REDUCTION / ELIMINATION OF ERRORS IN COST ESTIMATES USING CALIBRATION -
AN ALGORITHMIC APPROACH

A thesis presented to
the faculty of
Fritz J. and Dolores H. Russ College of Engineering and Technology
of
Ohio University

In partial fulfillment
of the requirements for the degree
Master of Science

Raju Gandhi
November 2005
This thesis entitled

REDUCTION / ELIMINATION OF ERRORS IN COST ESTIMATES USING CALIBRATION - AN ALGORITHMIC APPROACH

by

RAJU GANDHI

has been approved for

the Department of Industrial and Manufacturing Systems Engineering

and the Fritz J. and Dolores H. Russ College of Engineering and Technology

David A. Koonce

Associate Professor of Industrial and Manufacturing Systems Engineering

Dennis Irwin

Dean, Fritz J. and Dolores H. Russ College of Engineering and Technology
In estimating the cost of goods or services, errors between the estimated value and the actual cost should be expected. In this thesis, the assumption that this estimate error is consistent with similar goods and services is exploited to develop a calibration method for cost estimates produced with the cost estimator software system developed at Ohio University. This document discusses the various factors that need to be considered when implementing calibration in a hierarchical cost structure using multiple cost pools. An algorithm to implement calibration has been detailed along with pseudo-code. The end of the document describes how calibration can be applied to an estimate.

Approved:

David A. Koonce

Associate Professor of Industrial and Manufacturing Systems Engineering
# Table of Contents

1. Introduction............................................................................................................... 8  
   1.1. Need For Business .............................................................................................. 8  
   1.2. Federated Intelligent Product EnviRonment (FIPER) ......................................... 8  
   1.3. The Cost Estimator ............................................................................................ 9  
   1.4. Objectives ......................................................................................................... 9  
2. Background.............................................................................................................. 11  
   2.1. Cost Estimation Methods .................................................................................. 11  
   2.2. Cost Estimating Tools ...................................................................................... 19  
   2.3. FIPER ............................................................................................................. 23  
3. Calibration ............................................................................................................. 26  
   3.1. Types of Costs .................................................................................................. 26  
   3.2. Errors/Variability in Estimates ......................................................................... 26  
   3.3. Assumptions ..................................................................................................... 29  
   3.4. Purpose of Calibration ..................................................................................... 30  
   3.5. Calibration in the WBS .................................................................................... 33  
   3.6. An Algorithmic Approach to Calibration ....................................................... 41  
4. Calibration – an Implementation .......................................................................... 45  
   4.1. Introduction ...................................................................................................... 45  
   4.2. Requirements .................................................................................................. 45  
5. Calibrating Cost Estimates ................................................................................... 61  
   5.1. User Interface Design and Functionality of the Cost Estimator ....................... 61
5.2. Calibrating an Element.......................................................................................... 63

6. Future Work and Conclusions............................................................................... 69

6.1. Future Work.................................................................................................. 69

6.2. Conclusions................................................................................................... 70

References.............................................................................................................. 71
List of Figures

Figure 3.1 - WBS of a Binder Clip ................................................................. 30
Figure 3.2 - InitializeCalibration ................................................................. 42
Figure 3.3 - PushCostPoolRatios ................................................................. 42
Figure 3.4 - SetCostPoolRatios ................................................................. 43
Figure 4.1 - Weightage Calculation .......................................................... 58
Figure 5.1 - Cost Estimator Layout ......................................................... 62
Figure 5.2 - Setting up Calibration ............................................................. 64
Figure 5.3 - Calibration ........................................................................... 65
Figure 5.4 - Adding a Filter .......................................................... 66
Figure 5.5 - Calibration Status in the WBS Panel ..................................... 68
List of Tables

Table 3.1 - Cost Structure ..................................................................................................... 35
Table 3.2 - Applying Calibration at the Root Node .............................................................. 35
Table 3.3 - Pushing down the Scale Factor to the Children ................................................ 36
Table 3.4 - Interdependencies between Cost Pools.............................................................. 38
1. INTRODUCTION

1.1. NEED FOR BUSINESS

In today's world, accomplishments and breakthroughs in the areas of communication and information technology have created vast opportunities for an enterprise to enhance its growth and make its place in the world market. These technological endeavors and achievements have also made enterprises susceptible to immense competition. This is evident from the development of various techniques and strategies, such as TQM, re-engineering, etc., in order to improve efficiencies and increase product quality [1]. An organization, in order to remain profitable, should be able to deliver products that are competitively priced, of a certain quality standard. A good product cost estimate is crucial to this [3].

Cost estimation deals with forecasting costs related to a set of activities before they have been implemented [2]. It provides an organization with valuable information regarding the cost of products and processes during the design phase. It allows the organization to consider alternate product designs, or decide whether to make or buy a part. An accurate estimate is necessary when an organization is considering a make-or-buy decision, attempting to submit a quote, or considering manufacturability and marketability of a product [3].

1.2. FEDERATED INTELLIGENT PRODUCT ENVIRONMENT (FIPER)

The Federated Intelligent Product Environment (FIPER) was a $21.5 million project which created a framework that utilizes best-of-breed technologies to reduce design,
design-to-manufacture and manufacturing times and costs while ensuring higher levels of product quality and reliability. The FIPER project involved the collaboration of several partners, including General Electric, Engineous Software, Goodrich, Parker-Hannifin, Ohio Aerospace Institute and Ohio University.

The Ohio University FIPER team developed two application modules for FIPER. These deliverables are the Data Exchanger – which allows for data translation and exchange between ASCII files, and the Cost Estimator – a highly customizable cost estimation tool. This tool was developed using the Java Development Kit (JDK) 1.3.

1.3. The Cost Estimator

The cost estimator comprises an underlying cost engine, a graphical user interface and an element builder. The cost estimator is a hierarchical cost tool, in that products or processes are represented as a tree (hereon referred to as the Work Breakdown Structure or WBS) with the final product being the root element, and the constituent sub-assemblies, parts and manufacturing processes being the children of the root. The children themselves may be branches in the tree with leaf elements of their own.

1.4. Objectives

Calibration is the act of adjusting an estimate to account for anticipated bias. Calibration utilizes the variance observed between the estimated value and the actual value of known part to reduce or eliminate the error in future estimates.
The objective of this thesis was multifold.

1. To develop a calibration algorithm that allows the user to calibrate a node in hierarchical cost structure

2. To integrate the algorithm within the existing architecture of the Cost Estimator

3. To implement the algorithm so that a user can calibrate one or more elements in an hierarchical cost structure across one or more cost pools

4. To implement the algorithm in a manner so that it is transparent to the element developer, in that the developer need not write additional code to cater for calibration

5. To develop a structured format to act as a repository for calibration data and to provide a search algorithm to search for similar parts using user defined criteria
2. BACKGROUND

When a product is being designed, it is often useful to have an accurate estimate of the cost. Studies have shown that the as much as 80% of the product’s cost is determined at the design phase [14]. Therefore, it is important that designers be provided with a method that gives them accurate estimates on the financial consequences of their decisions, thus giving them a good guideline to evaluate different design alternatives. However, varying amount of information is available regarding the physical characteristics of the product during design. This is the role of cost estimation.

2.1. Cost Estimation Methods

In the past, a variety of methods have been used to estimate the manufacturing cost of a product during the design phase. Duverlie and Castelain [4] have identified four basic methods for evaluating costs – intuitive, analogical, parametric and analytical.

Depending on the context and how far the organization is in the design phase, some methods are more applicable than others.

The intuitive method relies on the experience and knowledge of the estimator to predict an accurate cost. This nature of intuitive estimation is ill structured. In many cases, it involves completing partial design descriptions and building process plans for the same. Adding to this, it is difficult to know if the design description is complete or the process plan is accurate for the same [14].
The analogical method is a comparative method that relies on similarities between the part being evaluated, and other parts. It may find other parts based on a group technology code [5]. That is, the cost recorded for manufacturing similar parts can be used as a template when attempting to economically evaluate a new part [2]. The parametric method uses certain parameters, or key part attributes, known as cost drivers. A parametric estimate may use very high level attributes or may rely on detailed geometric data. An analytical estimate uses the summation of steps used in the manufacturing process.

### 2.1.1. Parametric Method

The parametric method is often referred to as a mathematical approach to estimating cost [8]. It makes use of part characteristics, both physical and qualitative, or parameters, and known resources utilized for the development, manufacture and/or modification of a product. Parameters drive the cost, or are the cost drivers, of the product or service being estimated [6]. Examples of parameters are material, weight, etc. Equations, also known as Cost Estimation Relationships (CERs), are used to express the cost as a function of one or more cost drivers. CERs can be of varying complexity, and a set of CERs used to describe a product is often referred to as a model.

