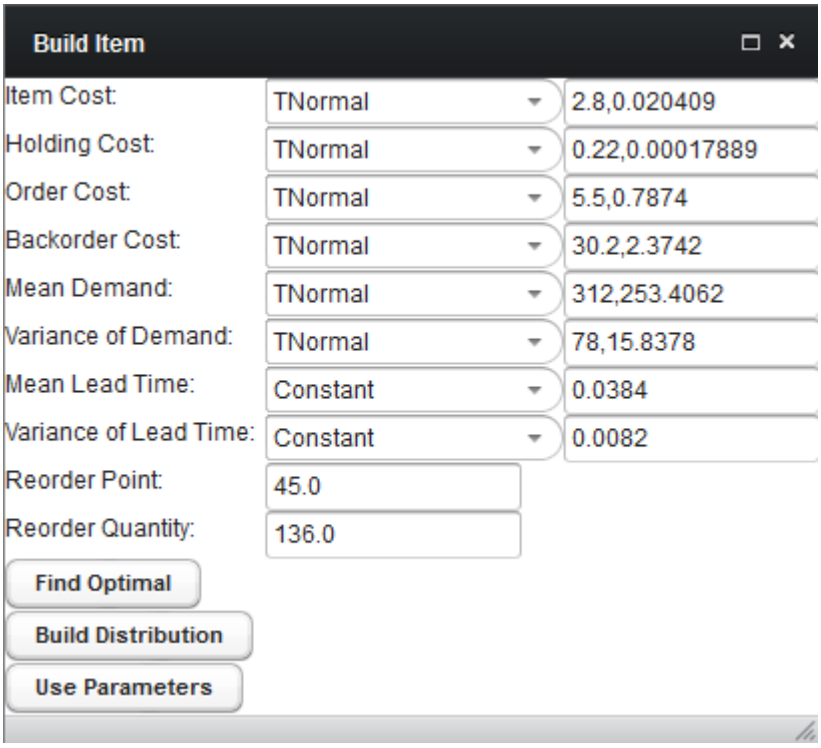


Figure 16 - Example 3 Parameters

The analyst runs the simulation and obtains the result in Figure 17:

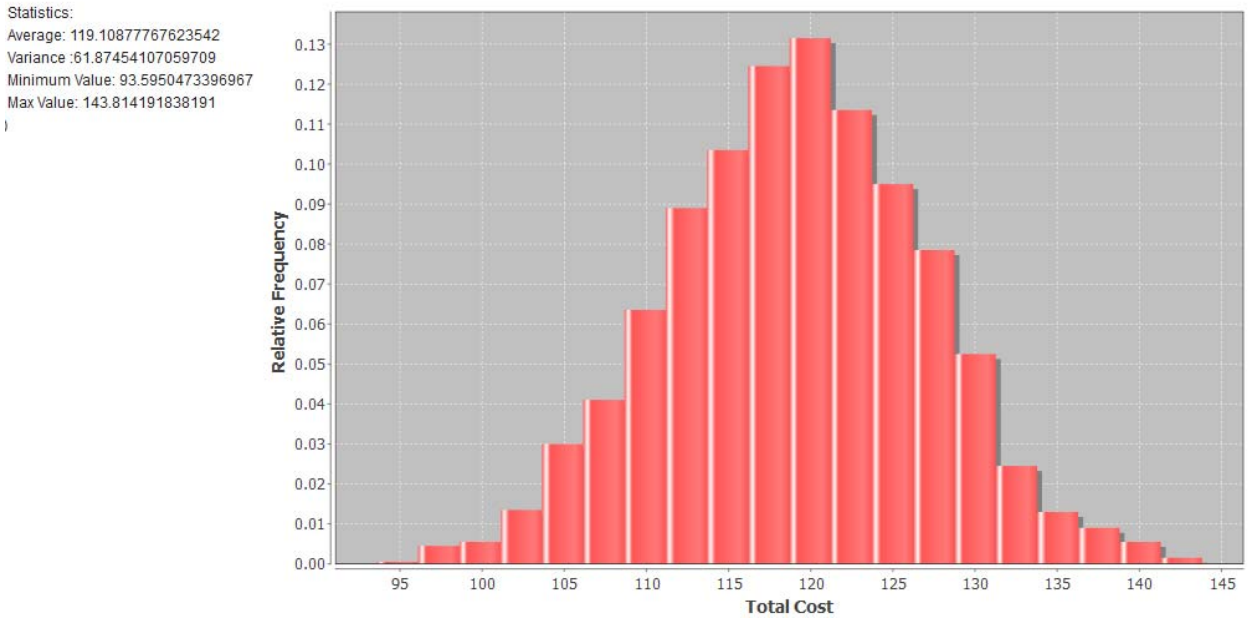


Figure 17 -- Example 3 Results

For now, it is enough to note that the average is about $119. The analyst wants to know which uncertain parameter is most responsible for higher than expected cost, so the analyst will use a value slightly higher than the average in the Rapid Comparison module.

The analyst loads the Rapid Comparison module and inputs the same distributions. For the condition, he enters 125 as the lower bound, selects Total Cost for the performance metric, and enters “Infinity” for the upper bound. The analyst selects Item Cost as the parameter of interest and obtains the following chart:

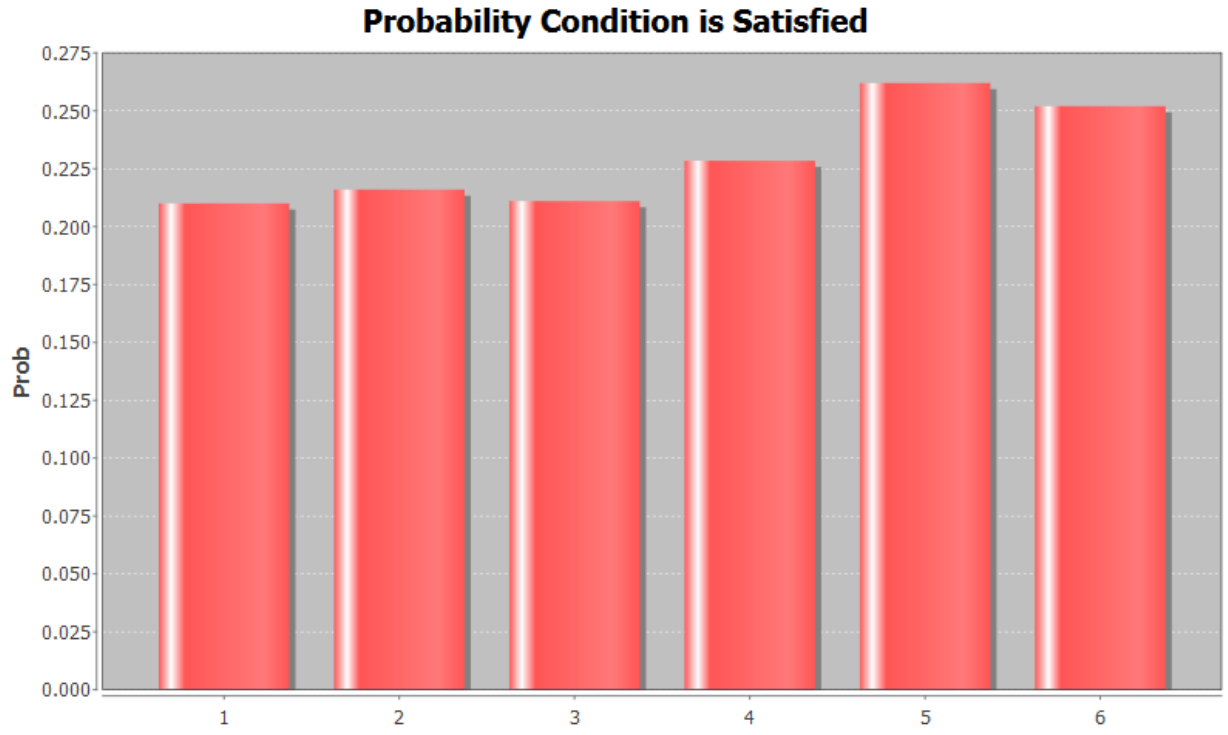


Figure 18 -- Item Cost Uncertainty Results

There is a slight increase in the probability of Total Cost being higher than $125 as the variance of the item cost distribution increases. From this, the analyst can conclude that any uncertainty about the item cost is a minor contributor to excessive total costs.

