12-2009

Design for Assembly Line Performance: The Link Between DFA Metrics and Assembly Line Performance Metrics

Krishna Kamath

Rochester Institute of Technology

DESIGN FOR ASSEMBLY LINE PERFORMANCE: THE LINK BETWEEN DFA METRICS AND ASSEMBLY LINE PERFORMANCE METRICS

A Thesis

Submitted in Partial Fulfillment of the
Master of Science in Industrial Engineering

in the

Department of Industrial and Systems Engineering
Kate Gleason College of Engineering

by
Krishna Kamath
December, 2009

DEPARTMENT OF INDUSTRIAL AND SYSTEMS ENGINEERING
KATE GLEASON COLLEGE OF ENGINEERING
ROCHESTER INSTITUTE OF TECHNOLOGY
ROCHESTER, NEW YORK

CERTIFICATE OF APPROVAL

M.S. DEGREE THESIS

The M.S. Degree Thesis of Krishna Kamath
has been examined and approved by the
Thesis Committee as satisfactory for the
thesis requirement for the
Master of Science degree

Approved by:

_____

Dr. Marcos Esterman, Thesis Advisor

_____

Dr. Andres Carrano

Dedicated to my love, Jenn Chan

Acknowledgement

I would like to thank everyone who has supported me to achieve and excel in my pursuit of research.

My special acknowledgements to Dr. Marcos Esterman who, like a father guides his children, has assisted me in completing this research. Dr. Marcos has always helped me in making a decision. He always said, “If I were you I would do this.” That’s what has made a huge impact in my thought process during my time at RIT. I always think of Dr. Marcos as a guide, a mentor. His mentorship has definitely made a tremendous improvement in my life.

Thank you so much Dr. Marcos for being there all the time when I made mistakes. Dr. Marcos made this research and my education a really good experience, one to remember and feel good about. I can definitely write a lot about his contributions but I would like to keep it short and say, “Thank you for everything.”

Also, special thanks to Dr. Andres Carrano, his innovative ideas and manufacturing support made this research successful.

Also, a note for Marilyn Houck, the one who fixed all my paperwork; Jeffery Robble, just a normal programmer who knows little bit of C++ and completed all the programs for this research; and all my friends in the lab who have been there and given me hope that I can do it.

Also, to my family and Jenn Chan who are the love and light of my life. Thank you so much Jenny Chan, we did it!

Abstract

Design for Assembly (DFA) is a tool that has been in use for almost 40 years. While it has been a useful design tool, it is not explicitly linked to actual manufacturing line performance. The motivation for this research came from the desire to link DFA directly to line balance and cycle time performance. The natural question that arose was whether these issues could be considered at the design stage by using the metrics that are derived from a DFA analysis. It is known that the time required to assemble a product can be estimated from both a DFA analysis and from a manufacturing analysis. This work links these two analysis methods so that the manufacturing parameters can be estimated and used to guide the design of a product.

The methodology developed begins with a DFA analysis of the product. The times and operations from the DFA analysis are used to determine the minimum number of workstations to balance the line while maintaining the production rate (takt time) and precedence constraints. The precedence constraints are systematically relaxed in order to generate measures on a component-by- component basis as to the impact it could have on reducing cycle time and improving Line Balancing performance. These measures, coupled with an understanding of precedence types, are used to identify design improvements to a product. To illustrate how product designer can consider assembly line performance issues during the design stage of the product, the methodology has been applied to an ABS brake assembly.

Table of Contents

"Limitations live only in our minds. But if we use our imaginations, our possibilities become limitless."

List of Figures

List of Tables

1. Introduction

1.1 Background

In the past, design and manufacturing were often treated as two independent areas of operation within a company. Today, the practice of concurrent engineering is more prevalent, and the wall between designers and the factory floor is less common. Industry has realized that in order to improve productivity they need to bridge the gap between the two areas. This thinking has given rise to methodologies like Design for Assembly and Design for Manufacturing, which are two of the earliest known tools in the Design for X toolset.

The Design for Assembly tool popularly known as DFA is an approach to designing products with the goal of increasing the ease of assembly of a given product. The process of assembly is faster and cost efficient when the parts are easier to put together. This in turn adds value for the customer and results in higher profit for the manufacturer.

DFA tools breakdown the assembly into discrete operations where the parts, the handling, the insertion, and the processing activities are evaluated according to stability, directionality, manipulability and other difficulties (Redford and Chal, 1994). Vincent and Filippo (2003) define DFA as "*a process for improving product design for easy and low-cost assembly, focusing on functionality and on assemblability concurrently*."

There are two basic approaches to DFA. One entails the application of simple heuristics and guidelines. They serve as a checklist that the designer can go through as they are designing their product and as they review their product concepts. The second is a detailed analysis that looks at assembly sequences and operations to assess their effectiveness and efficiency.

Chan and Salustri (2003) summarized a set of guidelines that are shown in Table 1. According to them there is an initial idea for a particular design. The next step is to use a set of guidelines that are summarized in Table 1 in order to decide on the factors that are applicable.

Table 1: Basic DFA Guidelines (Chan and Salustri, 2003)

Basic DFA Guidelines:
• Minimize part count by incorporating multiple functions into single parts. • Modularize multiple parts into single subassemblies. • Assemble in open space, not in confined spaces; never bury important components. • Make parts such that it is easy to identify how they should be oriented for insertion. • Prefer self-locating parts. • Standardize to reduce part variety. • Maximize part symmetry. • Design in geometric or weight polar properties if nonsymmetrical. • Eliminate tangly parts. • Color code parts that are different but shaped similarly. • Prevent nesting of parts; prefer stacked assemblies. • Provide orienting features on nonsymmetries. • Design the mating features for easy insertion. • Provide alignment features. • Insert new parts into an assembly from above. • Eliminate re-orientation of both parts and assemblies. • Eliminate fasteners. • Place fasteners away from obstructions; design in fastener access. • Deep channels should be sufficiently wide to provide access to fastening tools; eliminate channels if possible. • Provide flats for uniform fastening and fastening ease • Ensure sufficient space between fasteners and other features; for a fastening tool refer to easily handled parts.

The advantages of DFA include fewer parts and simplified assembly processes. Fewer parts in the assembly leads to lower material costs, lower work in process, less purchasing, simplified logistics, fewer parts to design and fewer subcontractors. Also, simplified assembly leads to benefits like higher quality, shorter assembly time, less cost and more flexibility to assemble. The advantages are summarized in Figure 1.

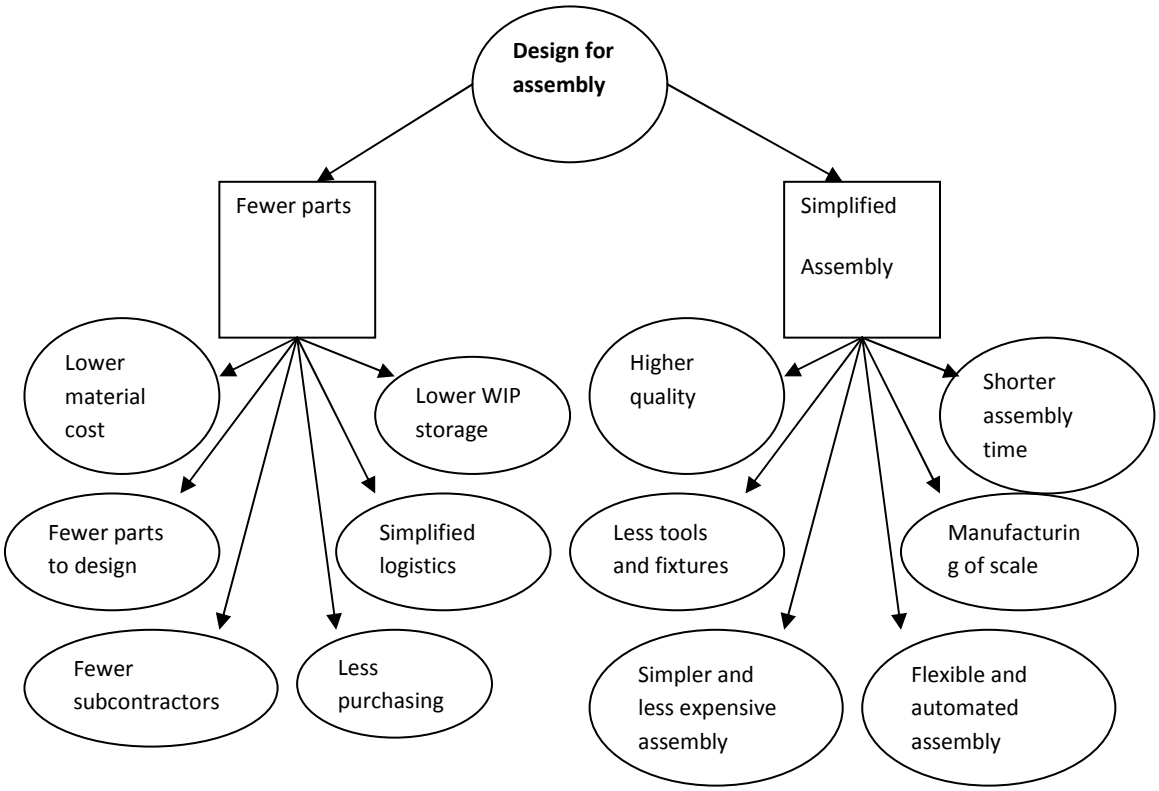


Figure 1: Advantages of DFA

1.2 Motivation

Researchers are constantly trying to generate new ideas to bridge the gap between design and manufacturing. Early works include Pahl and Bietz (1984) and Bralla (1986). The field has grown rapidly and is broadly known as Design for X, or DFX, where the X represents distinct life-cycle considerations, such as environment, supply-chain and reliability. Though DFA was one of the earliest tools developed, researchers continually work to improve the tool (Boothroyd and Yoosufani, 1983, Stone et al., 2003).