Some CERs can be generated using statistical techniques such as linear or nonlinear regression algorithms; the intercepts or coefficients for the CERs are then computed [7]. One requirement, hence, is the creation and maintenance of reliable databases that reflect the cost history of the organization, or historical cost and non-cost data to accurately predict cost relationships. As a result the accuracy of the estimate depends
on the reliability of this data. Further, if such data is not available, this estimating methodology cannot be effectively exploited.

Consider the example of manufacturing a bolt. The cost drivers in this case would be dimensions, material of the bolt, part tolerances, number of parts or batches, number of production units, etc. Each of these, or a combination of these would serve to drive the cost in the form of CERs.

2.1.1.1. Advantages and Disadvantages

The parametric method proves to be a quick method for design evaluation [4]. During the design stage, not all of the information is available. Sometimes, for the model to work, the estimator has to assume certain values for the estimate to be calculated. This allows the designer to quickly perform a quick evaluation of the design; while at the same time may cause uncertainty to be introduced. One important advantage of this method is that it directly ties attributes of the part to cost. Hence the designers know the cost implications of their decisions, encouraging consideration of alternative designs and optimization of the same.

A major issue in the implementation of parametric cost deployment is the necessity for an accurate and substantial database [8]. Assembling data necessary to produce an estimate, and ensuring that the data is reasonable and up-to-date is critical, and often proves to be time-consuming step [6]. The lack of one such data repository or having one that has not been maintained and/or updated has potential in undermining the validity of the CERs.
Again, CERs may not be pertinent to all cases. Sometimes one or more parameters that do not exist as cost drivers in the CERs may prove to have a large effect on the cost. Sometimes, the estimate may be adversely affected by the lack of detailed information. CERs are sometimes too simplistic to forecast costs. Often when detailed information is available, using other methods of estimation that CERs may prove to be more fitting [6].

2.1.2. **ANALOGICAL METHOD**

The analogical method of cost estimation uses historical data to estimate the cost of a part or a process. It involves evaluating the similarity between the new design, and designs in the past to predict the manufacturing cost. One application of the same is case based reasoning (CBR).

The CBR process is made up of four steps [9] - retrieve, evaluate, adapt and learn. In CBR, a case is a representation of an experience. Based on the certain criteria cases are searched to find, or retrieve, the ones that are most relevant to the current situation. The system, or the user, evaluates the cases to determine the degree of modification required to adapt the previous case to the current one, which is then stored or learned by the system.

2.1.2.1. **Advantages and Disadvantages**

CBR offers several advantages [9]. It allows managers and engineers across functions to record their design decisions in a structured manner, and increases the transparency of the company’s project experiences. It also provides an increase in the availability of
design information, thus helping the organization move towards a more conducive concurrent engineering culture.

Such a system can be costly to develop and implement and depends on the willingness of people to use and improve the system on a daily basis [9]. Another disadvantage of this type of system is that it relies on a search algorithm to locate a similar case(s). This search is crucial to providing the designer with the correct scenario(s) in order to predict the cost of the part.

2.1.3. Analytical Method

The analytical method attempts to break down the product with respect to the manufacturing operations required, find the cost of each process and hence evaluate the product cost. This method is usually applied when attempting to find the cost of a product for which no similar part exists, and hence the analogical method is not applicable [2].

In process planning, every decision made is subject to the constraints imposed by the decisions made earlier. For example, the selection of a particular type of material will restrict the manufacturing processes that can be used. Hence it is critical that every decision made is evaluated for its economic impact. The analytical method, is closely tied with process planning, and allows for such calculations.
2.1.3.1. **Advantages and Disadvantages**

The analytical method allows designers to evaluate the manufacturability of the product, as well as the cost associated with the same. It allows for an extremely detailed and precise cost estimate for the part. Further, for parts that cannot be evaluated using the analogical method, this method proves to be applicable [2]. A disadvantage of this method is that it requires a significant amount of detail to generate an estimate which can be a labor intensive and time-consuming process [27]. Duverlie and Castelain [4] conclude that the parametric and analogical methods are the most suitable methods during the design phase, since not all of the information is available when attempting to make an economic evaluation of the design.

The cost estimation tool developed by the Ohio University FIPER team can use a combination of generative (analytical) and variant (analogical) costing, and designs are estimated using either a work breakdown structure, or a parametric estimation method [5]. The cost estimator uses templates, a set of elements which may hold attributes that are parameters to be used in a parametric estimate. Or the element might hold constituent elements that represent manufacturing operations or a combination of the two.

2.1.4. **Other Methods**

A newer methodology for cost estimation, which includes the cost of the elements of manufacturing processes in the cost of the part has been introduced [5], [10]. Traditional cost accounting (TCA), such as those explained, assume that an organization
produces a particular part or product in large quantities over a period of time, which facilitates the development of knowledge systems which can then be used to estimate the cost of future products. On the other hand, in order to survive in today’s highly competitive marketplaces, companies have to offer a wide variety of products, which results in investing in advanced manufacturing systems (AMS). One effect of an AMS on costing is that direct costs are greatly reduced and overhead costs increase [11]. Cooper and Kaplan and Johnson and Kaplan argue that the majority of factors that drive the cost are a result of the complexity in the plants’ operations [19] [20] [21]. But under this cost scheme, labor based TCA systems pose as a poor indicator of the actual overhead of the manufacturing processes [11]. TCA methods do not completely describe the business processes involved in manufacturing and provide a poor picture of the actual cost drivers [10]. TCA methods allocate overhead costs in proportion to the production volume, thus distorting the product costs [19]. As a result, an impressive range of alternative costing models have been sought, and introduced to remedy this situation. These costing models have been developed fairly recently, and have already been given a lot of attention [10]. Some of the models such as activity-based costing (ABC); target costing and life-cycle costing will be discussed in the following sections.

2.1.4.1. Activity Based Costing (ABC)

ABC assumes that manufacturing activities consume resources, and manufacturing products require those activities. Models such as ABC attempt to break down the costs into categories such as direct material and labor costs, activity costs such as processing, tooling and setup, as well as non-activity costs such as waiting time cost and idle time.
There are two ways proposed to understand the structure of overhead costs. Miller and Vollmann [22] categorize overhead as “logistical, balancing, quality and change. Logistical changes involve the receipt and movement of materials through the plant. Balancing involves the coordination of supply and demand of resources in production. Quality transactions are to ensure that the quality of the product reaching the customer is satisfactory. Change transactions are used when revising manufacturing processes in response to change in product or process specifications.”

Another framework proposed by Cooper and Kaplan [23] suggest an activity driven categorization – “unit-level, batch-level, product sustaining, and facility sustaining activities. Unit-level activities are those that are involved in the manufacture of one unit, and these are proportional to the number of units manufactured, like quality inspection. Batch-level activities are those that correspond to the number of batches produced, like setting up of machines, but are independent of the total number of units actually produced. Product sustaining activities are those that undertaken to enable specific products to be manufactured, like product or process changes. Facility sustaining activates are those that are related to plant management and maintenance.”

2.1.4.2. Target Costing

Target Costing (TC) is an important area where accounting and management overlap. Using this strategy, the marketing and design functions of an organization, identify the features of a new product, and attempt to establish its selling price. From this selling price, the desired profit margin is subtracted, and the resultant cost is set to be the target. All functions in the organization work to design and manufacture the product
within the target cost. In essence, target costing attempts to reduce the cost of manufacturing a part or a product. It maintains that cost is a function of several decisions made during the development of a product, and hence should be assessed when making design, product or process development.

2.1.4.3. Life-cycle Costing

A product’s design not only determines its cost, manufacturability and the logistics required, but also its maintenance as well as ease and cost of disposal. With the growing awareness of environmental issues and laws, an organization has not only account for the amount the end user will spend on the product’s maintenance, but also the cost incurred for recycling and/or disposing the product. The estimation of the cost of the product, throughout its life cycle, is termed as Life Cycle Costing (LCC).

2.2. Cost Estimating Tools

A number of cost estimating tools are commercially available using one or more of the models described earlier, and yet many manufacturing firms have developed their own proprietary systems [5][6]. These systems range from simple spread-sheet applications to completely automated interactive systems. Their estimation abilities range from being highly customized for a particular part or assembly to being generic systems that can cost virtually any manufactured part. Often, companies develop proprietary systems owing to the fact that their specific cost estimation needs cannot be easily satisfied by using commercial models. The proper use of a validated proprietary system helps to increase the efficiency of the estimating process, as well as improving the quality,
accuracy and consistency of the estimates produced [6]. Further, the models developed already encapsulate the cost history of the organization. This means that the model developed is already adjusted based on the organization’s cost history, and hence the need for calibrating the model is eliminated [6].

Several specific systems developed by a variety of companies involved in defense contracting have been described in [6]. These systems include calculating program management costs, system engineering costs and system acquisition costs. Their inputs include quantitative parameters such as quantity, weight and qualitative parameters such as environmental specifications and scheduling parameters such as manufacturing rate.