Changing the parameter of interest to Order Costs produces the following chart:

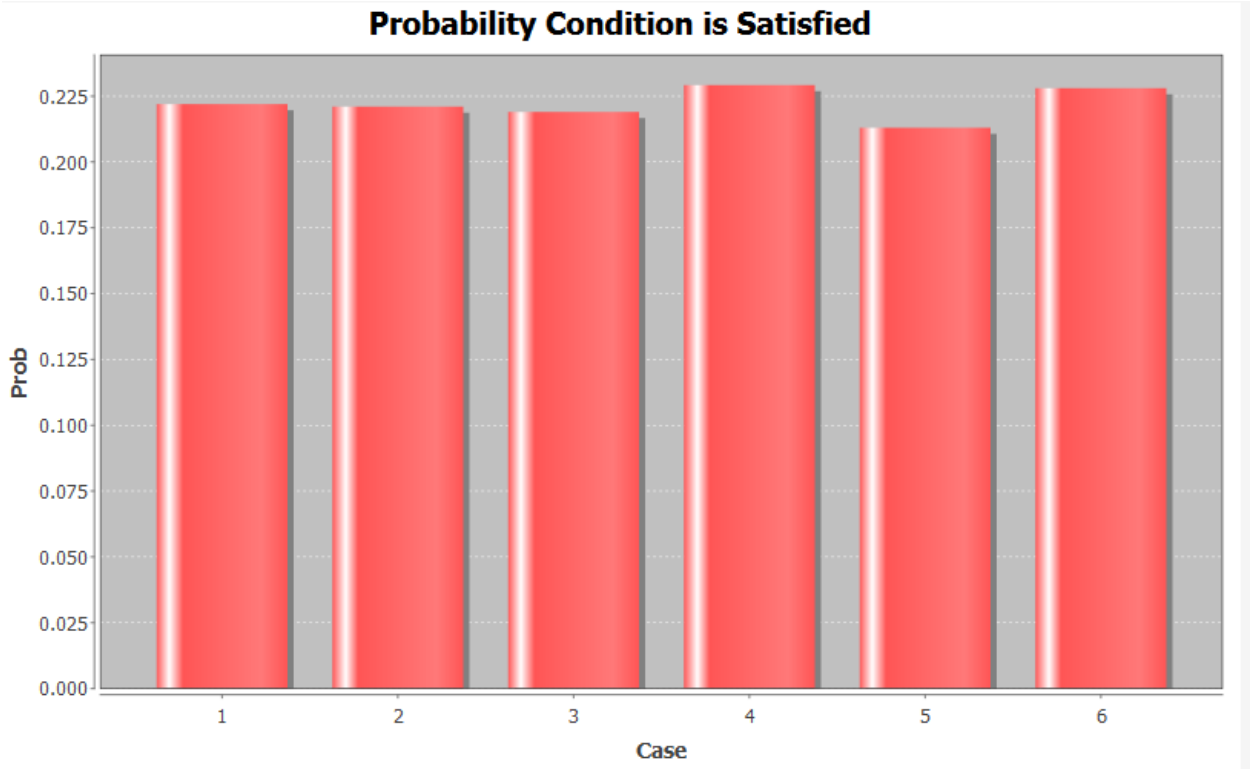


Figure 19 -- Order Cost Uncertainty Results

Here, there is no increase in the probability as variance increases. The analyst concludes that uncertainty about the order cost is contributing little, if any, to the excessive total costs.

The chart for mean demand shows a steeper increase in probability than item cost.

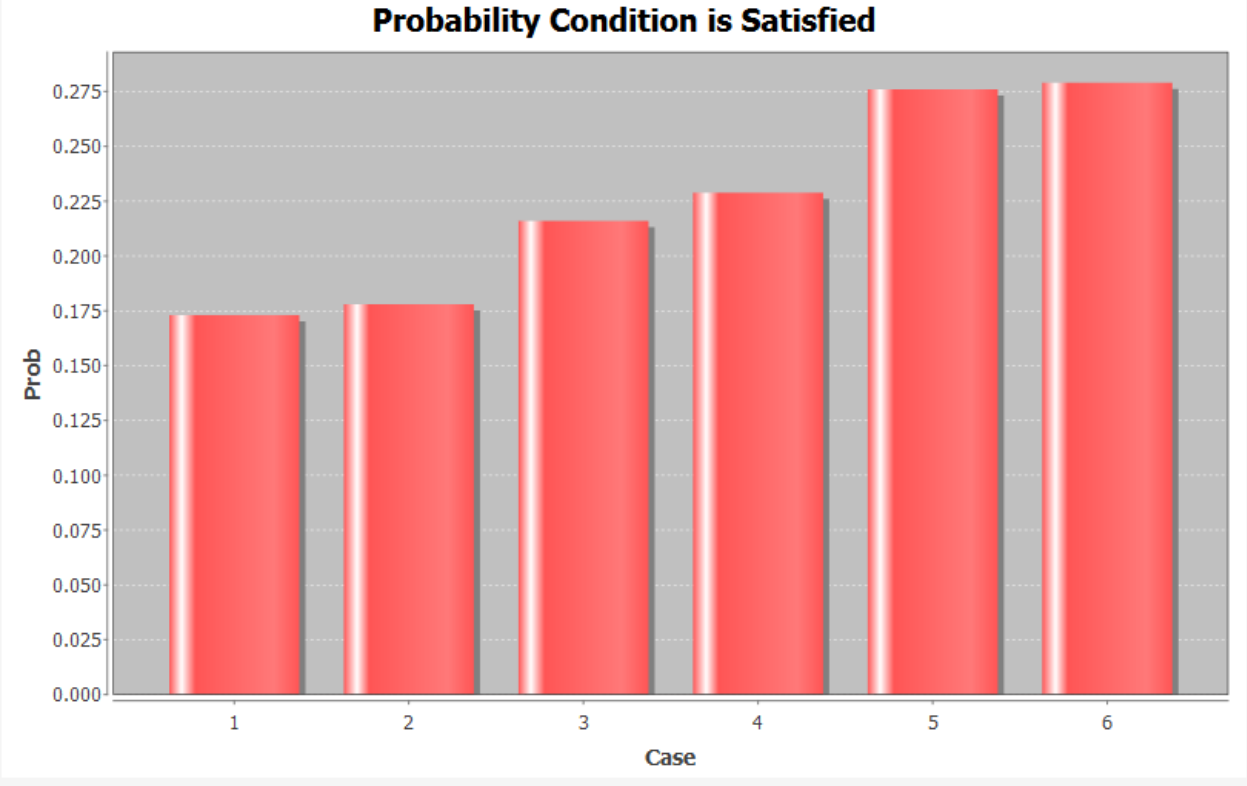


Figure 20 -- Mean Demand Uncertainty Results

From the chart, the analyst concludes that uncertainty about mean demand is contributing most to excessive cost.

3.4 Web Application Development

The following section details the development process, including the libraries and frameworks used in development, the overall application structure, the development and testing process, as well as future work that could improve the application.