This research is focused on exploring the link between DFA metrics and actual assembly line performance. The inspiration for this research came from the observation that DFA does not explicitly consider the issues of Line Balancing and cycle time. The natural question arose as to whether these issues could be considered at the design stage by utilizing the metrics that were derived from a DFA analysis. From the literature review in the next section we observe that there

is relatively little work that has been done on this topic. However, it is known that the time required to assemble a product can be estimated from the DFA analysis. The time required to manufacture a product can also be estimated from a manufacturing analysis. It would be interesting to see if the manufacturing parameters could be estimated from the DFA analysis. In turn, this would allow assembly line performance issues to be considered during the design stage of the product.

Current DFA techniques are widely used to reduce part counts and to simplify the product which results in the reduction of total assembly time. This total assembly time does not help the production operations to calculate the cycle time needed to manufacture the product. It would be valuable if we could calculate manufacturing line performance parameters, such as the total cycle time or a measure of Line Balancing performance, from parameters derived from the DFA analysis.

This leads to the question *"What is the link between DFA metrics and assembly line performance metrics?"*

Figure 2 summarizes the idea that given a design we can carry out various DFX methodologies to optimize this design. In this case, an assembly fishbone is first generated. This assembly fish-bone diagram is a tool to support the DFA tool to provide a sequence of operations and a representation of critical operations in a product's assembly process. This information is used to perform a DFA analysis and to estimate the time required for each operation and the total assembly time. The work elements and the cost required to assemble the product are also estimated. These calculations are based on previous experience and times obtained from the standard work charts, which are estimates of the actual performance. After calculating these parameters, the design of the product is modified until ultimately, the manufacturing stage begins. Similarly, starting from actual observations and previous data and using traditional manufacturing analysis, one can calculate work in process, cycle time and throughput. The assembly line is balanced to eliminate bottlenecks. Logic would dictate that there should be a link between Total assembly time (TAT) and cycle time. Thus, to make a connection we first have to research the literature on DFA analysis and manufacturing line analysis.

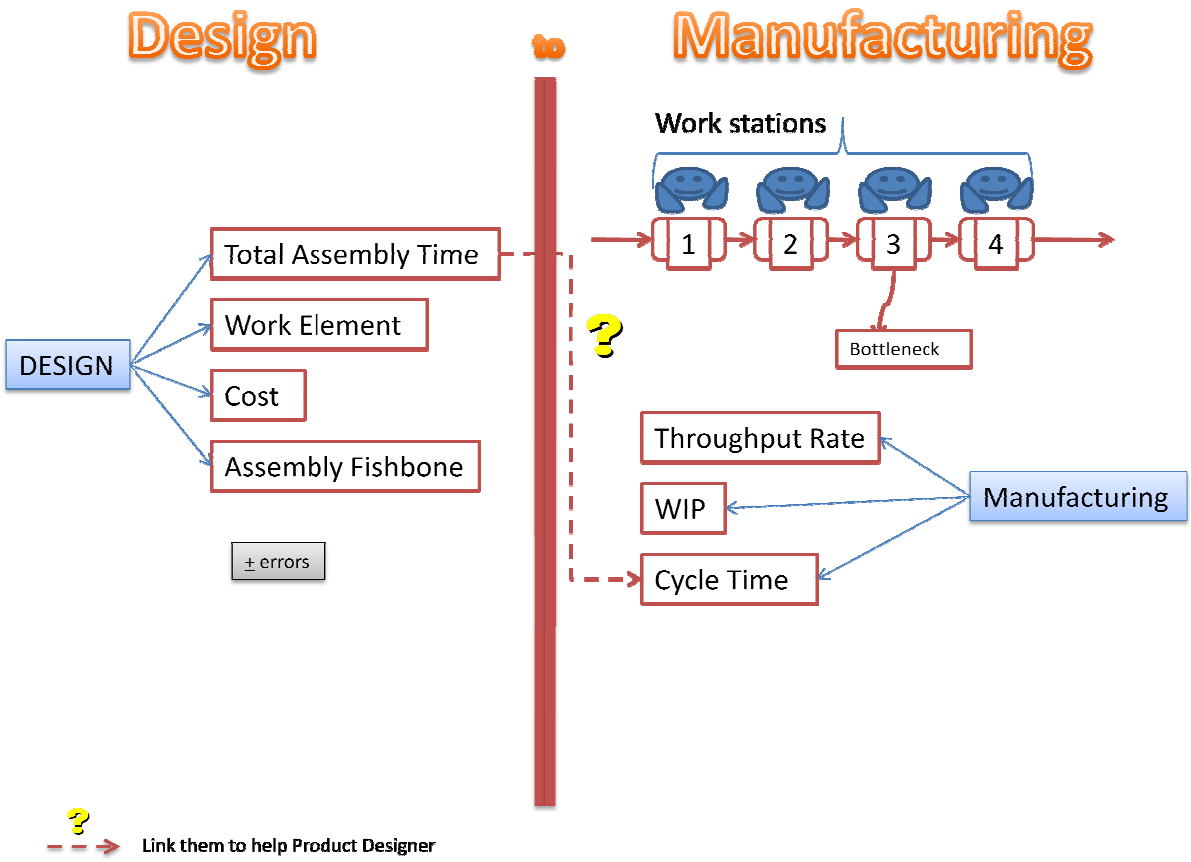


Figure 2: Design to Manufacturing

2. Literature Review

The literature in the area of DFA is vast but most of the work is focused on either modifying or improving the existing DFA methods. This section provides a comprehensive overview of the different DFA methods and their advantages and limitations. A section on the research related to DFA and product development has also been included. This has been an area of significant research and most of the literature is in this field of DFA. The following section introduces the related research area on the linking of DFA to manufacturing analysis. Finally, manufacturing analysis was included to provide a brief review on Line Balancing and the manufacturing cycle time analysis, throughput and work in process parameters since these are key concepts that will be leveraged in this work.

2.1. DFA Methods

The development of DFA started in the early 1960's to help designers consider the assembly problems at the design stage of the product (Boothroyd et al., 2002). During the 1980 to the 1990 period there were many variations proposed to the then existing DFA methodologies, namely, the Lucas method, the Westinghouse method and several others which were based on the original DFA method (Boothroyd et al., 2002).

2.1.1 Overview of the Existing Methods

Here the four major DFA methods are described. This includes a summary of the process steps to execute the methodology, as well as the advantages and disadvantages of each respective methodology. A preliminary DFA analysis was carried out using each of the following method to compare and distinguish their characteristics. This analysis is included in Appendix A. The four different methods that will be reviewed below are Boothroyd and Dewhurst, Boothroyd and Dewhurst DFMA software, Hitachi Assembly Evaluation Method and Westinghouse.

2.1.1.1 Boothroyd and Dewhurst

Boothroyd and Dewhurst (1983) have developed two forms of DFA. The first is the DFA handbook which contains charts and worksheets (Boothroyd and Dewhurst, 1983). The second codifies this knowledge into a commercial software package (Boothroyd and Dewhurst, 2007). The two approaches use the same calculation methodology to derive the DFA metrics.

The Boothroyd and Dewhurst DFA technique employs a DFA handbook, which gives equations and the extensive data necessary to estimate manufacturing and assembly cost during product design. This method is based on two principles: the application of criteria to each part to determine if it should be separate from all other parts and the estimation of the handling and assembly costs for each part using the appropriate assembly process.

It relies on an existing design which is iteratively evaluated and improved. The analysis is generally performed using some kind of worksheet. Tables and charts are used to estimate the part handling and part insertion time. These tables are based on a two-digit code that is in turn based on a part's size, weight, and geometric characteristics. The Table 2 shows a part of the manual handling estimation table. This is used to derive the two digit code used for the

Boothroyd and Dewhurst DFA analysis. The code is determined by the row number and the corresponding column number. The first digit represents the row number and second digit is the column number. As shown in Table 2 a code of 10 means that the assembly time will be 4 seconds. Also, the code 10 represents that there is no resistance to insertion of that part but the vision is obstructed during manual assembly. Similar method is used to generate the two digit codes and corresponding insertion times are used in the Boothroyd and Dewhurst DFMA system.

Table 2: Boothroyd and Dewhurst estimation table (Boothroyd and Knight, 2002)

				Manual Insertion - Estimated Times (seconds)							
				after assembly no holding down required to maintain orientation and location				holding down required during subsequent processes to maintain orientation or location			
				easy to align and position during assembly		not easy to align or position during assembly		easy to align and position during assembly		not easy to align or position during assembly	
				no resistance to insertion	resistance to insertion	no resistance to insertion	resistance to insertion	no resistance to insertion	resistance to insertion	no resistance to insertion	resistance to insertion
				0	1	2	3	6	7	8	9
addition of any part where neither the part itself nor any other part is finally secured immediately	part and associated tool (including hands) can easily reach the desired location		0	1.5	2.5	5.5	3.5	5.5	6.5	6.5	7.5
	part and associated tool (including hands) cannot easily reach the desired location	due to obstructed access or restricted vision	1	4	5	5	6	8	9	9	10
		due to obstructed access and restricted vision	2	5.5	6.5	6.5	7.5	9.5	10.5	10.5	11.5

The steps required for this DFA analysis are summarized as follows:

Step 1: Determine the number of items of each part in the assembly.
Step 2: Determine the tool acquire time, if any.
Step 3: Determine the handling code and estimate the handling time of each item using the tables from the DFA handbook.
Step 4: Determine the insertion code and insertion time using the table from the DFA handbook.

Step 5: Calculate the total operation time which is the sum of handling and insertion times multiplied by the number of items plus the tool acquisition time if necessary.
Step 6: Establish the theoretical minimum number of parts.
Step 7: Repeat steps 1-6 for each part.
Step 8: Calculate the DFA index and analyze the data for part elimination and redesign of specific parts.