2.2.1. PRICE H® Model

PRICE H® is a commercially available cost model developed by Price Systems, as part of their PRICE TruePlanning™ suite [15]. It is used to provide system acquisition cost estimates, resources and schedules for hardware projects. It was developed to operate with limited equipment so that many alternatives can be explored before a design is finalized. It uses parameters such as weight and size as its primary descriptors, to provide estimates for items that are more difficult to quantify such as cost and production schedules. The number of inputs to the system is minimalist, and the system attempts to deduce all non-supplied input values.
PRICE H® uses a parametric approach to estimation. As mentioned earlier, instead of using complex parts lists and labor resource charts as parameters, it uses characteristics that can be easily quantified. CERs form the underlying cost engine for estimation.

PRICE H® provides outputs that are categorized according to actual resources, such as design, manufacturing and tooling/test equipment. This information is provided for review both in tabular as well as graphical format. It also has functionality to directly feed the output into other information tools like ERP systems and databases.

### 2.2.1.1. Eurocopter - A case study

Eurocopter, Europe’s largest manufacturer of civilian and military helicopters, recently used the PRICE H® model to evaluate a particular vendor’s bid [16]. The vendor supplied Eurocopter with a proposal for the unit cost of a critical component. Using the PRICE H®, Eurocopter performed a cost analysis in five days, and came to the conclusion that the proposed cost was excessive. Using PRICE H®, Eurocopter was able to support their claims of a cost overrun, and renegotiate the price with the vendor. This amounted to savings of $1.1 million over a period of 10 years.

### 2.2.2. COMPEAT$™

Cost Offering Method for Affordable Propulsion Engineering Acquisition and Test (COMPEAT$™), is another highly complex proprietary model. It can be used for many purposes including calculating life cycle and target costing for jet engines. This model has been employed by General Electric Aircraft Engines. It utilizes common part characteristics such as size, part features, dimensions and efficiency to find a historical
match, and depending on the similarity between the two parts, processes the inputs through a series of CERs to calculate a cost [6].

2.2.3. SEER-DFM

SEER-DFM, a commercial tool by Galorath Incorporated, is a part of their suite of cost estimation tools, which can be used for cost and manufacturability analysis [17]. It allows an organization to manage the large number of variables that influence manufacturing processes such as costs and process and production variables.

SEER-DFM also uses a parametric approach to cost estimation. Examples of the types of parameters that the system takes as inputs include processes such as mechanical and electrical assembly, fabrication, plastic molding, finishing and welding as well as production quantities, labor and overhead rates and material costs.

SEER-DFM can be used for a variety of tasks such as

- Optimizing manufacturing and assembly methods
- Cutting costs early in the design and manufacturing cycle
- Identifying manufacturing and assembly cost drivers

The reporting capabilities include the ability to customize the output format. It also has a Quick Estimate Report option that provides a brief summary of the overall estimate for quick analysis.
2.2.3.1. **Lockheed Martin - A case study**

Lockheed Martin was looking to reduce the time required for performing cost estimates, while increasing accuracy. The traditional method for determining the cost of a new airframe involved searching a database to find a matching part. It took three analysts one week to estimate the cost of an airframe, and to evaluate multiple combinations of design alternatives, it took two to five times longer.

Using SEER-DFM, Lockheed reduced the time to perform cost estimates for airframe components by 75%. The system searches for similar parts by using CAD parameters such as length, width, thickness and generates an estimate. Further, the cost estimates prove to be accurate with +/- five percent [12].

2.3. **FIPER**

Over the past decade, the design process has manifested itself into an extremely complex process, where a variety of tools and methodologies are used to deliver high performance and quality products to the customer [13]. Products are now designed using the latest technologies so as to not only meet the expectations of the customer, but exceed them. A wide range of software tools are now available for the various activities involved in multidisciplinary analysis and design. Examples include tools for Design of Experiments to identify crucial design parameters, design optimization tools, and testing for reliability and robustness. Despite the existence of many such tools, engineers find it difficult to use them in a coordinated fashion. Several organizations are in the process of developing such an environment that provides easy access and use of
all such tools [13]. But the problem lies not in providing one all-encompassing solution, but rather a framework that supports plug-and-play functionality. The Federated Intelligent Product Environment (FIPER) is one such effort, which allows engineers to incorporate their choice of best-of-breed design and analysis tools, as well as exploit available computing resources to execute them [13].

FIPER is a framework, by Engineous Software, in conjunction with General Electric, Goodrich, Parker Hannifin, Ohio University, the Ohio Aerospace Institute, and Stanford University. It was a four year collaborative effort, sponsored by the National Institute for Standards and Technology (NIST) Advanced Technology Program (ATP).

2.3.1. FIPER ARCHITECTURE

FIPER has been designed to provide a framework that allows engineers to incorporate their choice of design tools into the environment, and permit interaction and flow of information with and between these tools. FIPER consists of three layers. The Core Infrastructure [13], lies at the heart of the FIPER architecture, and acts as a repository of services for communication and storage of data. This has been implemented by Engineous Incorporated, and with development and improvements provided by General Electric Global Research Center. The second layer, Core Extensions [13] that provides the plug and play functionality of FIPER, so that organizations can plug any software that they would normally use to access the functionality like resource management offered by FIPER. This provides the organization an easy transition to FIPER without having to rework their underlying IT infrastructure. It is the outermost layer, or the
Application Components [13] that the end user of FIPER would interact with. FIPER allows users to incorporate their choice of design tools into the environment.

FIPER is a component-based [13] framework in that users can develop their own modules or components to provide functionality that serves to solve their specific problems. These components can then be published [13] as specific services, by wrapping them in a standard Java wrapper. XML descriptors are used to describe how these services are to be used, and the kinds of inputs and outputs that they expose to the environment. For example, the Ohio University FIPER team has developed a cost estimation tool, which can be used either as a stand-alone application, or as a component within FIPER. FIPER also provides a set of Application Programming Interfaces (APIs), for tools that have complex forms of interactions, such as inputs and outputs.

FIPER supports a workflow methodology. The design process often entails the use of a variety of tools and applications, such as Design of Experiments and optimization tools. These are used in a particular order, so that one tool's output serves as the inputs of another. FIPER allows designers and engineers to model this flow in a logical fashion, through a drag-and-drop mechanism, allowing them to specify the control flow as well as information flow between components.
3. CALIBRATION

3.1. TYPES OF COSTS

Cost estimation deals with calculating the resources and the associated costs in designing, manufacturing and marketing a product [3]. According to Winchell [3] and Kingsman and Souza [28], costs can be classified in different ways – direct and indirect costs, standard and actual costs, and costs that are related to product volume, such as fixed, variable and step variable. Considering direct and indirect costs, direct costs are sub-categorized as direct labor and material. Direct labor reflects the wages of the personnel involved directly in the manufacturing process, while direct material is the cost of the material that will, at some point, be consumed to form a part of the final product. Indirect costs can be sub-categorized as tooling, design, costs of production planning, supervision and administrative costs. When attempting to draw an economic evaluation of a product, time is usually of essence [3] [29]. Hence, not only is product cost estimation an extremely complex process, and may be subject to variability, as discussed in subsequent sections.

3.2. ERRORS/VARIABILITY IN ESTIMATES

There are several reasons for errors in product cost estimates. With respect to the product itself, there is often a lack of specifications regarding the geometry, features and functional requirements. Also, indirect costs such as tooling and administrative costs which appear as overhead need to be appropriately allocated. Hence, cost estimation
proves to be a complex process requiring both technical knowledge as well as managerial experience.

Methods for cost estimation, including proprietary models available, tend to provide point estimates, that is they tend to be more go-or-no-go versus more or less [30]. Considering that at the early design phase, there may be limited details available, and although methods like parametric estimation may be used for quick economic evaluations, it often forces designers to assume certain values, which leads to uncertainty in the estimate.

Although an organization might have professional cost estimators, these are often not designers, in that they do not have extensive design experience, and might not be an integral part of the design process. Professional cost estimators tend to spend more time and effort on their estimates, and they have more experience and expertise in this field than designers [29]. Designers perform cost estimation as one way of understanding the financial impact of their decisions, while attempting to find more cost effective and optimal solutions. Cost estimation is not their primary function, and keeping the above mentioned limitations, may lead to inaccuracies in the cost estimate.

Another aspect that leads to inaccuracies in early design cost estimation is that invariably a lack of specifications is provided by the customer [28]. The customer may not know exactly what is needed, and sometimes may leave portions of the requirements vague [28]. Hence, designers have to prepare the cost estimate with incomplete specifications, leading to errors in the estimate.
Weustink, Brinke, Streppel and Kals [2] have identified four characteristics that make up the product cost - geometry, material, production processes and production planning. Depending on how far the design is evolved, some or all of these may not have been finalized. Consider the case of tolerance requirements. Estimating the time and resources required to achieve a particular fit proves to be a time-consuming and error prone [28] process.