3.4.1 JSL and Inventory Web Service

The Java Simulation Library and the Inventory Web service developed for previous research projects were used extensively to develop the simulation and web application. The JSL

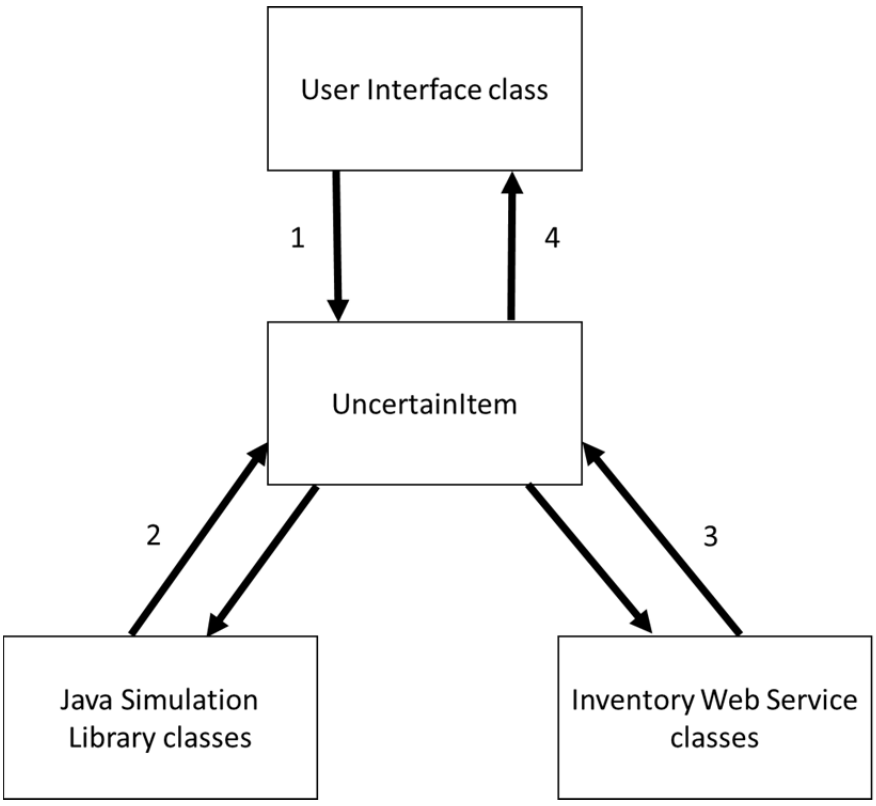
implements random number generation algorithms and statistics collection. The Inventory WS has code that handles computation of performance metrics and optimizing the inventory policy.

3.4.2 Vaadin Framework

The tool was developed using the Vaadin framework. Vaadin is a Java based framework for developing web applications. It allows the development of a web based application using Java as the programming language. Development is similar to developing a typical Java GUI application. Vaadin comes with a set of common user interface components than can be extended for custom uses.

3.4.3 Application Structure

The application is broken into three modules, Parameter Comparison, Optimal Policy Comparison, and Rapid Comparison. All three modules follow the same structure. A user interface class collects input from the user to create an UncertainItem instance. This class is the center of the application. For each replication, the UncertainItem will draw new values from JSL distributions and use them to create a new QRItem instance from the Inventory Web Service. The UncertainItem can draw raw performance data from the QRItem or summary statistics by using the JSL Statistic class. The UncertainItem sends the information back to the user interface to be presented to the user.



1	Interface creates an UncertainItem using user input
2	UncertainItem draws random numbers from JSL and uses Statistic class to generate summary statistics
3	UncertainItem uses random draws to create QRItems, can use optimization algorithms, and gather performance metrics
4	UncertainItem sends results to UI to be displayed to the user

Figure 21 -- Main Application Component Interactions

Each module of the software runs simulations slightly differently, requires different input and produces different output. The following sections summarize how each software module functions.

3.4.4 Development Process

The first step in the development process was to create a set of use cases that would guide the development of the application. The use cases were:

1. A user enters a set of cost parameters and a reordering policy and runs a Monte Carlo simulation. The results are displayed as summary statistics and as a histogram.
2. A user enters an interval and the application determines the probability that the performance metric falls within the interval.
3. A user performs 1 and 2 with multiple sets of parameters and/or reordering policies at once.
4. User enters a set of cost parameters and reordering policy to determine which parameters are causing the most variability in performance.
5. User enters a set of cost parameters. The application displays the range of optimal reordering policies.

Initially, it was planned to have only two modules, Parameter Comparison and Optimal Policy Comparison, to perform all five cases. However, after Parameter Comparison was completed, it was determined that an additional module was needed for case 4. It was time consuming to use Parameter Comparison for this task, so the Rapid Comparison module was developed for this case.

The UncertainItem class had been previously developed for the examples in section 3.2.1, and it was decided to reuse this class and to modify it as needed.

Development for each specific module followed a similar cycle. The first step was to determine what output the application would have and what input the user needed to provide for the application to function. The next step was to sketch a rough layout for the GUI and then to write the code for the user interface. With the user interface implemented, methods to parse user input were implemented, and the algorithms needed to make the application function correctly were written. The last step in the cycle was to write the code to display the output correctly.

The first module developed was Parameter Comparison, and the first version was implemented without using any custom GUI classes. Each element of the GUI had its own field in the UI class. This was functional, but with over 70 GUI elements in the application, the code was time consuming to implement, had poor readability, and was difficult to manage the layout and make changes. This was unacceptable, and the program was redesigned using object-oriented principles, with an eye towards making custom classes for common groups of elements, a label next to a text field, for example, and custom classes for elements that work together. For instance, all of the elements required for the user to enter model parameters are in the ParameterDistributionBox class. This greatly simplified the main user interface classes and made reusing code much simpler.

3.4.5 Common Composite GUI Components

There are two classes, LabelComboBoxTextField and LabeledTextField, which are nothing more than composites of basic Vaadin GUI components.

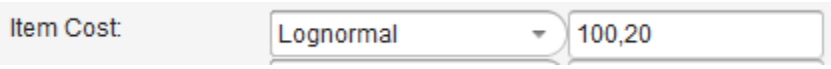


Figure 22 -- Example LabelComboBoxTextField

Figure 23 -- Example LabeledTextField

Like their names suggest, the classes consist of a Label, Combo Box, and a Text Field; and a Label and a Text Field, respectively. These are convenience classes to reduce the number of fields in the main applications. They have methods that return references to their fields.

The ConditionBar class contains GUI components and methods to allow the user to input a range and performance metric. The only fields are GUI components. The methods will return the lower and upper bounds as well as the selected performance metric.

Figure 24 -- Condition Bar

Table 9 -- Condition Bar methods

Method Signature	**Return Type**	**Notes**
public double getLowerBound()	double	Parses the lower bound input to a double. Valid input is any number, “infinity” or “-infinity”
public double getUpperBound()	double	Parses the upper bound input
public String getPerformanceMetric()	String	Returns the selected string from the combo box
private void addPerformanceMetrics()	Void	Adds the options for different performance metrics to the combo box

The PolicySettingsBox has GUI elements for the user to input a reordering policy when the application requires it. Its fields include the GUI elements as well as an instance of the OptimalPolicyWindow class.

Figure 25 -- Policy Settings Box

Table 10 -- Policy Settings Box methods

Method Signature	**Return Type**	**Notes**
public void setRQ(double r, double q)	Void	Sets the value of the r & q text fields
public double getR()	double	Returns the reorder point
public double getQ()	double	Returns the reorder quantity

The OptimalPolicyWindow class takes deterministic policy parameters and will find an optimal ordering policy for the user. Its only fields are GUI components. It has two methods.

One method calls the optimization algorithm, and the other copies the policy to the parent PolicySettingsBox.

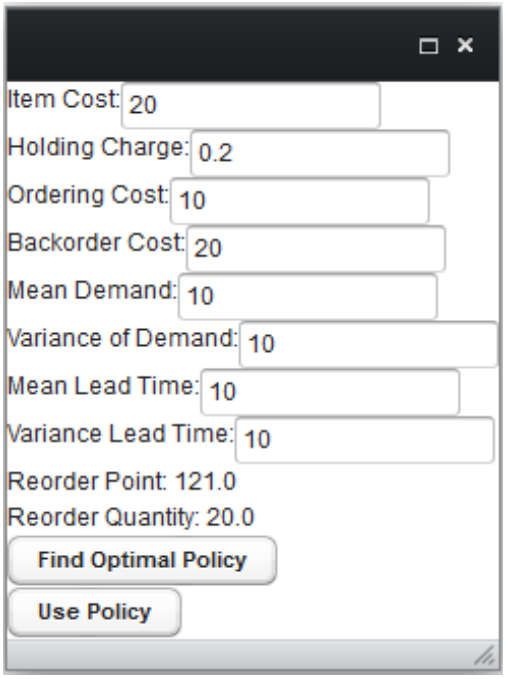


Figure 26—Optimal Policy Window

Table 11 -- Optimal Policy Window methods

Method Signature	**Return Type**	**Notes**
private void findOptimalAndUpdate()	void	Finds the optimal reordering policy and displays it for the user
public void usePolicy()	void	Copies the current reordering policy to the PolicySettingsBox

All of the components for defining random distributions are contained within the ParameterDistributionBox class. The only fields contained within the class are GUI components. It has methods to get the contents of the combo boxes and text fields.