The Boothroyd and Dewhurst DFA worksheet also includes non-assembly operations. For example, extra time is allocated for each time the assembly is re-oriented. Next, the parts are analyzed to determine whether they are really necessary in the assembly.

An important benefit of the Boothroyd & Dewhurst DFMA system is that the focus on assessing whether a part is really necessary will lead to a reduction in part count. Reducing part count not only saves assembly and manufacturing cost but also can save labor, inventory, floor space, documentation and administration. The major drawback of this method is that it considers assembly in ideal conditions. Also, the operator is assumed to work at standard efficiency. Although tables of data are available, the most accurate numbers are compiled through time studies in particular factories. However, even with these limitations, the method is very effective at providing deign improvements that help improve assembly efficiency.

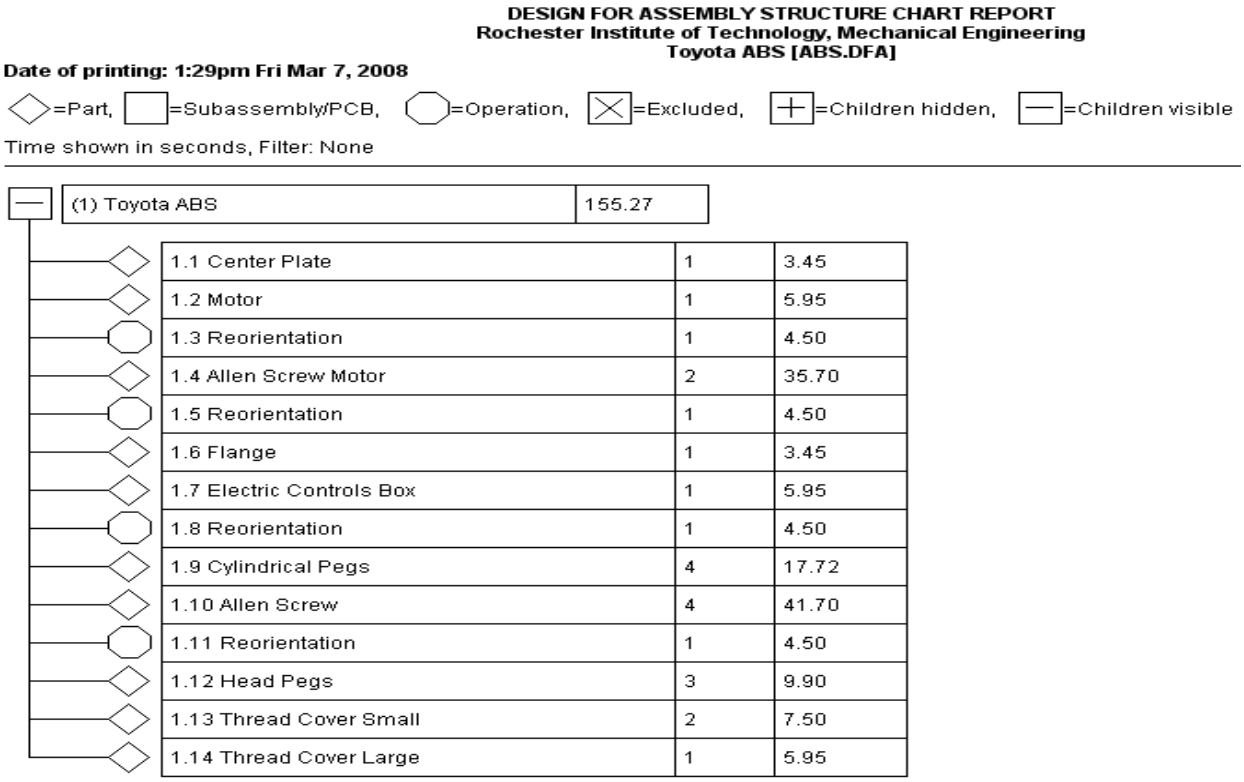
2.1.1.2 Boothroyd and Dewhurst DFMA Software Method

Many DFA Excel sheets have been developed to help the product designers with the process of Design for Assembly. DFA software packages are popular due to their ease of use, the reduction in time required for DFA analysis, and their ability to integrate with commercial CAD packages, such as Pro/ENGINEER, CATIA, Unigraphics, I-DEAS, AutoCAD, Solid Works, and many others.

Boothroyd and Dewhurst Inc. (Boothroyd and Dewhurst, 2007) have developed comprehensive software for DFA analysis. It has an easy to use graphical user interface. The user makes a flowchart which represents the assembly or disassembly process of a particular product. The software then analyzes the complexity of the assembly and differentiates between parts which are required for assembly and parts which can be eliminated or considered for

redesign. Figure 3 below, which was developed using the DFMA software, shows a tree structure chart for an ABS brake.

According to the DFMA website the software is a combination of two complementary tools: Design for Assembly (DFA) and Design for Manufacture (DFM) (Boothroyd and Dewhurst, 2007). DFA software is used to aid the designer in reducing the complexity of a product by identifying opportunities to consolidate parts. The major advantage of this software is to compare the costs of the new design versus the original design at the design stage. Since, the product is still in the development stage it assists in reducing the costs. This software does not make use of CAD models instead it relies on an assembly structure tree as shown in the figure below:-



DESIGN FOR ASSEMBLY STRUCTURE CHART REPORT
Rochester Institute of Technology, Mechanical Engineering
Toyota ABS [ABS.DFA]

Date of printing: 1:29pm Fri Mar 7, 2008

=Part, =Subassembly/PCB, =Operation, =Excluded, =Children hidden, =Children visible

Time shown in seconds, Filter: None

(1) Toyota ABS		155.27
1.1 Center Plate	1	3.45
1.2 Motor	1	5.95
1.3 Reorientation	1	4.50
1.4 Allen Screw Motor	2	35.70
1.5 Reorientation	1	4.50
1.6 Flange	1	3.45
1.7 Electric Controls Box	1	5.95
1.8 Reorientation	1	4.50
1.9 Cylindrical Pegs	4	17.72
1.10 Allen Screw	4	41.70
1.11 Reorientation	1	4.50
1.12 Head Pegs	3	9.90
1.13 Thread Cover Small	2	7.50
1.14 Thread Cover Large	1	5.95

Figure 3: Tree Diagram (Boothroyd and Dewhurst, 2007)

Benefits of using the Boothroyd Dewhurst DFMA software include Supply chain cost management, Product simplification and improved quality, improved communication between design, manufacturing, purchasing, and management and reduced manufacturing costs (Boothroyd and Dewhurst, 2007). The software heavily relies on computing power. The

investment cost to purchase the software and train engineers to learn this software is relatively high. It takes time to understand the software and implement it to optimize your design. The software cannot be customized for use with specific applications like electronic manufacturing.

2.1.1.3 Hitachi Assembly Evaluation Method (AEM)

Ohashi et al. (2002) state that the Hitachi AEM is the first method for assembly-producibility evaluation and that it has been widely used to achieve great cost reductions. Using this method, in the early design stage product design quality is analyzed quantitatively and the weaknesses in the design's assembly producibility are highlighted. In addition, the effects of design improvements are confirmed with respect to assembly cost. Through these activities, design improvements are realized.

This method is based on the principle of "one motion for one part." For more complicated motions, a point-loss standard is used and the ease of assembly of the whole product is evaluated by subtracting points lost. The method was originally developed in order to rate assemblies for ease of automatic assembly. The evaluation process is as follows:

Step 1: Express assembly operations for the parts that compose the product using a combination of elemental operations.
Step 2: Sum up the penalty scores of the elemental operations for the part and modify the sum to account for the complexity of the overall assembly operation.
Step 3: Calculate the product AEM scores as the average value of the part
Step 4: Estimate the assembly time and cost using the product's AEM score and number of parts.

Assembly time (AT) is measured in T-downs. One T-down is the time taken for one downward movement with a part. For the motor it is 1, as there is one downward movement, and the motor is fitted by snap fit. This method helps in an accurate understanding and comparison of assembly time and cost. Also, the AEM score is not closely related to the estimated part attachment operation cost.

2.1.1.4 Westinghouse DFA

This DFA method has been widely used. It is one of the most popular methods as it considers many factors and their interaction while analyzing the time required for assembly. The simplified Westinghouse Method (Hinckley and Kmenta, 2001) involves the following steps:

Step 1: *Feature Identification*
Identify each part in the assembly and document the functions of each part.
Step 2: *Product Disassembly*
Disassemble the product and carefully record the sequence of disassembly.
Step 3: *Function Identification*
Identify the parts that provide the functions detailed in Step 1. Explain how each function is achieved, using small sketches when needed to provide additional clarity.
Step 4: *Assembly Fishbone Diagram*
Create a fishbone diagram to provide a visual representation of assembly subassembly sequences.
Step 5: *Assembly Time Calculations*
Determine the assembly times for each subassembly sequence and for the final assembly sequence.
Step 6: *Pareto Analysis*
Produce a Pareto chart of assembly times for all operations and the cumulative percentage of assembly time.
Step7: *Design Recommendations*
Identify the areas of redesign, focusing on parts integration.

For an example in Appendix A, Table 18, we can see that for the first part of the ABS assembly, i.e., the motor the calculations are done as follows. First the time penalties for all the time factors are determined. The motor end-to-end orientation takes a time of 0.8 seconds as the motor has to be aligned with the center plate. Similarly, the other time factors like rotational alignment, part size, part thickness, etc, are associated with the motor. Next, the time for each operation is calculated which is the sum of all the time factors. For the motor the total time for operation is 6.6 seconds. Then if there is more than one part the repetition time is multiplied by the number of parts to get the repetition time. Here as there is only one motor the repetition time is the same as the time for operation, i.e., 6.6 seconds. Next the decision is taken whether to insert the part or eliminate it from the assembly. Because the motor is a critical part in the ABS brake assembly, we decide to insert the part.