Invariably, all the factors that drive the cost cannot be fully evaluated [3]. Manufacturing itself has so many parameters that have uncertain values. In the early stages of the design process so little is known of the product that a process plan may not have been formulated. Furthermore, sometimes a design may use a manufacturing base that has little or no observed data associated with it, or might use a completely new manufacturing process altogether. Within the manufacturing realm, some factors cannot be foreseen such as delays caused by defective materials, machine breakdowns, unexpected production planning problems and union and labor issues.

Another factor that makes cost estimation difficult is adjusting for inflation. The rapid pace of technology makes components like integrated circuits cheaper and smaller, while globalization can make hard-to-get and expensive raw materials, easier and cheaper. This is especially true for long term contracts, since predicting the cost of a part and its constituents over a longer period of time can prove to be very inaccurate [3].

The cost system that an organization employs may lead to inaccuracies in the cost estimate, such as traditional the cost accounting models as explained in 2.1.4, those
that allocate overhead based on production volume. Products that utilize automated facilities make cost estimation hard as it is difficult to allocate indirect costs [28].

3.3. Assumptions

In order to build a cost estimate for a part or product, a product structure is used. A product may be an assembly of several other sub-assemblies and/or parts. A hierarchical structure or WBS (work break down structure) can then be constructed, with the final product being the top or root node, and each of the constituent assemblies or parts forming child nodes or leaf nodes. Each node is known as an Element. A node can also represent a manufacturing processes such as turning or milling, as well as non-machining processes such as inspection. Consider the example of a binder clip. The top node of this assembly would be the binder clip itself, with two children, a Clamp element and an Arm element. The Clamp element in turn could have child element that represent manufacturing processes such as Drawing and Crimping and cost pools such as Material, Labor. This is shown in Figure 3.1.

Hence, in this structure, each node, or element, is any entity that incurs cost. Each of the leaf nodes will have a number of associated cost pools, and the parent nodes represent the roll up of costs in each cost pool of their children. This is not to say that the parent cannot have a cost pool that is not represented in any of its children. For example, the root node could have an “Estimated Price”, which is a sideways calculation of the “Total Cost” and a certain profit margin. Total Cost might represent the total cost of “Labor” and “Material”, which represent the summation of the labor and material cost for the entire tree.
Cost pools are calculated at each level in the tree, using cost estimation models such as parametric or otherwise, or are the accumulated values of the children that it contains, as well as costs incurred on that part itself, like quality inspection costs.

The cost structure may also evolve over time, as more of the specifications of the product are laid out and finalized. Initially, the tree may consist only of the final product, acting both as the root and a leaf, with a few parameters driving its cost. As the design evolves, more nodes can be added, thus achieving a higher level of fidelity.

3.4. PURPOSE OF CALIBRATION

As discussed earlier, the process of cost estimation proves to be crucial to the quality of the estimate, and without considerable investment of effort and time, has potential for errors. There are two functions in an organization that are concerned with the cost of a
product - cost estimating and cost accounting. Cost estimation involves predicting what a particular product will cost, while cost accounting is a function that deals with recording what the product actually cost. Not only does accounting reflect the efficiencies of manufacturing activities [3], but it exposes inaccuracies in the cost methodologies employed by the organization. By identifying, quantifying and allowing designers to account for this variance when performing the cost evaluation of a product, calibration is a method to reduce or eliminate error in cost estimates of future products.

Calibration builds on the premise of the analogical method (Section 2.1.2) of cost estimation. The analogical method uses a similar part to predict the cost of a newer design. The underlying assumption here is that if two products are similar then their costs are comparable. Taking this a step further, if two similar parts were estimated using a similar methodology then the variance between their estimated costs should be comparable too. Hence, the variance observed between the actual and estimated cost of the first part should be comparable to the variance that will be observed between the actual and estimated values of the second.

Calibration enables the estimator to utilize this variance when estimating and developing new designs. By considering the variance between the estimated and actual value of a known part, the estimator can potentially arrive at a more realistic cost estimate for the design. Calibration involves scaling the estimated cost by the relationship shown in equation 1.
Here $\text{SCALE}_{\text{MATCH}}$ is

$$\text{SCALE}_{\text{MATCH}} = \frac{\text{ACTUAL}_{\text{MATCH}}}{\text{ESTIMATED}_{\text{MATCH}}}$$  \hspace{1cm} (2)$$

The scale factor in equation 2 reflects the variance observed in the estimated cost and the actual cost of a part that has already been manufactured. The part that is being matched against should ideally have a structure similar to the one that is being estimated. That it, the product structure as well as the estimation methodology employed should be similar to the one that the designer is considering for the new part.

Thus, calibration entails the use of a database that records the estimated cost, the actual cost, the costing methodology employed as well as one that employs a search mechanism. The search mechanism should be able to present the user with one or more parts to calibrate against, using a set of pre-defined rules. The user then selects the part that best matches the new design, and the estimated cost is scaled to account for the variance.

Depending on the fidelity of information available, variances may be available on more than one cost pool. Consider three cost pools that are available for each node, Total Cost, Labor cost and Material cost. Further, the “Total Cost” pool is a function of the “Labor” and “Material” pools. The organization may choose to monitor the actual labor hours and material that was utilized, as well as the total cost of the product. Here, the
designer may choose to apply scaling to the individual cost pools versus applying it to the total cost.

### 3.5. Calibration in the WBS

As discussed earlier, the cost structure assumed is a hierarchical structure, similar to a tree. Calibration can be employed at various levels in the tree, that is, at the root level, at an intermediate level or finally at the leaves. This has ramifications, as explained in the following sections. Further, applying corrections at the various levels allows us to categorize the elements in the cost structure.

#### 3.5.1. Calibration States

1. **Not Calibrated** - This is the default state of the element. This implies that neither this element, nor any element above or below this element in that branch of the cost tree structure has been calibrated. The values of the cost pools are only the estimated values. The calibration ratio employed at each level and at each cost pool uses a default value of 1.0.

2. **Parent Calibrated** - This implies that the parent of this element is being calibrated. The parent may not be the immediate parent, and could be higher in the tree hierarchy.

3. **Child Calibrated** - This state implies that one or more children of this element are being calibrated. Again, these may not be the immediate children of the element, and may be lower in the hierarchy of the tree structure.
4. **Calibrated** - This state implies that the user has chosen to apply correction to one or more cost pools at this level. This means that all the elements below this in this branch of the tree would have the Parent Calibrated state, while all the elements above this would have their status set to Child Calibrated.

5. **Invalid Calibration Object** - This state implies that the user chose to calibrate this element but the calibration module detected

### 3.5.2. **Calibration Employed at the Root Level**

The root of the product structure will typically be a final assembly or product. The root is composed of other sub-assemblies, parts or processes. It should be noted that the root has no parents, that is, it represents the final accumulated cost of all its children, in one or more cost pools. The designer could opt for correcting the variance at this level, on one or more cost pools, as long as there are no interdependencies between cost pools as explained in 3.5.5. Applying a scaling factor at the top level is an option when insufficient data is available at the sub-component level.

Applying the scaling factor to one or more cost pools at the root level proves to provide erroneous results, especially if sideways calculations take place at this level. An example of sideways calculation would be “Total Cost” is a function of “Direct Cost”. In this case, “Total Cost” does not represent an accumulation of values from the root's children. Consider Table 3.1, where the root node is made up of two sub-components, A and B. Both A and B have two cost pools, Direct and Total, and Total happens to be a function of the Direct cost pool, in that for A, it is 1.1 times the Direct, while for B it is 1.05
times. The Root node accumulates these values to get a final Direct estimate to be 300 and Total to be 320.

**Table 3.1 - Cost Structure**

<table>
<thead>
<tr>
<th>Node</th>
<th>Cost Pool</th>
<th>Direct ($)</th>
<th>Total ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root</td>
<td></td>
<td>300</td>
<td>320</td>
</tr>
<tr>
<td>A</td>
<td></td>
<td>100</td>
<td>110</td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>200</td>
<td>210</td>
</tr>
</tbody>
</table>

Now, if the scale factor of 0.9 was to be applied to the Direct cost pool at the root level, the Direct cost would now have a calibrated value of 270. But since the Total cost is not a function of Direct at this level, Total would remain 320. The table will look as shown in Table 3.2.

**Table 3.2 - Applying Calibration at the Root Node**

<table>
<thead>
<tr>
<th>Node</th>
<th>Cost Pool</th>
<th>Direct ($)</th>
<th>Total ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root</td>
<td>(x0.9) 270</td>
<td>270</td>
<td>320</td>
</tr>
<tr>
<td>A</td>
<td></td>
<td>100</td>
<td>110</td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>200</td>
<td>210</td>
</tr>
</tbody>
</table>
This problem is solved by pushing down the scale factor to each of the children of the root. This means that the ratio of 0.9 would now be used by A and B, each one then calculating the Total cost. The Direct and Total cost would be accumulated by the Root node, and the resultant table would as shown in Table 3.3.