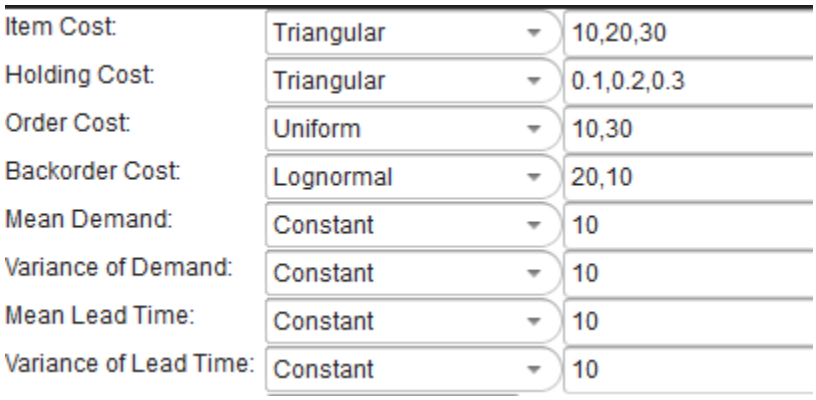


Figure 27 -- Parameter Distribution Box

Table 12 -- Parameter Distribution Box methods

Method Signature	**Return Type**	**Notes**
public String getICType()	String	Returns the selected distribution type for the item cost
public String getICParams()	String	Returns the contents of the item cost parameters text field
public String getHCType()	String	Returns the selected distribution type for the holding cost
public String getHCParams()	String	Returns the contents of the holding cost parameters text field
public String getOCType()	String	Returns the selected distribution type for the order cost
public String getOCParams()	String	Returns the contents of the order cost parameters text field
public String getBCType()	String	Returns the selected distribution type for the backorder cost
public String getBCParams()	String	Returns the contents of the backorder cost parameters text field
public String getMDType()	String	Returns the selected distribution type for the mean demand
public String getMDParams()	String	Returns the contents of the mean demand parameters text field
public String getVDType()	String	Returns the selected distribution type for the variance of demand
public String getVDParams()	String	Returns the contents of the variance of demand parameters text field
public String getMLTType()	String	Returns the selected distribution type for the mean lead time
public String getMLTParam()	String	Returns the contents of the mean lead time parameters text field
public String getVLTTYpe()	String	Returns the selected distribution type for the variance of lead time
public String getVLTParams()	String	Returns the contents of the variance of lead time parameters text field

The NormalBuildWindow allows the user to find a truncated normal distribution that fits parameters provided by the user. The only fields in the class are GUI components. Its one method calls the algorithm to find a fitting distribution.

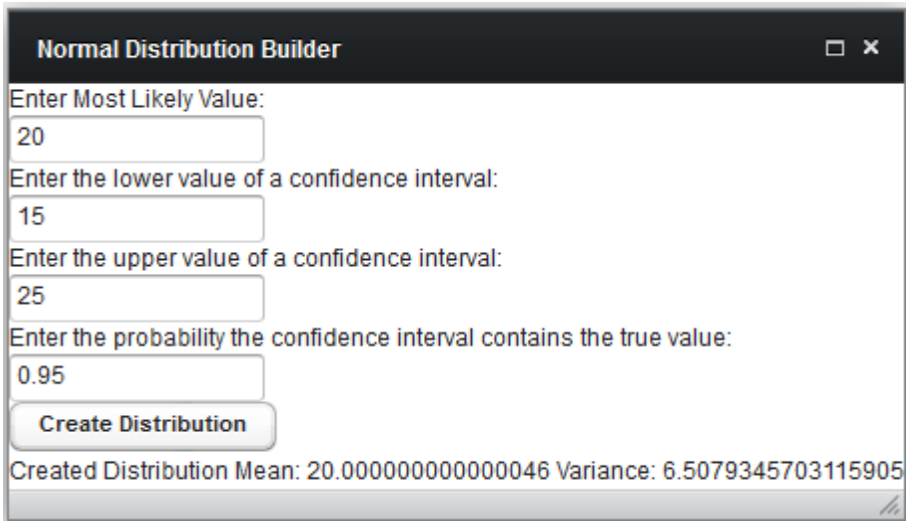


Figure 28 -- Normal Build Window

The BuildItemWindow contains all of the elements necessary for the user to create a new instance of class UncertainItem. The fields are a NormalBuildWindow, a PolicySettingsBox, a ParameterDistributionBox, and the parent user interface. The class has one method that creates a new instance of UncertainItem and sends it to the parent.

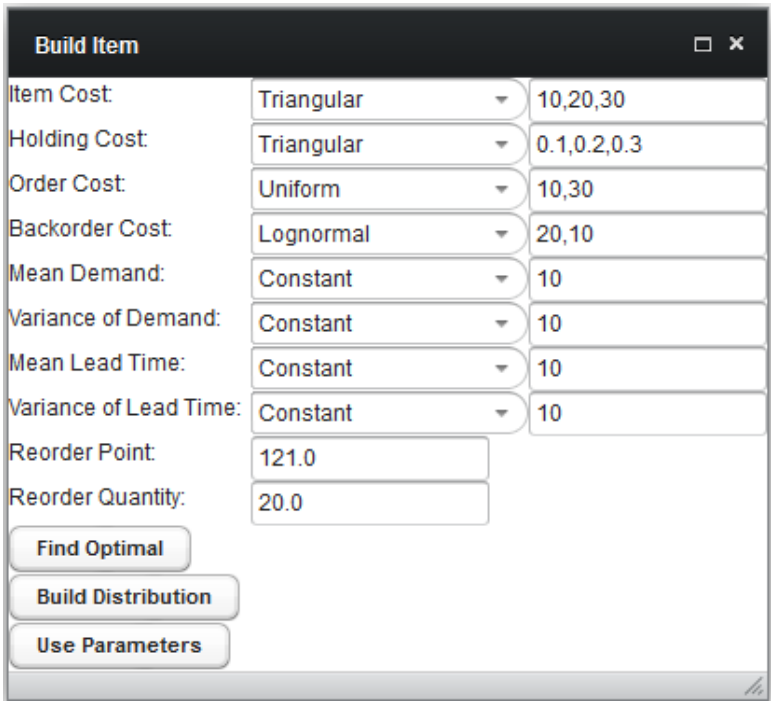


Figure 29 -- Build Item Window

3.4.6 User Interface Classes

All of the main user interface classes implement the ItemContainingUI interface. Implementing this interface allows the BuildItemWindow class to communicate with the main user interface. There is only one method in the interface. The setupItem should receive an UncertainItem from the BuildItemWindow and appropriately process it.

3.4.6.1 Rapid Comparison Module

The main class in the Rapid Comparison module is the RapidComparisonUI class. It contains all of the GUI elements and methods necessary to run the module. In addition to the GUI components, the class has fields to store the user created UncertainItem and the items the program creates.

Table 13 – Rapid Comparison UI fields

Field Type and Name	**Notes**
UncertainItem uItem	The user created item
UncertainItem[] allItems	An array containing the items created by the program
TruncatedNormal changedDistribution	The distribution that the user has chosen to vary the variance.

Table 14 -- Rapid Comparison UI methods

Method Signature	**Return Type**	**Notes**
protected void init(VaadinRequest request)	void	Runs when the UI is first opened. Creates all of the UI components
public UncertainItem getUItem()	UncertainItem	Returns the user created item
public void setupItem(UncertainItem uItem)	void	Recieves a newly created item from the BuildWindow and stores it for future use.
private JFreeChart createBlankBarGraph()	JFreeChart	Creates a blank graph to be displayed when the UI is first loaded
private void updateTableAndGraphs()	void	Runs when the user presses the update button. Clears the table, calls methods to run the simulation, fill the table, and display a new chart.
private void createAdditionalItems()	void	The first part of the simulation. Creates new uncertain items and stores them in the allItems array
private void runReps()	void	Runs 1000 replications of the Monte Carlo simulation for every item in the allItems array
private void populateTable()	void	Displays the results of the simulation in the main UI table
private UncertainItem copyItemAndChangeVariance(double varChangeFactor)	UncertainItem	Returns a new UncertainItem. The variance of the user selected distribution has been multiplied by varChangeFactor
private JFreeChart createBarGraph()	JFreeChart	Returns a new bar graph of the results
private TruncatedNormal getSelectedDistribution(UncertainItem item, String selection)	TruncatedNormal	Returns the distribution specified by selection from item.