The Westinghouse DFA is particularly useful to help reduce part count and to reduce time to assemble, thereby reducing assembly costs. In addition, it improves design features which make it easier to grasp, move, orient and insert parts. The reduction of the number of parts in an assembly has the added benefit of generally reducing the total cost of parts in the assembly.

2.1.1.5 Lucas DFA Method

The development of the Lucas DFA was done by the Lucas Corporation during 1980 in United Kingdom (Chiang, 2004). In comparison to the Boothroyd-Dewhurst method, the Lucas DFA uses a point system to measure assembly difficulty (Chan and Salustri, 2003). The steps are as follows:

Step 1: *Specification*
Step 2: *Design*
Step 3: *Functional analysis* (This is the first Lucas analysis)
In this analysis, the components of the product are reviewed only for their function. The components are divided into two groups. Parts that belong to Group A are those that are deemed to be essential to the product's function; Group B parts are those that are not essential to the product's function. Group B functions include fastening, locating, etc.
Possibly loop back to step 2 if the analysis yields problems.
Step 4: *Feeding analysis* (This is the second Lucas analysis)
The part handling and insertion times are examined here. In the feeding analysis, the problems associated with the handling of the part are scored.
Step 5: *Fitting analysis* (This is the third Lucas analysis)
Step 6: *Manufacturing Analysis* (This is the fourth Lucas analysis)
The last part of the Lucas method is to calculate the cost of manufacturing each component. This manufacturing cost can influence the choice of material and the process by which the part is made. Although not a true "costing" of the part, this method does help guide designers by giving a relative measure of manufacturing cost.
Step 7: *Assessment* (Possibly return to step 2 if the analyses identify problems.)

This method does help guide designers by giving a relative measure of manufacturing cost. This part manufacturing cost allows designers to calculate the effect of part complexity versus part reduction. The problem with Lucas DFA is that it focuses on part reduction. This often results in multi-functional parts with very high complexity, which increases manufacturing costs.

2.1.2. Benefits of DFA

The DFA tools help to rate the product design in terms of its relative ease of difficulty for assembly. In particular, each part is analyzed with respect to how it is grasped, oriented and moved for insertion. This enables the identification of problem areas in the design, so that appropriate actions can be taken to mitigate these problems. DFA analysis guides the design

process, verifying improvement as the design evolves. As you eliminate redundant parts or operations and remove assembly difficulties, assembly efficiency scores noticeably improve. In the process the design complexity and cost is reduced.

For example, if the parts are provided with features which make them easier to grasp, move, orient and insert, this will also reduce assembly time and assembly costs. An additional result is that parts are identified as candidates for elimination. If a product contains fewer parts it will take less time to assemble, thereby reducing assembly costs. The reduction of the number of parts in an assembly has the added benefit of generally reducing the total cost of parts in the assembly. This is usually where the major cost benefits of the application of Design for Assembly occur.

Another major benefit of DFA analysis is that it also helps in comparison of various product designs with respect to ease of assembly and manufacture. This helps in benchmarking between various product designs alternatives or competing products.

Hence DFA as a tool assists in the product design decision making process and helps enhance and improve the design for efficient assembly and manufacture.

2.1.3. Limitation and criticisms of DFA

The DFA is a powerful tool and gives the designer an edge while making design decisions however without having considerable expertise in product design, it is not possible for a novice designer to make any design decisions based solely on the DFA analysis. The designer has to take into account all the other factors and the results obtained from other similar Design for Manufacture tools so that a sound decision can be made on improving the design. Hence DFA lacks the all in one feature for making a design decision. The DFA approach is limited to rigid objects. Flexible objects in computer manufacturing or other manufacturing processes are difficult to analyze using the DFA tool. This reduces the use of DFA to only a limited range of product designs. The cost savings benefit, however, sets the DFA tool apart from other similar DFX methods and hence its popularity in the industry.

2.2. Research on DFA and Product Development

The previous section provided an overview of the more popular DFA methods. In this section, more advanced research focused on improvements to DFA will be discussed. These improvements tend to customize the DFA analysis for a particular product or to enhance the decision making process during the design stage.

Mascle (2003) establishes a new concept called "FuzzyDFA". It is based upon the fuzzy logic principles that deal with the uncertainties of a designer. This technique may be used at early design phase where the product design can be optimized using the DFA methodology. It implements geometric algorithms to evaluate the assembly process automatically. Thus, Mascle (2003) introduces a fuzzy decision support system to enhance the effectiveness of DFA.

Wu and Xie (2007) proposed to create a linkage between design data in CAD with assembly operations in CAM. They introduced Open Structured Assembly Coding System (OSACS) a virtual coding system in order to reduce assembly costs. This system uses a virtual environment such as a CAD file in order to identify various features of the CAD model and visualize the part mating operations. The extractor is used to identify specific model codes to represent assembly operations in CAM. Thus, they create a virtual environment to analyze the product design and all its features with respect to manufacturing. The proposed design depicts significant cost savings and also connects the CAD/CAM phases.

Additional useful insights come from various works like Stone et al. (2003). They introduced a conceptual design for assembly method. It incorporates the DFA analysis in the conceptual design phase. The product architecture-based approach is shown in Figure 4. Thus, the product architecture-based conceptual DFA technique can be used to accelerate the rate of product improvement, or perhaps achieve a fully mature design in a first product offering.

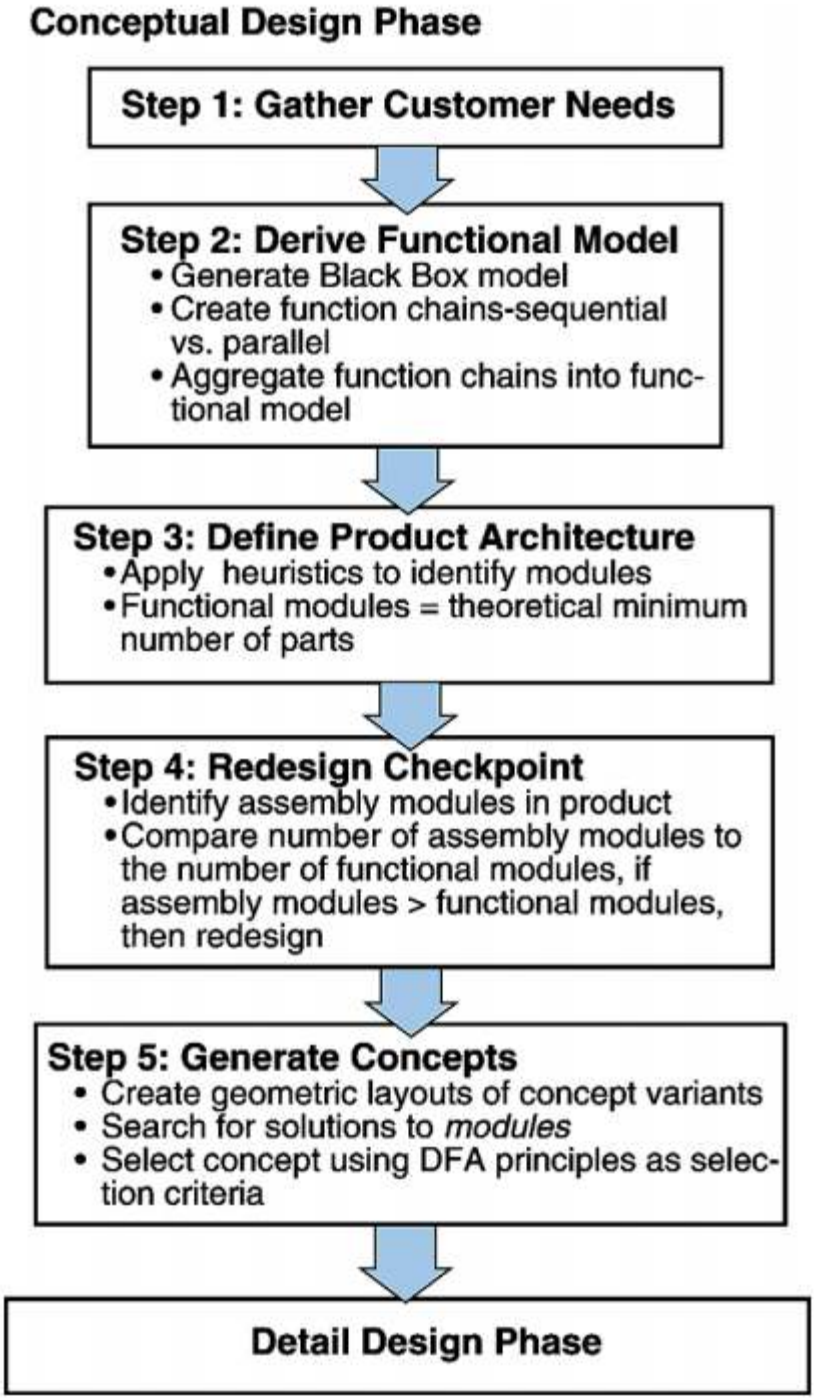


Figure 4: Product Architecture-based DFA (Stone et al., 2003)

2.3. Related Research

The literature in the area of linking DFA to manufacturing analysis is fairly new. This field is of significant interest today, as design engineers and manufacturing engineers have realized that bridging the gap between design and manufacturing will yield the best results.

Caputo and Pelagagge (2008) had a vision to evaluate the affect of product features on the presentation of manufacturing lines. They provided a specific rating to assembly components with respect to an assembly line. Caputo (2008) selects four distinct product features, mainly:-

1. the number of assembly tasks to be performed;
2. the average number of DOF in the assignment of a task to a station given the precedence constraints in the assembly sequence;
3. the ratio of average task time to the maximum task time TAVG/TMAX (Figure 5);
4. the ratio of the maximum task time to the cycle time TMAX/TC (Figure 5).