<table>
<thead>
<tr>
<th>Cost Pool Node</th>
<th>Direct ($)</th>
<th>Total ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root</td>
<td>270</td>
<td>288</td>
</tr>
<tr>
<td>A (x0.9)</td>
<td>90</td>
<td>99</td>
</tr>
<tr>
<td>B (x0.9)</td>
<td>180</td>
<td>189</td>
</tr>
</tbody>
</table>

In this case, the calibration ratio for Total at the root can be calculated. In this case, it happens to be 0.9 since this is simply a summation of values. The calculated ratio reflects the corrected variance that was applied to the accumulated cost as a function of the costing algorithm employed for the estimation.

In view of this example, the Root node would have a calibration status of Calibrated, while every other element in the tree structure would have a status of Parent Calibrated. This is because the correction ratio would be pushed down to all of its children, until the leaf elements are reached. Here it is to be noted that calibration is done at the leaf level, although calibration was employed at a higher level.
3.5.3. Calibration employed at the leaf

As in the case of the Root node, several cost pools may be calculated at the leaf level, and more than one may be calibrated, again, as long as there are no interdependencies between cost pools, as explained in 3.5.5. The leaf represents the highest level of fidelity for cost estimation within the work breakdown structure, and has no sub-components. Correcting for variance at this level means that the ratio will be directly applied to the cost pool, and both the estimated values as well as the calibrated values will be rolled up to the parent level.

The user may apply different calibration ratios to the same cost pool across different leaf elements. This allows for a more fine-tuned approach to calibrating the estimate. This requires the existence of an accurate and up-to-date database that records the costs that were estimated and that were accounted for these elements across various cost pools.

In the case of calibrating the leaf elements, all the elements that have the corrected variance applied to them will have a status of Calibrated. The parent of each of those elements, and so on and so forth till the root will have a status of Child Calibrated, while other elements will have a status of Not Calibrated.

3.5.4. Calibration employed at an intermediate level

An element at an intermediate level has parents above it, as well as sub-components beneath it. Applying correction at this level has the same effect as calibrating the root, that is, all the children will use the ratio, while this element, and every element above it
will merely accumulate estimated as well as calibrated values. Again, this element, as well as its parents could calculate a calibration ratio for each of the cost pools, that is divide the accumulated calibrated value by the accumulated estimated value.

In this case, this element will have a status of Calibrated, while all of its children will have a status of Parent Calibrated, and its parents will have a status of Child Calibrated. Every other element in the cost structure will have a status of Not Calibrated.

### 3.5.5. Interdependencies Between Cost Pools

As discussed earlier, the user may opt to calibrate one or more cost pools in an element. The estimated value of the cost pool is then multiplied by the ratio and the calibrated value is calculated. The parent then accumulates the values of the cost pools, and calculates its calibration ratio. Again, consider the example of a leaf element that has three cost pools – Labor, Material and Total cost. Further, Total cost is a function of Labor and Material, in that Total cost is the sum of Labor and Material.

<table>
<thead>
<tr>
<th>Table 3.4 - Interdependencies between Cost Pools</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
</tr>
<tr>
<td>---------</td>
</tr>
<tr>
<td>Estimated</td>
</tr>
<tr>
<td>Labor (Calibrated)</td>
</tr>
<tr>
<td>Total (Calibrated)</td>
</tr>
<tr>
<td>Labor And Total Cost (Calibrated)</td>
</tr>
</tbody>
</table>
In Table 3.4 - Interdependencies between Cost Pools if Labor is calibrated using a calibration ratio of 0.9, the value of Total cost is 99. Alternatively, the user could choose to calibrate Total cost using a calibration ratio of 0.89; resulting in a cost of 97.9. But calibrating both Labor and Total cost not only proves to be redundant, it also provides erroneous results as shown in the last row. Since Total cost is a function of Labor cost, after calibrating Labor, the value of Total cost has already been adjusted. Applying a correction to this value will clearly be an overcorrection. The user may choose to calibrate the Total cost pool, assuming necessary and sufficient data exists, in which case calibration for the Labor cost pool be switched off. Hence, it is necessary that such inter-dependencies, where the value of a cost pool is driven by another, be detected and, the user be allowed to calibrate only one of the related cost pools at a time.

3.5.6. CONFLICTS BETWEEN ELEMENTS

The cost structure of the estimate, as discussed, is assumed to be a tree structure. This implies two things – the children of a particular node constitute the parent, and the cost of the parent is a function of the accumulated values of the costs of each of its children. Hence, the cost of the parent is dependent on the cost incurred by its children. As discussed earlier, if a node is calibrated, the ratio is pushed down to each of its children, until the leaf nodes are reached, which actually apply the ratio to their respective cost pools. The parent merely accumulates both the estimated and the calibrated costs.

Consider an example with two intermediate nodes, A and B where B, is a child of A. If A was calibrated and a calibration ratio was applied to its Total Cost pool, it would push down its ratios to the end of that particular branch in the tree so that the leaf elements
would apply that ratio to their Total Cost pool. Now, if B also was calibrated and a calibration ratio was applied to its Total Cost pool, it too would push down its ratios, overriding the ratios that A was utilizing. Hence, it is pointless to calibrate A, since all of the ratios would be overridden by the values used by B.

Alternatively, consider a similar example with two intermediate nodes A and B where B is a child of A. If A was calibrated and a calibration ratio was applied to its Total Cost pool, this ratio would be pushed down to all the leaf elements in that branch of the tree which would apply the ratio to their Total Cost pools. Now, if B was calibrated and a calibration ratio was applied to its Labor cost pool, this ratio too would be pushed down so that the leaf elements employ this ratio to their Labor cost pools. Although at first glance it seems that there is no conflict, it could be that there are interdependencies (Section 3.5.5) between the Labor and the Total cost pools at the lower levels. The implementation would then need to detect such dependencies not only that the level in the WBS where calibration was being employed but, at lower levels also. This proves to be a complicated mechanism.

Hence, it is necessary for the implementation to ensure that only one element be calibrated in a particular branch. That is, only one element can have a Calibrated state, while every other element will have a state of Parent Calibrated or Child Calibrated in a particular branch of a tree if one is calibrated.
3.6. **An Algorithmic Approach to Calibration**

Section 3.5 explained a high level approach to calibration as well as some of the details that, if overlooked would provide erroneous results. This section explains an algorithm that will enable its implementation, in pseudo code, while the next chapter will discuss the implementation in Java.

The algorithm consists of two steps, the first step is setting up the cost structure for calibration, the second is executing the costing algorithm to calculate and accumulate both the estimated and the calibrated values for each of the cost pools. The first step includes selecting the element/s (subject to the restrictions described in 3.5.6) and the cost pools that are to be calibrated (subject to restrictions described in 3.5.5). The element/s selected will then have a status of Calibrated, while its children, if any, will have a status of Parent Calibrated and its parents will have a status of Child Calibrated (discussed in section 3.5.1). Further, the ratios selected for each of the cost pools will have to be pushed down to the matching cost pools in each of the calibrated element's children, if any. This routine sets the status of the element to Calibrated, and then populates a list with the names and respective calibration ratio for each of the cost pools selected. It then pushes each of the ratios to the elements children and sets their statuses. This is shown in InitializeCalibration (Figure 3.2).
METHOD InitializeCalibration( )
  FOR every element selected
    SET element.calibration_status = CALIBRATED
  FOR EVERY cost_pool selected
    ListOfCostPoolNameAndRatio(cost_pool).name = cost_pool.name
    ListOfCostPoolNameAndRatio(cost_pool).value = cost_pool.ratio
  END LOOP
  CALL PushCostPoolRatios WITH ListOfCostPoolNameAndRatio
  CALL SetParentCalibrationStatus WITH CHILD_CALIBRATED
  END LOOP
END METHOD

Figure 3.2 - InitializeCalibration

The method PushCostPoolRatios (Figure 3.3) is as described below. This routine merely sets the calibration ratios for the element’s cost pools, and then iterates over the children, if any (denoted by the HasChildren flag) and sets their calibration ratios.

METHOD PushCostPoolRatios (ListOfCostPoolNameAndRatio)
  Call SetCostPoolRatios WITH ListOfCostPoolNameAndRatio
  IF this.HasChildren Then
    FOR EVERY Element in this.ListOfChildren
      SelectedElement = this.ListOfChildren[Element]
      SelectedElement.calibrationStatus = PARENT_CALIBRATED
      CallSelectedElement.PushCostPoolRatios
    END LOOP
  END IF
END METHOD

Figure 3.3 - PushCostPoolRatios

The method SetCostPoolRatios is detailed in Figure 3.4. Here it is important to note that the parent and child may not have the same cost pools. It may be the case where a parent has a cost pool that does not exist in any of its children. Hence, the sub-routine first attempts to see if this element has a cost pool with the same name as one that is being calibrated (ParentCostPool). Then, if this element has any children the algorithm
iterates over all the children to find one that has a cost pool with the same name. If at least one such child is found, it sets the calibration ratio of ParentCostPool to 1.0. This is due to the fact that eventually the calibration ratio will be pushed down to the children. If such a child is not found, which implies that either the element has no children, or that it has a cost pool that does not represent an accumulation of the values of its children it sets the calibration ratio of ParentCostPool to that found in ListOfCostPoolNameAndRatio.