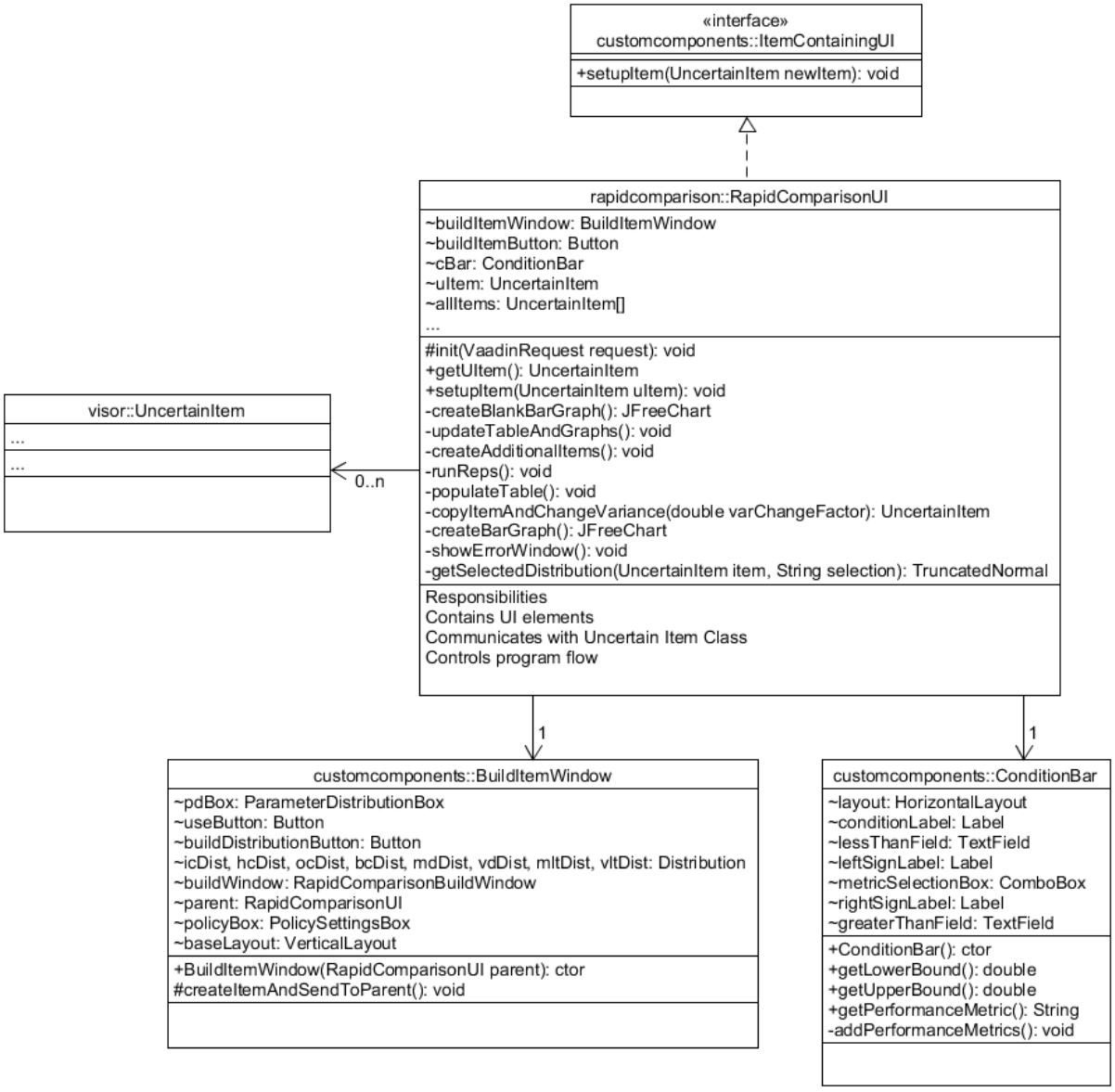


Figure 30 -- UML Diagram for Rapid Comparison

3.4.6.2 Optimal Policy Examiner Module

The GUI components and methods for the Optimal Policy Examiner module are contained within the OptPolUI class. In addition to the GUI components, there are fields to store the user

created UncertainItem, an ArrayList to store the optimized policies, and a separate ArrayList that acts as a tally for the number of times a particular reordering policy occurs.

Table 15 -- OptPolUI methods

Method Signature	**Return Type**	**Notes**
protected void init(VaadinRequest request)	void	Runs when the UI is first opened. Creates all of the UI components
public void runReplications()	void	First creates a new UncertainItem from user input. Then runs the simulation and optimizes all of the policies.
public void tallyPolicies()	void	Counts the number of occurrences of specific policies. Can bin them into user specified bins
public void writeTallyToTable()	void	Writes the reordering policies and counts to the table
public void drawScatterPlot()	void	Displays the reordering polices as a scatter plot
public JFreeChart makeBlankChart()	JFreeChart	Creates a blank chart to display when the UI is first displayed

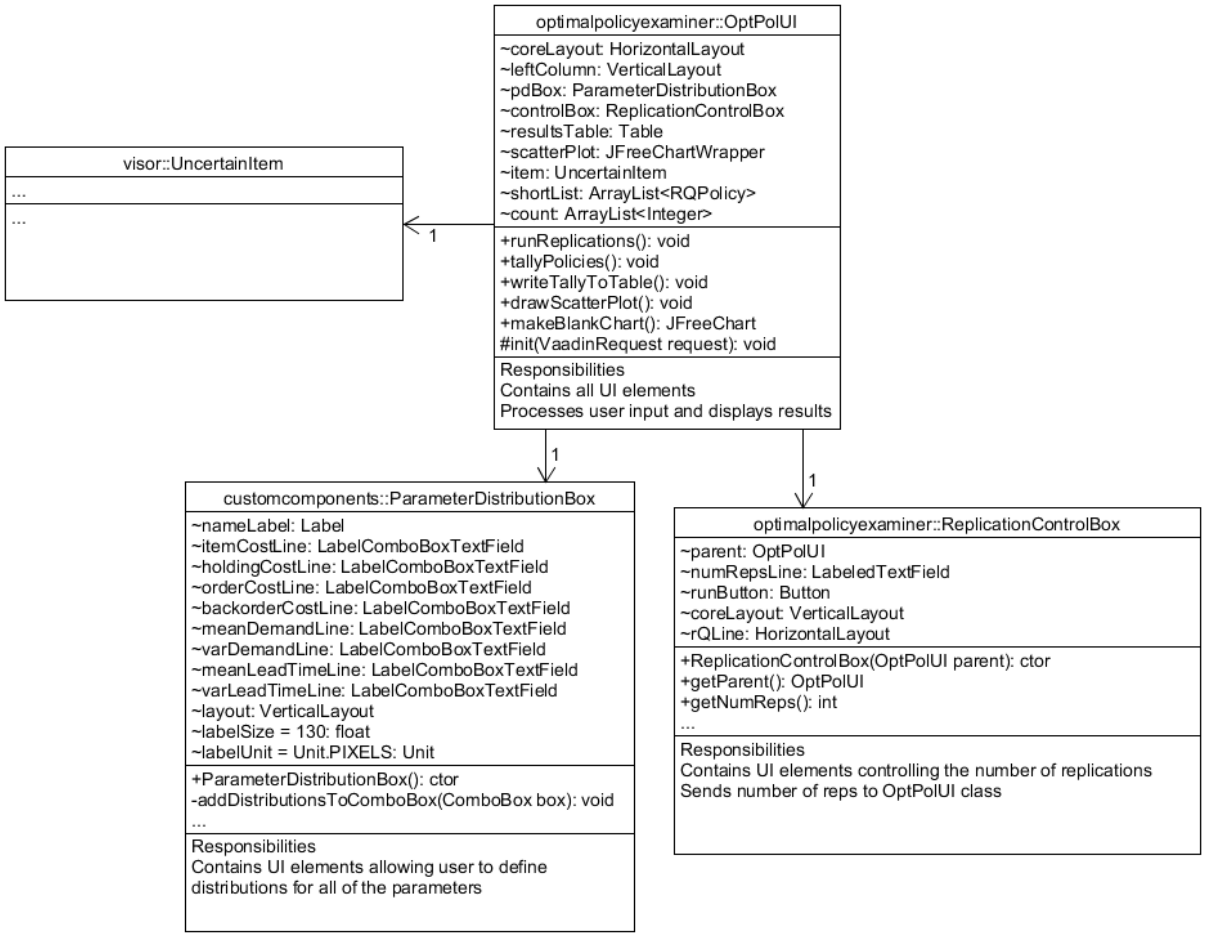


Figure 31 -- UML Diagram for Optimal Policy Examiner

The Optimal Policy Examiner has one unique custom GUI component, the ReplicationControlBox. This component has GUI components to control how many replications the module runs, how the results are displayed, and the button that actually runs the simulation.