To generate various scenarios, Caputo and Pelagagge (2008) manipulated each of the four features at various levels. The scenarios led them to suggest guidelines for the designers based on the four features. This paper only considers average degrees of freedom so it qualitatively rates each product feature and suggests design recommendations. Also, it does not focus on finding the best solution for Line Balancing and cycle time simultaneously.

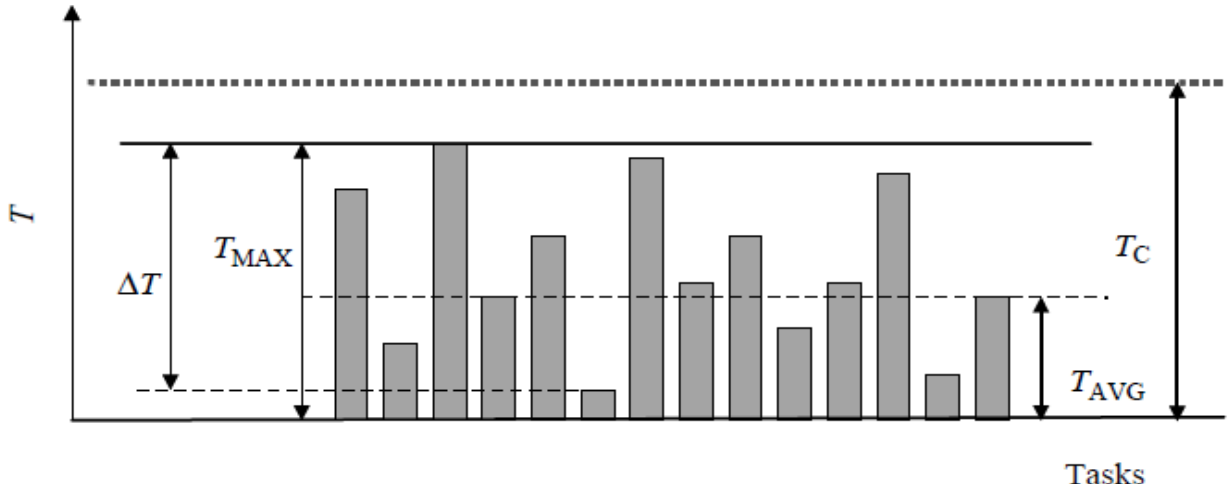


Figure 5: Effects of product design (Caputo and Pelagagge, 2008)

The Designers' Sandpit is an EPSRC-funded research project that aims to address these issues by developing an environment for "Assembly Oriented Design," incorporating methods for the generation and evaluation of concept design ideas, assembly planning and design advice.

The project (Ken and Swift, 1993) is a collaborative effort between the Universities of Hull and Cranfield -and a CAD software developer, Radan Computational Ltd. Their DFA research is currently focused on the following areas:

> *"Proactive DFA:* In order to improve the effectiveness of DFA so that it is useful in an assembly-oriented design environment, a proactive, rather than reactive, approach is required. This approach enables a 'right first time' design with minimal additional effort, so that subsequent changes are not required. Thus, it is important that the DFA evaluation becomes a fundamental part of the design process, occurring simultaneously with other design activities so that designers may optimize design in line with DFA recommendations. The notion of 'Proactive DFA' is aimed at providing the designer with the necessary tools for design evaluation at the earliest stages of the design and development process and thus ultimately, reducing lead times for product introduction.
>
> *Concept design:* To enable consideration of a product design, both in terms of structural configuration and ease of assembly, from the earliest stages of development, design software must be capable of representing concept design ideas. Furthermore, if a product design is to be optimized in the first instance, then the system should also assist the user in generating and evaluating alternative concepts. To achieve this, it is necessary to represent the functional requirements of a design so that appropriate design solutions may be selected. Such a representation also enables transmission of the design intent to the later stages of product development and ensures that satisfaction of the design requirements is maintained.
>
> *Geometric Reasoning:* The role of geometric reasoning is to reduce subjective and time consuming user input by the automatic extraction of data already available within the CAD model of a product design. In particular, the validation and evaluation of the assembly sequence is geometry-dependent, as are many elements of the DFA methodology.
>
> *Product complexity:* Measuring complexity is considered as a tool to support assembly-oriented design and to guide the designer in creating a product with the most effective balance of manufacturing and assembly difficulty. The goal is to provide the designer with such information throughout the design process so that an efficient design is produced in the first instance."

Lambert (2004) focuses importance of Line Balancing problems and the planning of assembly sequence. According to Lambert, sequence planning means developing the various steps that lead to assembly of a product. He establishes the use of precedence graphs for line balancing. These precedence graphs are basically AND/ OR graphs that help in the assembly sequence planning. These graphs help in selecting the optimum assembly sequence. The focus of this research is inclined towards the parallel tasks in an assembly sequence.

While there has been significant advancement of the DFA methodology over the years, no explicit link has been established between this DFA analysis and Manufacturing analysis, particularly Line Balancing and cycle time. The next section introduces the manufacturing concepts in greater depth.

2.4. Manufacturing Analysis

In a typical product development cycle, after the design of the product is complete the next step is to develop the manufacturing process and ultimately, the manufacture of the product. The focus of manufacturing analysis is typically to optimize the three parameters WIP, cycle time and throughput. A related topic is that of Line Balancing. In defining a methodology that explicitly links DFA and manufacturing analysis, these areas are important and will be discussed in greater detail below.

Work in Process (WIP): The inventory between the start and end points of a product routing is called work in process (WIP). Since routings begin and end at stock points, WIP is the entire product in between, but not including the ending stock points (Hopp, 2001).

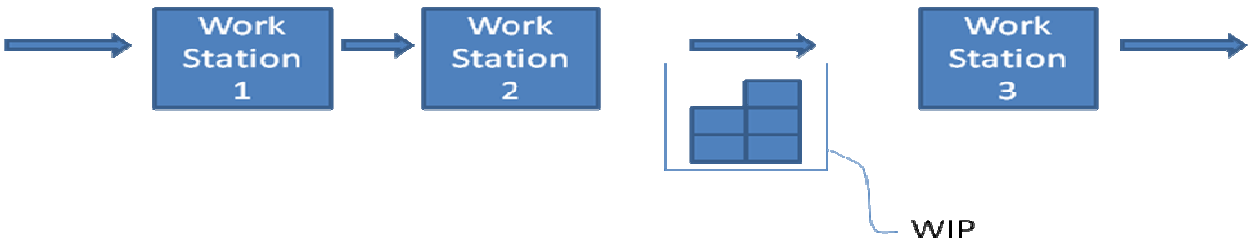


Figure 6: WIP

Figure 6 shows an assembly line with work in process. This WIP inventory includes the set of unfinished items for products between workstation 2 and 3. These items are not yet completed but are either waiting in a queue for further processing or in buffer storage. Optimal production management aims to minimize work in process. This WIP may indicate shortage in the supplies at workstation 3 or insufficient capacity to process the parts from workstation 2.

Cycle Time (CT): The cycle time of a given routing is the average time from the release of the job from the beginning of the routing until it reaches an inventory point at the end of the routing (Hopp, 2001).

This definition of cycle time is narrow and applies only to the assembly line as described in Figure 7 , i.e., for individual routings.

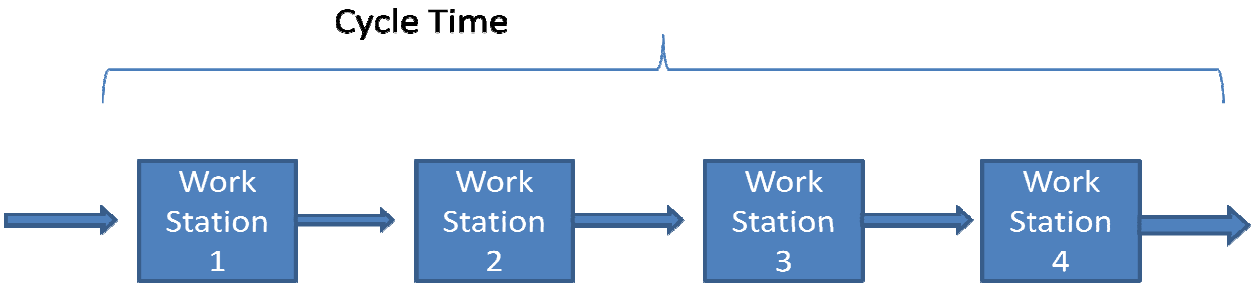


Figure 7: Cycle Time

Throughput Rate (TH): The average output of a production process per unit time is defined as the systems throughput. However for a plant throughput is the average quantity of parts produced per unit time. The throughput of workstations is different where workstations service multiple routings (Hopp, 2001).

According to Little's Law: $\boldsymbol{WIP = Throughput\ X\ Cycle\ Time}$ (Equation 1)

From the above little's law we see that there is a relation between these parameters. We would like to explore the link between the total assembly time for one product obtained from the DFA analysis and the cycle time required to manufacture the product.

The next focus of our research is to obtain a balanced line which will help the assembly to be lean from the beginning. Assembly Line Balancing, or simply Line Balancing (LB), is the problem of assigning operations to workstations along an assembly line, to optimize the operations. Ever since Henry Ford's introduction of assembly lines, LB has been of industrial importance (Falkenauer, 2000).