METHOD SetCostPoolRatios (ListOfCostPoolNameAndRatio)
FOR EACH CostPool IN this.ListOfCostPools
    ParentCostPool = ListOfCostPools[CostPool]
    FOR EACH Name IN ListOfCostPoolNameAndRatio
        SelectedName = ListOfCostPoolNameAndRatio[Name].name
        IF ParentCostPool.name = SelectedName THEN
            IF this.HasChildren THEN
                FOR EVERY Child IN this.ListOfChildren
                    FOR EACH ChildCostPool IN child.ListOfCostPools
                        IF child.ListOfCostPools[ChildCostPool] = SelectedName THEN
                            ParentCostPool.ratio = 1.0
                            ChildHasBucket = True
                            BREAK
                        END IF
                    END LOOP
                    IF ChildHasBucket = False THEN
                        ParentCostPool.ratio = ListOfCostPoolNameAndRatio.ratio
                    END IF
                END LOOP
            ELSE
                ParentCostPool.ratio = ListOfCostPoolNameAndRatio.ratio
            END IF
        ELSE
            ParentCostPool.ratio = ListOfCostPoolNameAndRatio.ratio
        END IF
    END LOOP
END METHOD

Figure 3.4 - SetCostPoolRatios
The method `SetParentCalibrationStatus` traces the parents of each of the elements of the selected elements up the tree to the root, and sets the calibration status to child calibrated. This has not been detailed as a consequence of its simplicity.

The methods described (Figure 3.2, Figure 3.3, Figure 3.4) previously ensure that the WBS now reflects the correct status for each element, and that each cost pool that has been selected for calibration now holds the correct ratio. The execute cycle, as described below, goes to each node in the tree and executes the underlying cost algorithm to find the estimated cost.

The algorithms described are very simplistic versions. As the reader can appreciate, this algorithm does not cater for the restrictions discussed in 3.5.2, 3.5.3, 3.5.4, 3.5.5, 3.5.6.
4. CALIBRATION – AN IMPLEMENTATION

4.1. INTRODUCTION

As discussed in earlier sections, cost estimation proves to be a critical activity in the early design phases of any product. A team at Ohio University has developed the Cost Estimator, a tool that allows a manufacturer to model and estimate the cost of any product. The Cost Estimator was developed using Java on Java Development Kit 1.3. This chapter discusses the implementation of calibration, which is a module within the Cost Estimator.

4.2. REQUIREMENTS

Calibration allows for the adjustment of estimates based on the error found when estimating similar elements in the past. Specifically, it allows an organization to utilize information stored in a knowledge base of its experience with similar parts. This information can then be used to “tweak” the estimate to eliminate any errors or inaccuracies in the underlying costing algorithm. This chapter will discuss each of the following in detail.

4.2.1. CALIBRATION IMPLEMENTATION

This is the actual implementation of the calibration algorithm within the cost engine of the Cost Estimator. Keeping in mind that the underlying implementation of the Cost Estimator is in Java, there are several classes that interact to provide the necessary infrastructure for the calibration functionality.
4.2.1.1. Element

The Element class is an encapsulation of any entity that can incur cost, as explained in earlier sections. This class represents the super-class of any entity whose cost can be estimated by the Cost Estimator. It has and provides all the necessary Application Programming Interfaces (APIs) to create and execute the element. In order to model a part, or a process, the developer has to extend this class. Elements are typically organized in the WBS, so that an element may have one parent and one or more children. An Element instance may also hold one or more instances of Attributes (Section 4.2.1.2); Buckets (Section 4.2.1.3) and Methods (Section 4.2.1.4) (Note that this relationship is bi-directional, in that the Attributes, Buckets and Methods know which Element they belong to).

The Element class maintains the calibration state of the instance as explained in Section 3.5.1.

The Element class implements all the routines described in Section 3.6. These routines allow the Element to be calibrated, update the calibration states of itself, its parents and its children, and push down relevant cost ratios to all of its children. In order to prevent conflicts between Cost Pools as discussed in Section 3.5.5, there is an API that must be overridden by the developer if such interdependencies do exist. This is because the Element class has no means of detecting such dependencies as these will be presented in sub-classes of the Element class.
This API is verifyBucketDependencies and has two arguments – the first being the cost pool object \( B \) that needs to be checked for dependencies and the second, a collection of cost pool objects that \( B \) may be dependent on. The developer may need to iterate over the collection of cost pools and check to see if \( B \) is dependent on any one or more cost pools and if yes, return a true, else return a false. This method is called by the calibration implementation when the user attempts to select cost pools to calibrate.

The cost engine, when executed, will recursively execute each of its children (if any), accumulate the cost of each of its children’s cost pools and calculate the cost of its own cost pools. When calibrating any Element in the WBS, this routine is executed twice (Section 4.2.1.1.1). The first time through, the cycle executes in the same manner as if no element is being calibrated. That is, when calculating the cost incurred in each of the cost pools, no calibration ratios are applied. All elements merely calculate and accumulate the estimated costs as dictated by the underlying costing methodology (Section 4.2.1.4). In the second cycle, however, those cost pools that were selected by the user to be calibrated apply the cost ratios to the estimated values, so that now those cost pools hold calibrated values. Other cost pools that have not or could not be calibrated (Section 4.2.1.3.1) use their estimated values, or alternatively apply a calibration ratio of 1.0.

4.2.1.1.1 Double Execute cycle

The implementation employs a double execute mechanism to calculate the estimated and calibrated values of the cost pools for each element. This is to ensure that interdependent cost pool values are calculated correctly. Consider an example where the
root node of the tree has a cost pool named Final Cost that was a function of Labor and Material cost pools, which in turn are accumulated values of their children's values. In this case, the children do not have a Final Cost pool. There are two points that need to be noted:

1. The setValue method in Bucket has a value (integer or double) as its argument.

2. The calibration algorithm needs to be completely transparent to the element developer, in that the element developer does not need to write additional code for the calibration algorithm to execute correctly. In this example the value of Final Cost would be set as `FinalCost.setValue(LaborCost.getValue() + Material.getValue)`.

After the first execute cycle, the values of Labor and Material will reflect the accumulated estimated values of their respective children. The value of Final Cost would be set as a function of the two cost pools. Even if in the first cycle we were to calculate the calibrated values for Labor and Material, there is no way to set the calibrated value of FinalCost without the element developer writing additional code. Hence the need for the second execute cycle, where the calibrated values of Labor and Material would be accumulated and the calibrated value of Final Cost would be set. This problem could be avoided by one of two approaches.

1. The setValue method takes a Bucket object as an argument instead of a value. In this case, the line of code that sets the Final Cost value would look like `FinalCost.setValue(LaborCost + MaterialCost)`. Then, there could be only one
execute cycle which would accumulate both estimated and calibrated values. FinalCost could then query the Labor Cost and Material Cost pool objects for the estimated and calibrated values, set its values and then compute the calculated calibrated ratio by comparing the two values. The reason for not implementing this was that we did not want to break existing Cost Estimator elements that were already in use.

2. The second approach would require a feature that presents itself as a shortcoming of the Java language, and that is operator overloading. If this was available overridden mathematical operators could be used to calculate both estimated and calibrated values if the element was being calibrated.

4.2.1.2. **Attribute**

This class represents the properties of an element. Although this class does not directly participate in the implementation of the calibration algorithm within the cost estimator, it provides a means for the user to search for similar elements as discussed in Section 4.2.2. An element typically will hold many attributes, each performing one of the following functions.

1. The attribute may be a value that is used in a cost equation, such as the outer diameter of a lock nut.

2. It may used as an intermediate variable for a table lookup. For example, a drop down attribute that lists material names may be used to find the unit cost of the material in a table.
3. It may simply serve an informational purpose, such as a part name.

4. It may define other elements in the WBS. For example, the number of holes in a face plate, where drilling is an element defined in the WBS.

The Attribute class is the highest abstraction of a properties object, and the Cost Estimator defines several concrete implementations of the same (Attribute is abstract) to accommodate for several data-types as well as multi-dimensional values. Currently the Cost Estimator implements the following types of attributes.

1. BooleanAttribute holds boolean values.

2. BooleanArrayAttribute holds a one dimensional array of Boolean values.

3. CascadedListAttribute contains a nested list of values that allows a hierarchy of values to be presented.

4. DoubleAttribute holds double precision numbers.

5. DoubleArrayAttribute holds a one dimensional array of double precision values.

6. DoubleTableAttribute holds a two dimensional array of double precision values.

7. IntegerAttribute holds integer values.

8. IntegerArrayAttribute holds a one dimensional array of integer values.

9. ListAttribute presents itself as a drop-down of allowable string values.
10. ListArrayAttribute allows the user to select multiple strings from a drop-down of allowable strings.