Table 16 -- ReplicationControlBox methods

Method Signature	Return Type	Notes
public OptPolUI getParent()	OptPolUI	Returns the parent UI
public int getNumReps()	int	Returns the number of replications the user has specified.
public int getRBlock()	int	Returns the value of the r block text field. Determines the size of the bin policies will be sorted into.
public int getQBlock()	int	Returns the value of the Q block text field. Determines the size of the bin policies will be sorted into.

3.4.6.3 Parameter Comparison Module

The main class of the Parameter Comparison Module is the ParameterComparisonUI class. It contains all of the GUI elements for the module and controls the simulation. In addition to the GUI elements, the class has an ItemCollection object to store all the items the user has created and an ArrayList to store only those items that the user currently has selected. There is also a counter used to assign ID number to newly created items.

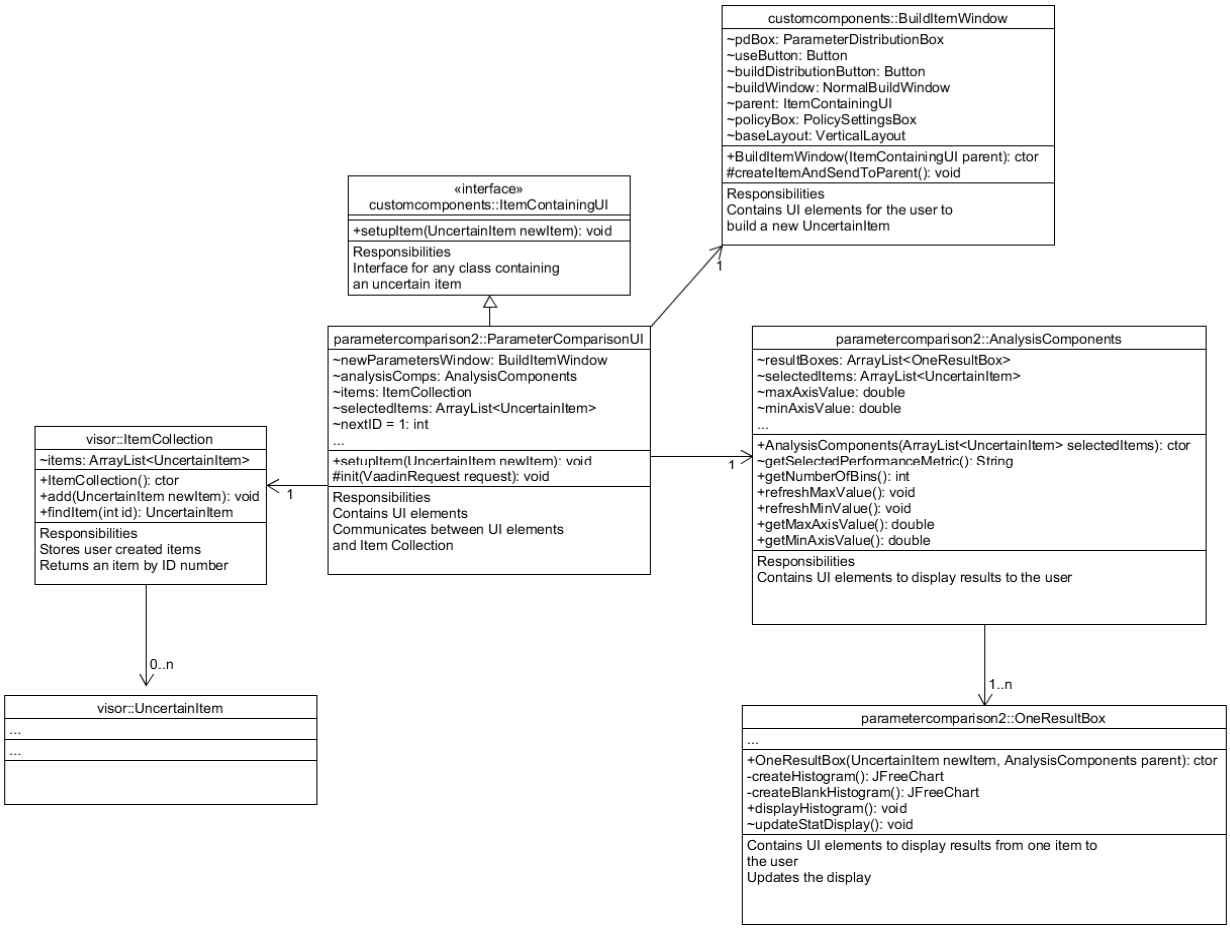


Figure 32 -- UML Diagram for Parameter ComparisonUI

Table 17 -- ParameterComparisonUI fields

Field Type and Name	**Notes**
ItemCollection items	Contains all of the user created UncertainItems
ArrayList<UncertainItem> selectedItems	Contains only the items the user currently has selected
static int nextID	A counter used to assign unique ID numbers to each item as it is created.

Table 18 -- ParameterComparisonUI methods

Method Signature	Return Type	Notes
protected void init(VaadinRequest request)	void	Runs when the UI is first opened. Creates all of the UI components
public void setupItem(UncertainItem newItem)	void	Adds newItem to the ItemCollection and the UI table.

The Parameter Comparison Module has two unique custom GUI components to display the results of the simulation. The first class is OneResultBox. It displays the summary statistics and histogram for one selected item. It also has a field to reference the UncertainItem it will be displaying. Its methods update the statistics and histogram.

Table 19 -- One Result Box methods

Method Signature	Return Type	Notes
private JFreeChart createHistogram()	JFreeChart	Creates a new histogram from the simulation results
private JFreeChart createBlankHistogram()	JFreeChart	Creates a blank histogram for when the OneResultBox is first created
publiv void displayHistogram()	void	Clears the current histogram, calls createHistogram() and displays the new chart
void updateStatDisplay()	void	Gets a Statistic object from a UncertainItem and updates the summary statistic labels

The second custom component is the AnalysisComponents class. It contains all of the GUI components to control how the results of the simulation are displayed. It also has fields to store references to the items the user has selected and an OneResultBox instance for every selected item. It has several methods called when the user presses the Refresh button.

Table 20 -- AnalysisComponents fields

Field Type and Name	**Notes**
ArrayList<OneResultBox> resultBoxes	Stores the OneResultBoxes created for each selected item
ArrayList<UncertainItem> selectedItems	Stores references to the items the user has selected
double maxAxisValue	The maximum value for the x axis of the histograms
double minAxisValue	The minimum value for the x axis of the histograms

Table 21 -- AnalysisComponents methods

Method Signature	**Return Type**	**Notes**
String getSelectedPerformanceMetric()	String	Returns the performance metric the user has selected
public int getNumberOfBins()	int	Returns the value the user has entered into the number of bins text field.
public void refreshMaxValue()	void	Finds and stores the maximum value for the selected performance metric among the items.
public void refreshMinValue()	void	Finds and stores the minimum value for the selected performance metric among the items
public double getMaxAxisValue()	double	Returns the maximum axis value for drawing the histograms
publid double getMinAxisValue()	double	Returns the minimum axis value for drawing the histograms

3.4.7 Utility Classes

One of the most important classes in the application is the UncertainItem class. It represents an item following a (r, Q) reordering policy but with random distributions in place of deterministic parameters. It also stores the deterministic QRItems during the simulation and can report the deterministic item's performance.