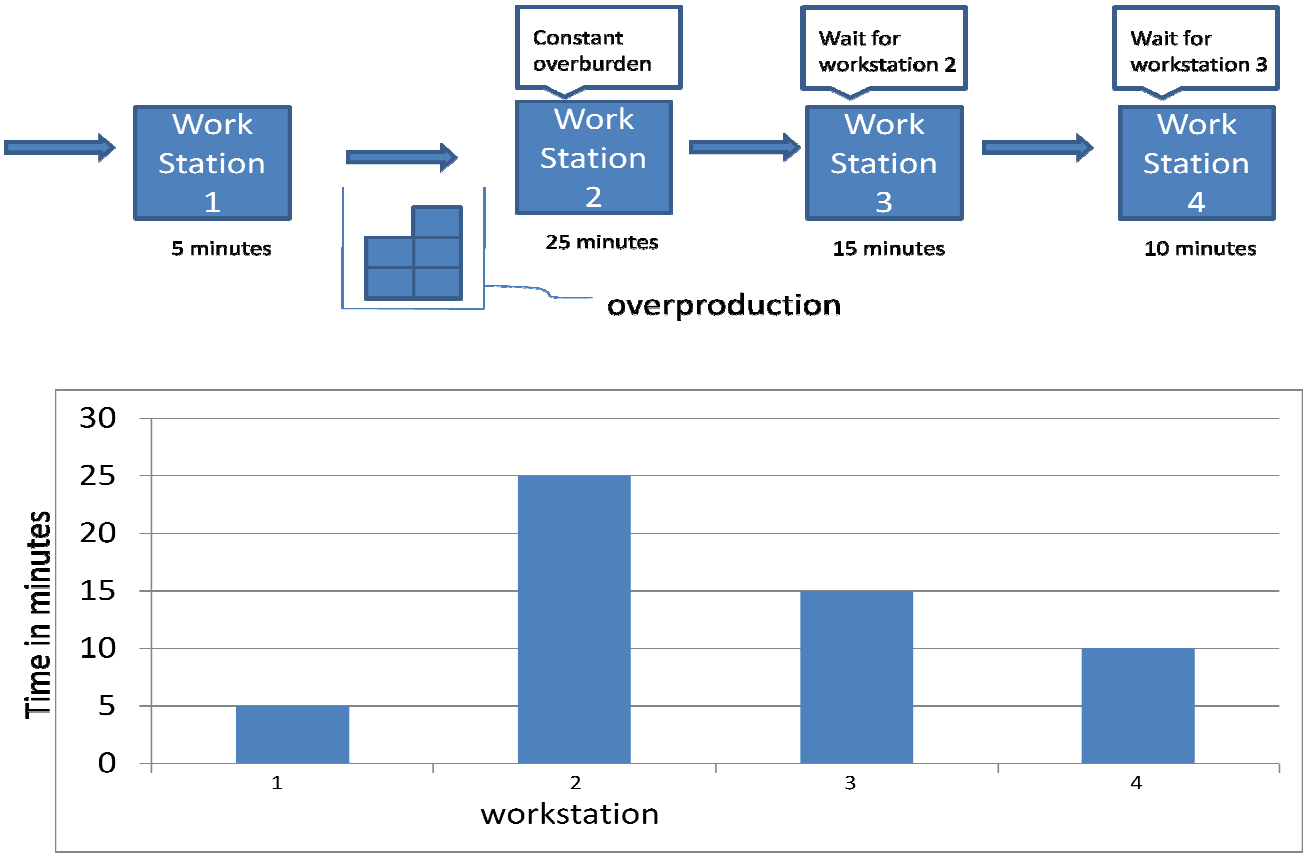


Figure 8: Line Balancing: an unbalanced line

For a manufacturing line to be balanced, the following must be taken into consideration:

- Everyone is doing the equivalent amount of work.
- Work is equivalent with requirements of the customer.
- There is less variation in output.
- No one must have more work , i.e., overburdened
- No one must be waiting for parts.
- Everyone must work in a balanced fashion.

Figure 8 depicts an unbalanced line where there is a constant burden on workstation 2. This can be resolved by distributing the load equally among workstations. A manufacturing analysis is typically carried out by first creating a current state map which is a diagram representing the current manufacturing flow of materials and processes. This map is then analyzed to understand the flow of materials, the production schedule, the daily orders, the quantity of production by each work station and the inventory level.

2.5. Summary

Briefly, DFA and manufacturing analysis are still too disjointed and creating a link between these entities is a challenge for researchers. The literature review presented here helps in understanding the two aspects of the research: design and manufacturing. First the various DFA methods and the research conducted in the DFA area have been investigated. Then the research on DFA helps in understanding the progress of the DFA research until the present day. The manufacturing analysis gives us a clear view of the parameters of Line Balancing and cycle time. We learn the significance of design and manufacturing and clearly see the opportunity to bridge the gap between them.

All the input data such as the schedule, the orders, the daily production, and the in-stock inventory is usually available at the production stage. However during the design stage this type of data is usually unknown. Hence, it is difficult to predict the manufacturing parameters like throughput, work in process and cycle time. This research is focused on finding the methodology to make it possible to estimate these parameters at the design stage of the product. These parameters further help balance the line and to reduce cycle time, which are critical manufacturing improvements at the design stage.

3. Methodology Development

3.1. Research Vision

The objective of this research is to take the DFA analysis a step further to consider throughput or cycle time and Line Balancing. The following research questions will be the focus of this work:

- Can an explicit link between DFA and assembly line performance be made?
- If so, can this link be leveraged to provide a method to aid product development practioners make development decisions?
- What kind of design actions can we take to optimize the cycle time given an initial design candidate?

Once the connection between DFA and assembly line performance, design changes can be implemented that would enable a focus on design for throughput.

Here is an overview of this idea. The first stage of the product is the design stage. At this point, the product is expanded and a detailed design is prepared. An assembly fishbone diagram is created to illustrate the overall assembly structure of the product. The fishbone diagram facilitates the execution of the DFA analysis which estimates the total assembly time of the product. Up to this point, this is standard practice for most of the DFA methods that are used. The goal of this research is to extend DFA by using the results of the DFA analysis to simulate an actual assembly process. From this simulation, manufacturing line performance metrics like cycle time, work in process and throughput can be estimated. With this knowledge, the product developer is now in a position to make designs that will influence assembly line performance. In this manner the link between design for assembly analysis to the actual assembly line can be established to aid in product development. These ideas are represented in Figure 9.

3.2. Methodology

The previous section provided an overview of the research vision. This section will provide a prescriptive summary of the steps involved in the methodology, with a more detailed explanation of each step in the following sections. Figure 9 will again be used as a frame of reference to describe the methodology.

Step 1. Representation of the Design candidate: A detailed enough representation of the design needs to be generated so that components and the precedence relations of these components can be understood. Note that this does not necessarily imply that a detailed design needs to be in place. As in the work of Stone (2003), a functional module could be defined as a 'component' that is yet to be defined.

Step 2. Fishbone Diagram: From the representation in Step 1, an assembly fishbone diagram is created to show the overall assembly structure of the product and its precedence relationships.

Step 3. DFA analysis: The actual DFA analysis is performed which will generate the following information:

Total assembly time
Time for each operation
Number of repetitions of each operation
Repetition time

Step 4. Manufacturing Analysis: These times and operations from the DFA analysis become inputs to a simulation model. Also, the precedence constraints from the design stage are inputs to the model. In the simulation stage the line is balanced such that the number of workstations is minimal while maintaining the production rate (takt time) and precedence constraints. This is done by software using the COMSOAL algorithm (Computer Method for Sequencing Operations for Assembly Lines) (Arcus, 1966). This serves as a baseline measure of the design.

Step 5. Relaxing precedence relationships: The precedence constraints are systematically relaxed. During this step, a line balance index and cycle time index are developed to identify components that are candidates for re-design.

Step 6. Redesign Action: Once possible opportunities are identified an improved design can be generated.