11. StringAttribute holds a string value.

12. StringArrayAttribute holds an array of string values.

### 4.2.1.3. Bucket

The Bucket class is an encapsulation of a cost pool. A Bucket holds both estimated and calibrated values (if the Element that holds the Bucket is being calibrated) as well as the calibration ratio. The Bucket class offers appropriate *getters* and *setters* (accessors and modifiers) to get and set the value of the Bucket. These would typically be used when the developer would override the calculate() API in Method (Section 4.2.1.4).

Since this object holds both the estimated as well as calibrated values, when the element is being executed, this object has to be notified as to whether it is to store and return estimated or calibrated values. This is a ramification of the double execute cycle as discussed in Section 4.2.1.1.1, in that the object needs to know whether to set or return its estimated or calibrated values. In the first cycle of the execute method, it will take the value that is being passed in its setter method and set its estimated value. In the second cycle however, it has to scale the value passed into it by the calibration ratio and set its calibrated value. Further, when queried for its value (via a *getter* method) it should return the calibrated value.
The Bucket class is the highest level of abstraction for a cost pool within the Cost Estimator's environment, and has two concrete implementations (Bucket is abstract) -

1. DoubleBucket is used for holding values that have double precision.

2. IntegerBucket is used for holding values that have integer values.

4.2.1.3.1 Invalid Calibration objects

Under a special set of API calls, it is possible that an invalid calibration object may be created. An invalid calibration object is an element that has been chosen to be, but cannot be calibrated due to an implementation of the calculate() API in Method (Section 4.2.1.4).

Recall that when calibrating elements the execute cycle is called twice (Section 4.2.1.1.1). During the first execute cycle, when estimated values are being calculated, each of the cost pools would return their estimated values (via the getters). Alternatively when the setValue method of the cost pool is called the cost pool merely sets its value to the value passed to it. But in the second iteration of the execute cycle, the cost pool multiplies the value passed into it with the calibration ratio to calculate its calibrated value. Further, it is the calibrated value that is returned when the getValue method is called.

Now consider a scenario in which the element has a Total cost pool which a function of the accumulation of its children’s Total Cost pool values, and its Labor and Material cost
pools. In order to calculate the final value of the cost pool the user would do the following

\[
\text{TotalCost.setValue(TotalCost.getValue()} + \text{Labor.getValue()}) \quad (3)
\]

\[
\text{TotalCost.setValue(TotalCost.getValue()} + \text{Material.getValue()}) \quad (4)
\]

In the above equations \text{TotalCost.getValue} will return the accumulated values of its children. Further, the user applies a calibration ratio to the Total cost pool. Now, in the first execute cycle, the two equations will perform just as expected considering only the estimated values of the cost pools are being returned and set. In the second iteration of the execute cycle, again Equation 3 will perform just as expected, just that on the \text{TotalCost setValue} call, it will multiply the value being passed in with the calibration ratio applied and set its value. When Equation 4 is executed, the getter returns the \textit{calibrated value}, and again, the setter multiples this value by the calibration ratio when settings its value. This obviously represents an over-correction, and must be avoided.

Reiterating, the following sequence of steps create an invalid calibration object – having a set after one or more gets after one or more sets after one or more gets. An elaborate mechanism has been installed in the accessor and modifier of the Bucket implementations to detect such an occurrence of API calls, and if so, render the Element that holds that cost pool as an invalid calibration object.
4.2.1.4. Method

A Method class represents an encapsulation of a cost estimation method. Every Element class implementation must have at least one method, and some may have more than one. Each method corresponds to a way to estimate the cost of the element. A method will potentially utilize one or more attributes (Section 4.2.1.2) and cost relationships to calculate and accumulate costs in one or more cost pools (Section 4.2.1.3). An Aggregate method may hold several other elements. Aggregate methods typically accumulate the costs of their children. There are two methods that the developer needs to override. One is initAttributes(), which is an empty implementation in Method. This method can be used to initialize any attributes (Section 4.2.1.2) that are going to be used in the calculation. The other method is calculate(). This method calculates the values of the cost pools at the end of the execution cycle.

4.2.2. Knowledge Base

In the Cost Estimator, the knowledge base of prior estimates is in the form of a CSV (Comma Separated Values) file. This file holds information about one or more types of Elements, as well the calibration ratios for one or more cost pools in each element. The calibration ratio, as explained earlier is the ratio of the actual manufacturing cost of a part or process to the estimated cost of the same. For the search mechanism (Section 4.2.3) to work, the files that hold that estimate must also exist.

The CSV file used for calibration has to have at a minimum 6 columns.

1. Part, the part number of the part whose cost was estimated.
2. **Element Class**, the fully qualified Java class name of the element of the estimate. For example, edu.ohiou.estimator.calibrationExample.Vise.

3. **Name**, the name of the element. This is merely a unique name to identify elements. The calibration module assumes that Part Number (column 1) along with Name forms a unique key. When searching for elements, if one or more rows are found to have the same combination of these two keys, only the first one will be selected, irrespective of data held in other columns.

4. **Method**, a column that lists the method (Section 4.2.1.4) that was applied when the estimate was generated.

5. **Element Path** is the element we may want to calibrate against may not necessarily be the top level element. For example, the user may want to calibrate the Back of one Chair versus the Back of another. This path reflects the position of the element in the WBS tree hierarchy. These have to be the actual class names of the elements.

6. **Estimate File** is the path of the file that holds the actual estimate. This can be an absolute file path, or inside a jar file.

Having these 6 columns makes a valid csv file for calibration, although not a very useful one, considering that such a file does not hold the ratios for calibrating other elements. For calibration, additional columns holding the calibration ratios need to be added. There are two ways in which the columns can be added. One would be to put the ratios themselves in the columns. If this case, the column header should be a “/” followed by
the cost pool’s (Section 4.2.1.3) name. So for example, if the ratios for the cost pool’s with the name Labor was to be in the csv file, the column header would be /Labor. The alternative to this would be to have two columns, one holding the actuals, the other holding the estimated. In this case, the column that holds the actuals would start with a “~” followed by the name, and the column holding the estimated value would have its header as the name. Keeping in mind the previous example, the column that holds the actuals would be ~Labor, followed by another column that holds the estimated column, with the header Labor. In case, both the “/” and the “~” are found, then the “/” will be given preference, and any values in the other columns for that cost pool’s name will be ignored. Again, note that the column headers should have the name of the cost pool. The cell may hold an empty value, in which case, the user will NOT be allowed to calibrate that cost pool. Also, the element may be holding other cost pools that do not appear in the csv file. These cost pools too, will not be allowed to be calibrated.

The csv file has to have columns in the same order as listed (columns 1 through 6). These cells CANNOT be empty. The columns reflecting the ratios may be in any order, as long as the column headers hold the name of the cost pools, and they may contain empty cells. The file may hold calibration information of several classes of elements, as the Element Class (column 2) will allow the calibration module to find the appropriate rows.

4.2.3. SEARCH FOR SIMILAR ELEMENTS

As discussed earlier, the CSV file may have information regarding the calibration properties of several elements, as well as several types of elements. The calibration
module provides a way for the user to discard elements from the csv file that do not match certain characteristics. This is done by way of filtering. Filters will have a type, a value, a level, and appropriately, a range. The type is one of Class, Method or Attribute. There are three levels that a filter can use, one of Must Match, Critical or Desired.

Critical or Desired differ in the weightage that they allot to matching elements. For example, if an attribute filter (e.g. Height) was set to have a value of 0.6in ± 0.2 (this is the range), and its level was set Critical, any other element found with height of 0.6 would be given a weightage of 10. Any element found with a height of 0.4in or 0.8in would be given a weightage of 1 and elements having heights within the range will be given proportional weightages. This is shown in Figure 4.1. Now, if the filter level was set to Desired, the maximum weightage allotted would be 5. Now, if the level was set to Must Match, and the value of the matching element was found to be within the range, then it would appear on the list of available elements; else it would be dropped off the list. Must Match filters are a means to eliminate elements from the list that do not match a particular criteria. They do not allot any weightage to the element.
The filters provided by the calibration module are of the following types

1. **Class Filter** - Considering that an Element is actually an executable Java class, the class of the Element can be used as a very high level filter to eliminate elements of other classes. It uses the Element Class column in the csv file for restricting the table. This filter will have its value set to the class of the element which cannot be changed; this filter can be dropped from the list of applied filters. Further, this will have a level of Must Match, which again cannot be changed. Hence, if this filter is applied, only elements that hold the same class as the element being calibrated in the csv file will appear on the list.

2. **Method Filter** - Much like the Class Filter, this filter uses the current method of the element being calibrated. Again, this is filter whose level is set to Must...
Match, and cannot be changed. This too will use the Method column in the csv file for restriction.