Table 22 -- UncertainItem fields

Field Type and Name	**Notes**
int id	An id number used for finding a particular instance
Distribution itemCost	Item cost distribution
Distribution holdingCost	Holding cost distribution
Distribution orderCost	Order cost distribution
Distribution backorderCost	Backorder cost distribution
Distribution meanDemand	Mean demand distribution
Distribution varDemand	Variance of Demand distribution
Distribution meanLeadTime	Mean lead time distribution
Distribution varLeadTime	Variance of lead time
ArrayList<QRItem> deterministicItems	The collection of deterministic items created by the simulation
double initialR	The reordering point
double initalQ	The reordering quantity

Table 23 -- UncertainItem methods

Method Signature	**Return Type**	**Notes**
public void addRandomItem(int numNewItems)	void	Adds numNewItems deterministic items to the collection. Random draws are made from the distributions for each parameter, and initialR and initalQ are used for the reordering policy
public void optimizePolicy()	void	Finds optimal reordering policies for all of the items in the collection
public RQPolicy[] getOptimizedPolicies()	RQPolicy[]	Returns the reordering policies for all of the deterministic items
public double[] getPerformanceData(String requestedMetric)	double[]	Returns the raw performance data for the requested metric
public Statistic getPerformanceStatistic(String requestedMetric)	Statistic	Returns the performance data for the requested metric as a Statistic object
public Statistic probSatisfiesInterval(String perfMetric, double lowerBound, double upperBound)	Statistic	Returns the probability that the selected performance metric falls between the lower and upper bounds
public Object[] getInterfaceTableValues()	Object[]	Returns an object array for displaying the Uncertain Item in a Vaadin table

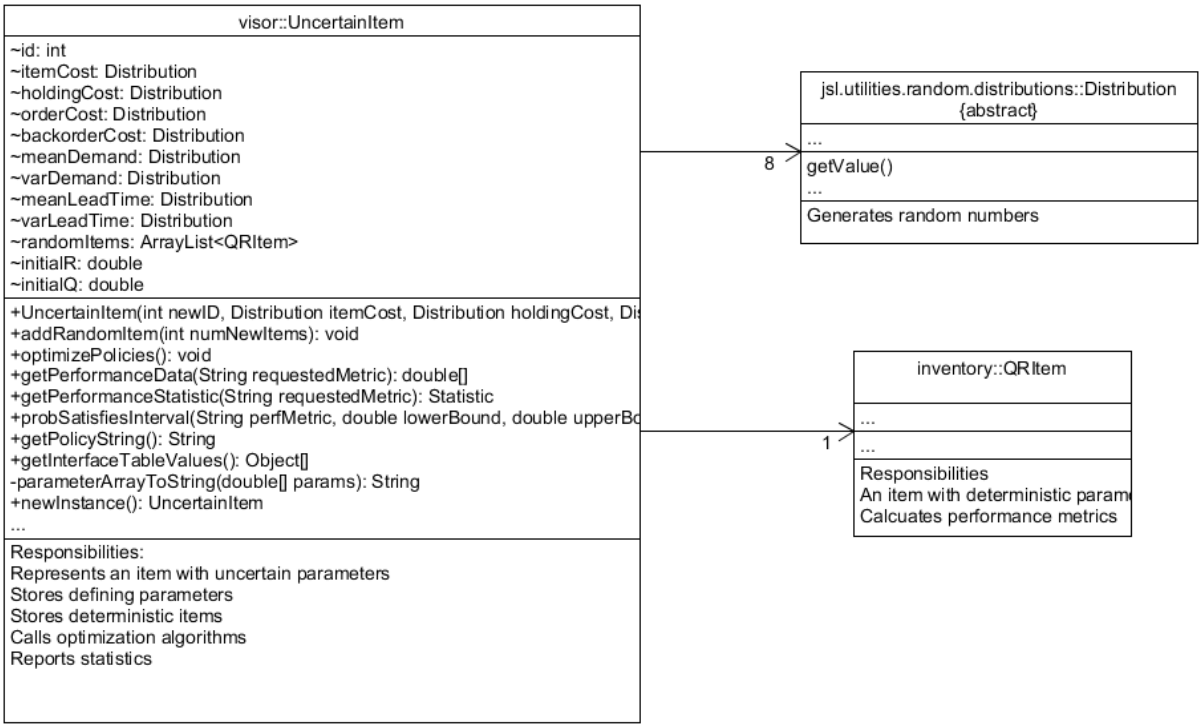


Figure 33 -- UML Diagram for UncertainItem class

Collections of UncertainItems are stored in the ItemCollection class. The class contains an ArrayList to store the UncertainItems and has methods to add more items and to find items based on the ID number.

Table 24 -- ItemCollection methods

Method Signature	**Return Type**	**Notes**
public void add(UncertainItem newItem)	void	Adds newItem to the ItemCollection's array
public UncertainItem findItem(int id)	UncertainItem	Returns the first item with the same ID number

Transforming user input into a distribution object is handled by the DistributionTranslator class. It is a static class with two methods, one to create new Distribution instances, and the other to display an error window if the user enter incorrect parameters.

Table 25 -- Distribution Translator methods

Method Signature	Return Type	Notes
public static Distribution translate(String name, String type, String parameters)	Distribution	Returns a Distribution of the correct type base on the parameters. Name is any string to ID the distribution, type is "Normal" "Triangular" etc. parameters are the comma delimitated parameters for the distribution
private static void displayDistributionTranslationErrorWindow(String badDistro)	void	Displays a window informing the user of an error and identifying the distribution with the error.

Using truncated normal distributions required building a class to represent them. The TruncatedNormal class meets these requirements. It extends the JSL TruncatedDistribution class. Its fields store the underlying Normal distribution object and other parameters required by the TruncatedDistribution class. It has methods to change the variance of the underlying distribution, to return a reference for the distribution, and to create a copy of the TruncatedNormal instance.

Table 26 -- TruncatedNormal fields

Field Type and Name	Notes
Normal distribution	The underlying distribution the truncated distribution is based on
double cdfLL	The lower limit of the CDF for the underlying distribution. Should always be negative infinity
double cdfUL	The upper limit of the CDF for the underlying distribution. Should always be infinity
double truncLL	The lower limit of the CDF of the truncated distribution
double truncUL	The upper limit of the CDF of the truncated distribution

Table 27 -- TruncatedNormal methods

Method Signature	Return Type	Notes
public void setVariance(double newVariance)	void	Changes the variance of the underlying normal distribution
public TruncatedNormal()	TruncatedNormal	Creates a copy of the TruncatedNormal instance
public Normal getDistribution()	Normal	Returns the underlying distribution

Truncated normal distributions are built by the static utility class TruncatedNormalDistributionBuilder. The class has one method to create a new TruncatedNormal instance from user input.

Table 28 -- TruncatedNormalDistributionBuilder methods

Method Signature	Return Type	Notes
public static TruncatedNormal buildTruncatedNormal(double mode, double lowerBound, double upperBound, double targetProb, double min, double max)	TruncatedNormal	Builds a distribution to match user input. Mode – most likely value, lowerBound – lower bound of a range, upperBound – upper bound of a range, targetProb – probability of a random number is within the range

3.4.8 Testing

There were three elements of the application that required testing: handling user input, the utility to build normal distributions, and the Monte Carlo simulation.

The user interface was tested by having the application print the types and parameters of the distributions the application created from user input to the server log. The log was compared to the user input to insure that the application was handling user input correctly.

Two small applications were developed to test the normal distribution utility. The first application started with a known truncated distribution and printed all the information the utility would require to build the distribution. The second application built a distribution from that information and printed the mean and variance of the distribution. The mean and variance was compared to the distribution in the first application to make sure the values were within a reasonable tolerance.

The Monte Carlo simulation logic in the application was tested by comparing the output of the application to a simulation run in Excel using Palisade @Risk. The variance and mean of

the two simulations were compared to one another to insure the application was working correctly.

3.4.9 Future Work

There are many ways that the web application could be improved. File input and output could be implemented. Currently all inputs must be entered in the browser by hand. This can be time consuming and does not allow the user to save work. Allowing the user to export simulation results would enable the user to use statistical software to more closely examine the data.

One possibility would be to implement (r, Q) models with intermittent demand. Currently, when the application is building the lead time demand distribution, it assumes that there are few periods where there is zero demand. When there a significant number of periods with zero demand, the lead time demand distribution needs to be modified. The Inventory Web Service that the application makes use of already implements these changes. The application would have to be modified so the user could supply the probability of zero demand.