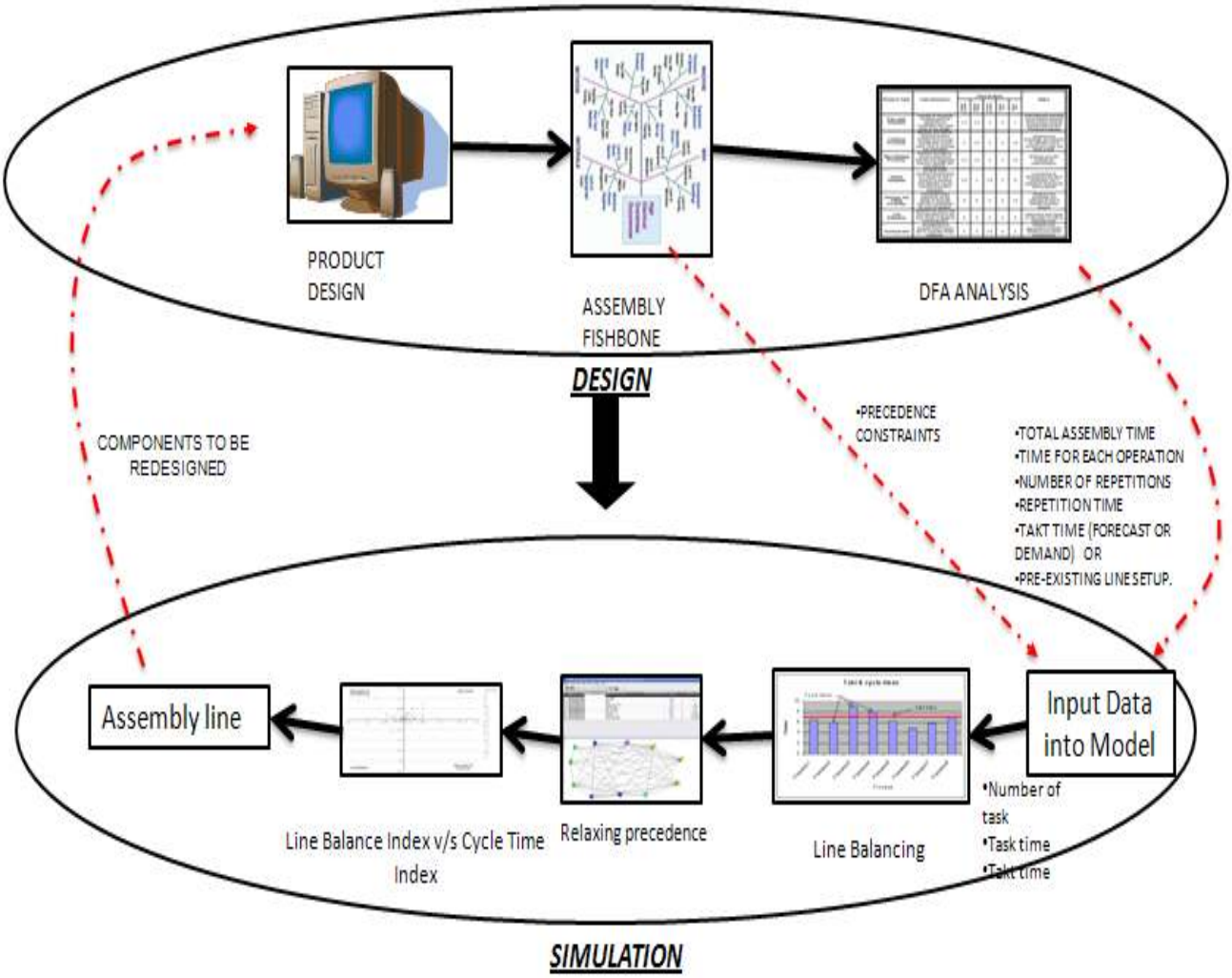


Figure 9: Design for assembly analysis linked to real assembly line

In order to validate the methodology, this process is implemented on a brake assembly case study which is described in chapter 4. The following section describes the preceding steps of the methodology in greater detail:-

3.2.1. Step 1: Representation of the Design Candidate

Product design is the first step towards the implementation of an idea or concept for manufacture of a product. The design candidate should go through the conceptual design phase as well as the detail design phase. The candidate must be well defined with respect to the number of components. The idea here is to analyze a design candidate, i.e., the design must be complete with an engineering sketch of each component. A sketch of the overall assembly of the design components is significant for considering it for manufacturing purposes. This sketch helps in defining the precedence relationships and the sequence of assembly for the particular design.

3.2.2. Step 2: Generate Fishbone Diagram

The fishbone diagram is a tool that assists DFA analysis. This fishbone diagram helps to visualize each of the steps in the assembly procedure. One of the most effective ways to enhance product design for ease of assembly is to plan the assembly process in advance. It is an advanced planning method which designers use for the assembly process. The designers are compelled to identify cost reducing assembly tasks. It serves as a document in order to evaluate assembly difficulties. Ishii and Kmenta (1995) introduced a fishbone diagram for depicting the assembly sequence. In fact, the fishbone diagram forms a key step while conducting the DFA analysis.

Figure 10 shows the core idea of a fishbone diagram using the mechanical pencil example from Ishii and Kmenta (1995). The handle is the significant component of the pencil and, therefore, is used as the base for assembly. From the interpretation of the assembly fishbone diagram, the core is inserted into the body and the cap fits over the body. The next step is to insert the tip and rotate it for attaching it to the body of the pencil. The arrows in the assembly fishbone are symbols which indicate insertion directions and reorientation. This information directly feeds into the DFA worksheet for computing assembly ratings using the revised

Westinghouse methodology (Ishii and Lee, 1995). The diagram may include other symbols indicating time penalty factors such as need for inspection and testing.

The procedure (Ishii and Lee, 1995) for constructing the assembly fishbone diagram is as follows:

1. Start with the part that other parts attach to.
2. Parts that attach directly are shown with a slanted arrow.
3. Denote special operations with icons next to arrows.
4. A subassembly consists of a separate tree that attaches to the main assembly.

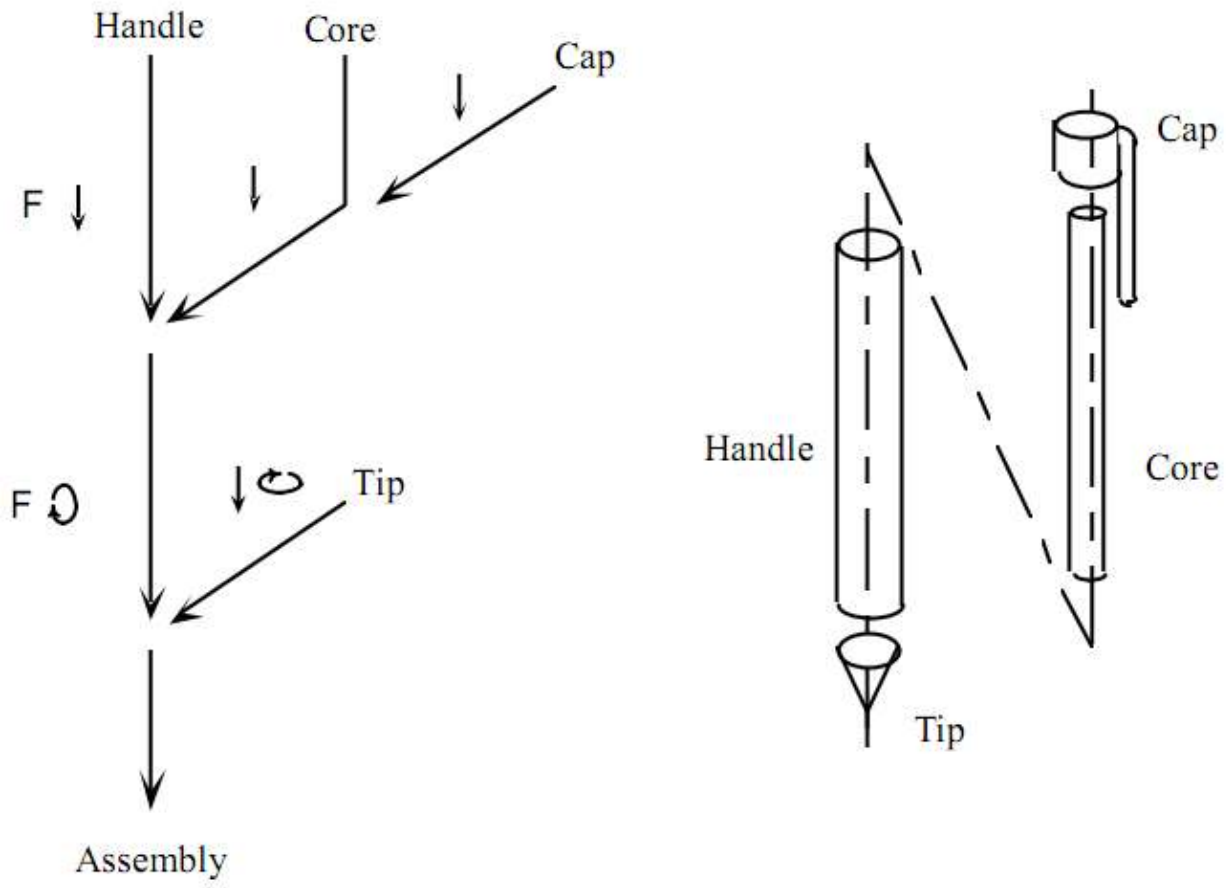


Figure 10: Assembly fishbone diagram of mechanical pencil (Ishii and Lee, 1995)

3.2.3. Step 3: DFA Analysis

The methodology begins with the DFA analysis applied to an existing product design. DFA is a formal analysis procedure that facilitates consideration of assembly issues, bringing together multi-disciplinary teams to validate and evaluate designs and assess their suitability for manufacture and assembly.

The Westinghouse DFA method forms a major part of the design phase in this research. The fishbone diagram in the previous section is an input to the Westinghouse DFA. The list of parts from the fishbone diagram is populated in the part description column of the Westinghouse DFA table which is included in Appendix A, Table 18. After the part description column is populated, the time required to insert each part or complete each assembly operation is estimated based on the Westinghouse tables (Hinckley, 2001). These time factors are influenced by various attributes such as size, shape, symmetry, weight and flexibility. The sum of all the time factors leads us to the time required for each operation. This operation time is then multiplied by the number of repetitions of a particular operation in order to get the repetition time. The total sum of all the repetition time presents us with the Total Assembly Time (TAT). This total assembly time is an estimate of the time required to assemble that particular assembly.

DFA methodologies that were discussed in the literature review were developed to support the designer by generating feedback on the consequences of design decisions on product assembly. From the various DFA methodologies the Westinghouse methodology has been chosen for use with this project due to the detail time for operation that is obtained from this analysis as seen in Appendix A, Table 18. Also, the author has more expertise in this methodology. This methodology takes into consideration the subtle variation in time factors like time for handling, time for orientation, time due to part size, etc. These subtle observations in time differences are important when calculating the total operation time. This is one of the major advantages of this method. Hence it is chosen for the analysis performed here.

3.2.4. Step 4: Manufacturing Analysis

The simulation stage visualizes the operating parameters of the manufacturing assembly line. The parameters of Line Balancing and manufacturing cycle time are most critical in

decision process during actual manufacturing. This stage thus simulates those parameters and helps in optimizing these parameters during product design.

The first step in the simulation stage is Line Balancing. The Line of Balance problem is very complex and requires an excessive amount of computational time. Also, the problem becomes NP hard so a heuristics approach is a more practical way to solve the Line Balancing problem. When these different procedures are compared one can conclude that heuristics is the best way to approach the real world Line Balancing problems. This section review related research and the methods used in optimum seeking procedures, as well as heuristics.