3. Attribute Filter - This filter uses Attributes for filtering. Each filter instance can be set to any of the three levels, and each will have a range. Attribute filtering is again, of two types, those that use numeric values, and can have a range, and those which uses attributes that uses exact values, such as those which use Strings and Booleans. In case of those Attributes that use numeric values, the user can specify the range that is desirable. For example, the user may specify that the matching element should have its Height attribute within $50 \pm 5$. This range can be specified either in absolute or percentage values relative to the current value of the attribute. All elements that have a height of 50 will be allotted a weightage appropriate to the level of the attribute. Only one filter per attribute is allowed.

The search module, by default, constructs two filters, the Class Filter and the Method Filter. The user may add, remove or edit these or other filters.

After the user chooses a calibration csv file, the module will filter the available elements in the file based on existing filters. The calibration module then adds or eliminates elements, or update their weightages as filters are added, removed or edited.

The user must then select one element to calibrate against from the list of potential matches. It may be the case that no elements are available that match the requirements of the user as specified in the filters. This indicates that some of the filters may have to
be relaxed or dropped. The calibration module then displays the cost pools for which calibration information (ratios or estimated/calibrated values) are available in the CSV file. The user then must select one or more of the cost pools requesting that the ratios that are associated with those cost pools in the CSV file be used for calibration ratios.
5. CALIBRATING COST ESTIMATES

As discussed in Section 1.3, Ohio University, as part of FIPER, has developed a powerful cost estimation tool named “Cost Estimator”. This tool is a hierarchical cost estimation tool in that it models the WBS of the product for estimation purposes. This section discusses the Cost Estimator as well as demonstrates how a simple assembly such as a vise can be calibrated.

5.1. USER INTERFACE DESIGN AND FUNCTIONALITY OF THE COST ESTIMATOR

The Cost Estimator has been designed as a thick-client, or a stand-alone application using Java and the Java Foundation Classes (Swing). The user interface comprises primarily of four sections.

1. The WBS panel that displays the hierarchical cost structure of the element being estimated.

2. The Method (Section 4.2.1.4) panel that displays the costing algorithm being used for estimation. This panel also displays the attributes (Section 4.2.1.2) that are used in the cost methodology.

3. The Summary panel that displays the costs accumulated within each cost pool (Section 4.2.1.3) for the selected element, as well the costs in each of the cost pools held by its immediate children.

4. The menu and tool bar that exposes several pieces of functionality within the Cost Estimator.
Figure 5.1 - Cost Estimator Layout
An examination of Figure 5.1 reveals the following. The WBS panel exposes the hierarchical structure of the top element, displaying all of its sub-elements and their children. Next to each element is displayed the estimate of what an element contributes to the cost in any one of its cost pools (This can be set by the user).

The Method panel displays the attributes that the current costing algorithm uses for estimating cost. The element in the example has only one method available, but if several options were available then these would appear as tabs at the top left hand corner of the panel. By changing the values of the attributes the user can easily see changes in the cost of an item, thus allowing the user to perform a what-if analysis while considering alternative design decisions.

The Summary panel displays the various cost pools held by the selected element as well as those held by its immediate children, along with their values. The pie chart displays the contribution of each sub-element to the total for a particular cost pool (Again, this can set by the user).

5.2. CALIBRATING AN ELEMENT

Consider a firm that manufactures vises. When designing and cost estimating a new vise, calibration could be used to eliminate bias in the costing methodology employed. The data for calibration can be gathered from previously manufactured vises by comparing the actual manufacturing costs with that of their respective estimated costs.

To form an estimate for a new vise, the WBS of the vise needs to be defined. This could be done within the vise template (a Java class that extends the Element class defined in
the Cost Estimator library). This template will contain the definitions of all the attributes that the vise will have along with their default values, the various cost pools that will be used to accumulate the cost and the various costing methodologies (methods) that can be used for estimating the cost of the vise. This template is a Java class and hence can be used within many estimates as a starting point. Alternatively, the user could use an existing estimate file (*.est file) as a starting point.

In order to calibrate an element, the user right-clicks on the element in the WBS panel and selects Calibration → Set Up Calibration as shown in Figure 5.2. This brings up a window (Figure 5.3) that allows the user to specify the CSV file (Section 4.2.2) that is to be used; add/edit/delete filters (Section 4.2.3) that will be applied when searching for a matching part and finally selecting a part and the appropriate cost pool(s) to calibrate against.

![Figure 5.2 - Setting up Calibration](image)

Clicking on the Browse button allows the user to navigate the file system to locate the CSV file. The module will then parse the contents of the file, apply the filters as specified
by the user and display a result set of available elements and associated cost pools in
the lower panel.

**Figure 5.3 - Calibration**

The user can add, delete or edit filters using the buttons provided on toolbar. Clicking on
the Add filter button brings up a dialog that allows the user to specify the type of filter
(Section 4.2.3), and the appropriate level of weightage (Must Match, Critical and
Desired). Depending on the type of filter the user specifies the user may be required to
fill in the desired range for the filter, for example, if the filter type specified is of type Attribute, and the attribute selected if of type DoubleAttribute (Section 4.2.1.2).

![Figure 5.4 - Adding a Filter](image)

**Figure 5.4 - Adding a Filter**

In Figure 5.4 the thread diameter of the matching MainViseScrew must be within 10% of the diameter of the MainViseScrew being calibrated.

The elements that are found matching the filtering criteria are displayed in the lower panel along with the cost pools for which data is available. The user can select any one of the elements by highlighting the appropriate row, selecting the cost pools that are to be calibrated and clicking the Apply Selection button. Note that the first row represents the element that is being calibrated and hence cannot be selected. Further, on selecting an element in the list, one or more Buckets may gray out. This is because the list in the Buckets panel reflects the cost pools held by the element being calibrated. Since there might not be calibration information pertaining to all cost pools of these in the CSV files, only those have the necessary columns filled out in the CSV file will be available for the user to select. Also, sometimes on selecting one cost pool in the Buckets panel causes other cost pools to gray out. This happens when these cost pools have
interdependencies (Section 3.5.5) that the calibration module can detect. To ensure the integrity of the calibration, the tool prevents the user from selecting two or more cost pools for calibration that have interdependencies between them.

Once the element has been calibrated the Cost Estimator displays appropriate icons in the WBS panel to indicate the status of each of the elements. The Cost Estimator uses four icons to indicate elements which have been calibrated, those that have their parents calibrated, those that have one or more children calibrated or those that could not be calibrated.

1. Calibrated (/calibrated) signifies that the element is being calibrated. All elements beneath this node will have a state of Parent Calibrated while all elements above this node will have a state of Child Calibrated.

2. Parent Calibrated (calibrated) signifies that one of the parents of this element are being calibrated. This parent need not be the immediate parent of the element but could be higher in the tree.

3. Child Calibrated (calibrated) signifies that one or more children of this element are being calibrated. This child need not be one of the immediate children but could be lower in the tree.

4. Cannot Calibrate (calibrated) signifies that the element is an invalid calibration object (Section 4.2.1.3.1).

The WBS panel displays the icons as shown in Figure 5.5.
Figure 5.5 - Calibration Status in the WBS Panel

Hence, in Figure 5.5 MainViseScrew is being calibrated and has a status of Calibrated. As discussed, the calibration ratio is then pushed down to its children namely Material, Turning, Threading and Drilling which have a calibration status of Parent Calibrated. Both MainViseScrew and Vise in this case accumulate estimated as well as calibrated values and Vise has a status of Child Calibrated.
6. FUTURE WORK AND CONCLUSIONS

6.1. FUTURE WORK

At the time of writing this text, there are some enhancements that could be made to the calibration module within the cost estimator -

1. Integration with Risk Analysis / Trade Study modules - The cost estimator offers modules for Risk Analysis and Trade Studies. Risk Analysis, performed using Monte Carlo simulations, requires a set of inputs for one or more attributes, and using a user-defined run length, produces minimum, maximum and average cost estimates for the part or product. Trade Studies allows a user to perform a what-if analysis on a cost estimate. The user can change values of attributes, see the change in the cost estimate and if necessary capture the state of the estimate at that point as a Trade Study. The cost estimator allows comparison of different cost estimates in a graphical user interface. These modules present only the estimated values to the user. It would be desirable if the outputs of these modules could be calibrated.

2. Calibration back-end - Calibration currently uses a CSV file (Section 4.2.2). This file has to be generated by hand, and hence proves to be a time-consuming and error-prone task and is at risk of being incorrect and inconsistent with real data. Further, the file format being comma delimited text is prone to corrupt easily. It would be desirable to have a robust format such as XML, as well as having a
means to generate the file automatically from an existing repository of saved estimates.

6.2. Conclusions

The calibration module was designed to accomplish the following goals -

1. It should allow the user to calibrate the existing estimate. The implementation should be integrated within the existing cost estimator architecture and its execution within the cost engine of the cost estimator should be transparent to the user.

2. It should allow for calibration at any and all levels of the WBS, as well as across cost pools as long as any conflicts are detected and the user notified.

3. There should be an easy and intuitive way for the user to search for existing parts that are similar to the part being estimated.

The implementation discussed in this text has satisfactorily achieved all of the above mentioned goals.
REFERENCES