The charts in the application could be improved. Currently the application uses an open source Java library to generate the charts as static images. Additional Vaadin packages include interactive and dynamic charts as well as different chart types that would show performance uncertainty more clearly.

4 Conclusion

This report has presented different methods to handle data variability, multiple data sources, and data validity issues. It has also reviewed different existing software solutions for addressing many of the issues caused by uncertainty in the supply chain and examined the gaps in existing software. A web based application was developed to address some of these gaps. Using the software, an analyst can easily run Monte Carlo simulations to better understand the

consequences of uncertainty in the supply chain. Using the application an analyst can determine if an item poses a significant risk of poor service levels or cost overruns. They can also determine the value of more accurate information, determine which parameter is contributing most to performance uncertainty, or find a range of near optimal ordering policies under uncertainty.

5 Works Cited

Alp Akcay, Bahar Biller, and Sridhar Tayur. "Overcoming the “Triple-Threat" in Managing Inventory with Limited History of Intermittent Demand." Diss. Carnegie Mellon University, 2012. Print.

Bleiholder, Jens, and Felix Naumann. "Data fusion." ACM Computing Surveys (CSUR) 41, no. 1 (2008): 1.

Cachon, Gérard P., and Marshall Fisher. "Supply chain inventory management and the value of shared information." Management science 46, no. 8 (2000): 1032-1048.

Chatfield, Dean C., Jeon G. Kim, Terry P. Harrison, and Jack C. Hayya. "The bullwhip effect—impact of stochastic lead time, information quality, and information sharing: a simulation study." Production and Operations Management 13, no. 4 (2004): 340-353.

Chen, Frank, Zvi Drezner, Jennifer K. Ryan, and David Simchi-Levi. "Quantifying the bullwhip effect in a simple supply chain: The impact of forecasting, lead times, and information." Management science 46, no. 3 (2000): 436-443.

Chopra, Sunil, and M. S. Sodhi. "Managing risk to avoid supply-chain breakdown." MIT Sloan Management Review (Fall 2004) (2012).

Chu, L. Y., J. G. Shanthikumar, Z. M. Shen. 2008. Solving operational statistics via a Bayesian analysis. Operations Research Letters 36(1) 110-116.

Cirtita, Horatiu, and Daniel A. Glaser-Segura. "Measuring downstream supply chain performance." Journal of Manufacturing Technology Management 23, no. 3 (2012): 299-314.

Croston, J. D. 1972. Forecasting and stock control for intermittent demands. Operational Research Quarterly 23(3) 289-303.

Croston, J. Do. "Forecasting and stock control for intermittent demands." Operational Research Quarterly (1972): 289-303.

Davenport, Thomas. *Enterprise Analytics: Optimize Performance, Process,and Decisions through Big Data*. 1st ed. Upper Saddle River, NJ: Pearson Education, Inc., 2012.

Dey, Debabrata, and Subodha Kumar. "Data Quality of Query Results with Generalized Selection Conditions." Operations Research 61, no. 1 (2013): 17-31.

Dey, Debabrata. "Record matching in data warehouses: a decision model for data consolidation." Operations Research 51, no. 2 (2003): 240-254.

Fang, Xin, Cheng Zhang, David J. Robb, and Joseph D. Blackburn. "Decision support for lead time and demand variability reduction." Omega 41, no. 2 (2013): 390-396.

Federgruen, Awi, and Min Wang. "Monotonicity properties of a class of stochastic inventory systems." Annals of Operations Research 208, no. 1 (2013): 155-186.

Handfield, Robert, Don Warsing, and Xinmin Wu. "(Q, r) Inventory policies in a fuzzy uncertain supply chain environment." European Journal of Operational Research 197, no. 2 (2009): 609-619.

Hayes, R. H. 1969. Statistical estimation problems in inventory control. Management Science 15(11) 686-701.

Heese, H. Sebastian. "Inventory record inaccuracy, double marginalization, and RFID adoption." Production and Operations Management 16, no. 5 (2007): 542-553.

Huh, W. T., R. Levi, P. Rusmevichientong, J. B. Orlin. 2011. Adaptive data-driven inventory control with censored demand based on Kaplan-Meier estimator. Operations Research 59(4) 929-941.

Kreye, Melanie E., Yee Mey Goh, Linda B. Newnes, and P. Goodwin. "Approaches to displaying information to assist decisions under uncertainty." Omega 40, no. 6 (2012): 682-692.

Lee, Hau L., Kut C. So, and Christopher S. Tang. "The value of information sharing in a two-level supply chain." Management science 46, no. 5 (2000): 626-643.

Lim, Sungmook. "A joint optimal pricing and order quantity model under parameter uncertainty and its practical implementation." Omega 41, no. 6 (2013): 998-1007.

Lowe, Timothy J., and Leroy B. Schwarz. "Parameter estimation for the EOQ lot size model: Minimax and expected value choices." Naval Research Logistics Quarterly 30, no. 2 (1983): 367-376.

Niranjan, Tarikere T., Stephan M. Wagner, and Vijay Aggarwal. "Measuring information distortion in real-world supply chains." International Journal of Production Research 49, no. 11 (2011): 3343-3362.

Ramamurthy, V., J. G. Shanthikumar, Z. M. Shen. 2012. Inventory policy with parametric demand: operational statistics, linear correction, and regression. Production and Operations Management 21(2) 291-308.

Ruiz-Torres, Alex J., and Farzad Mahmoodi. "Safety stock determination based on parametric lead time and demand information." International Journal of Production Research 48, no. 10 (2010): 2841-2857.

Shukla, Vinaya, Mohamed M. Naim, and Nina F. Thornhill. "Rogue seasonality detection in supply chains." International Journal of Production Economics 138, no. 2 (2012): 254-272.

Solyali, Oguz, Jean-François Cordeau, and Gilbert Laporte. "Robust inventory routing under demand uncertainty." Transportation Science 46, no. 3 (2012): 327-340.

Stadtler, Hartmut, and Christoph Kilger. *Supply Chain Management and Advanced Planning: Concepts, Models, Software, and Case Studies*. 4th ed. Berlin: Springer, 2008. 281-282.

Swaminathan, Jayashankar M., Stephen F. Smith, and Norman M. Sadeh. "Modeling supply chain dynamics: A multiagent approach*." Decision sciences 29, no. 3 (1998): 607-632.

Van der Vorst, Jack GAJ, and Adrie JM Beulens. "Identifying sources of uncertainty to generate supply chain redesign strategies." International Journal of Physical Distribution & Logistics Management 32, no. 6 (2002): 409-430.

Vujošević, Mirko, Dobrila Petrović, and Radivoj Petrović. "EOQ formula when inventory cost is fuzzy." International Journal of Production Economics 45, no. 1 (1996): 499-504.

Waller, Matthew A., and Stanley E. Fawcett. "Data science, predictive analytics, and big data: a revolution that will transform supply chain design and management." Journal of Business Logistics 34, no. 2 (2013): 77-84.

Wan, Xiang, and Philip T. Evers. "Supply chain networks with multiple retailers: a test of the emerging theory on inventories, stockouts, and bullwhips." Journal of Business Logistics 32, no. 1 (2011): 27-39.

Wang, Juite, and Yun-Feng Shu. "Fuzzy decision modeling for supply chain management." Fuzzy sets and systems 150, no. 1 (2005): 107-127.

Wang, Shanshan. "Modeling Supply Chain Dynamics with Calibrated Simulation Using Data Fusion." PhD diss., Arizona State University, 2010.

Willemain, T. R., C. N. Smart, J. H. Shockor, P. A. DeSautels. 1994. Forecasting intermittent demand in manufacturing: a comparative evaluation of Croston's method. International Journal of Forecasting. 10(4) 529-538.

Zhang, Xiaolong. "Delayed demand information and dampened bullwhip effect." Operations Research Letters 33, no. 3 (2005): 289-294.

Zheng, Yu-Sheng. "On properties of stochastic inventory systems."Management Science 38, no. 1 (1992): 87-103.

Zhou, Li, and Stephen M. Disney. "Bullwhip and inventory variance in a closed loop supply chain." OR Spectrum 28, no. 1 (2006): 127-149.