3.2.4.1. Review of Line Balancing Approaches

Line Balancing is an approach which helps in optimizing the work content throughout the assembly line. This in turn improves the throughput of the assembly process. Figure 11 shows the different procedures used to solve the Line Balancing problem.

As shown in Figure 11 there are four main categories of Line Balancing algorithms:

1. Optimum seeking procedures
2. Heuristics
3. Knowledge
4. Simulation

- **Optimum Seeking Procedures**

Sprecher (1999) attempts to solve the simple assembly line balancing problem type 1 (SALB-1) using a branch-and-bound algorithm. The algorithm relies on a linear programming approach to specify a precedence diagram. As the number of workstations are fixed in a SALB-1 problem the task are assigned to workstations linearly. Here he adapts a generic algorithm developed for the resource-constrained project scheduling problem (RCPSP) with multiple modes.

Peeters and Degraeve (2006) propose a classic integer programming approach to tackle the simple assembly line balancing problem. They develop a column generation algorithm which

aims for optimality but only gives an approximation of the optimum line balancing. This is mainly due to the fact that the computational complexity increases whenever the assembly is complex.

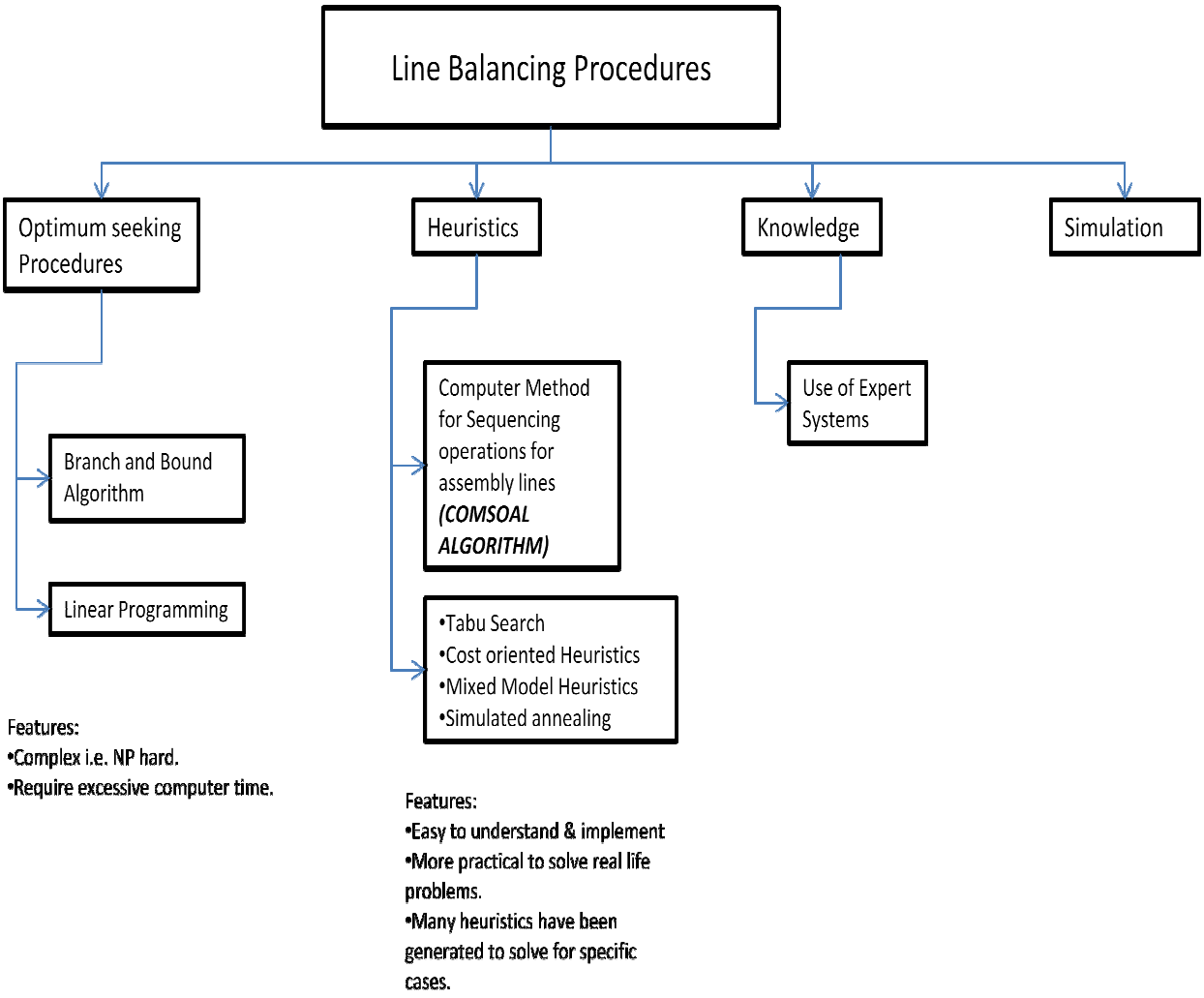


Figure 11: Line Balancing procedures

- **Heuristics: COMSOAL Algorithm**

The Computer Method for Sequencing Operations for Assembly Lines (COMSOAL Algorithm) was developed as part of an industrial Operations Research project (Arcus, 1966). It is a record keeping procedure that uses several lists for speed computation.

The COMSOAL program proceeds as follows (Arcus, 1966):

Step 1: For each task, identify those tasks which immediately follow it in precedence order.
Step 2: Place in LIST A for each task in the assembly, the total number of tasks which immediately precede it in the precedence diagram.
Step 3: From LIST A, create LIST B composed of the tasks which have zero predecessors. If no task remains unassigned to stations, then stop.
Step 4: From LIST B, create LIST C composed of the tasks whose performance times are no greater than the available time at the station. If LIST C is empty, open a new station with the full cycle time available and go through STEP 4 again.
Step 5: Randomly select from LIST C a task for assignment to the station.
Step 6: Update the time available at the station and LIST B to reflect the time consumed and the completed predecessors at this stage. If LIST B is empty update LIST A and return to STEP 3 otherwise return to STEP 4.

This algorithm is of two types as follows:

Type 1: Given TAKT, find the minimum number of workstations (N stations) which will maximize (Utilization).

Type 2: Given N workstations, find the minimum (TAKT) to maximize (utilization).

The COMSOAL algorithm simplifies complex assembly line problems. It is easy to understand & implement and more accurate than calculating by hand. The solution quality could be improved by increasing the iterations using computers.

- **Other Heuristics Methods**

The other heuristics methods focus on specific line types and multi-objective Line Balancing. Some of them are as follows:

1. Tabu Search
2. Multi manned workstations
3. U Line Balancing
4. Cost oriented algorithm
5. Simulated annealing algorithm

Pastor et al. (2002) present an integrated approach based on four heuristics. Their approach has two main objectives: (1) an improvement procedure based on tabu search, with the objective of minimizing the cycle time. (2) a second tabu search in order to increase the uniformity of the tasks performed at each workstation. This is a real world case study on an industry manufacturing four different products.

Dimitriadis (2005) studies an unconventional assembly line balance problem. This line balance problem involves multiple workers working on the same workstation and on the same product. This involves large parts like car body where multiple workers have to work on the same component. A heuristic approach is proposed as a two level procedure. This approach helps in optimizing number of workers needed and minimizing the total idle time of the line.

Chiang and Urban (2004) focus on an efficient heuristic procedure for the stochastic U-Line Balancing problem. This procedure constitutes of two major parts. The first one is the initial feasible solution and the second one is applied as an improvement module. This improvement module helps in finding near optimal solutions. This heuristic has been analyzed under different scenarios for improving the algorithm.

Amen (2001) focuses on solution quality and computing time requirements of heuristic methods for cost-oriented assembly Line Balancing. It compares existent and new heuristic methods for solving the cost-oriented assembly Line Balancing problem. The work emphasizes the economic view of production in order to cut down production cost.

Baykasoglu (2006) presents a new multiple objective simulated annealing (SA) algorithms for simple and U-type assembly Line Balancing problems with the aim of minimizing the number of workstations. This algorithm makes use of task assignment rules in constructing feasible solutions. It shows how task assignment rules can be included into an optimization routine to optimize an assembly line.

3.2.4.2. Baseline Line Balancing Analysis

Line Balancing is done on the product design "as is", i.e., first product design is analyzed for optimum line balance. This is done using the COMSOAL algorithm which is a heuristic approach to Line Balancing. The COMSOAL algorithm is easy to understand and implement. Also, it yields precise results for complex line balancing problems. In this case, a takt time is assumed because generally the number of workstations is decided based on the takt time and not vice-versa. Line Balancing is done either by optimizing the number of workstations or by optimizing the total number of workstations. Thus, the total operation time and time for each operation obtained from the DFA analysis are an input to the Line Balancing software. From this

Line Balancing the parameters such as number of workstations, the time at each workstation and the total manufacturing cycle time are obtained. The results from this step form the baseline. This baseline design is then manipulated and all results are compared with this baseline to identify redesign opportunities.

3.2.4.3. Development of Performance Indices

Given a baseline design, it would be useful to have a set of metrics that would identify areas in the existing design that would lead to improved manufacturing performance, specifically Line Balancing and cycle time. In this section, the use of the baseline design obtained above is helpful in the development of these performance indices. Since the goal is to assist the design engineer in the decision making process a graphical way of presenting the results is desirable. This would lead to simple interpretation of the results obtained.

Given this vision, Figure 12 shows a conceptual graph of a Line Balancing index versus a cycle time index. With a graph like this, the designer can easily determine which of the components are the affecting the manufacturing parameters the most. The idea is that the higher the index, the greater the impact the component has on improving the desired manufacturing performance dimension.

The graph as shown in Figure 12 is arbitrarily divided into the following four regions:

1. High Impact region: A point in this region of the graph signifies that the component and/ or interaction have a very high Line Balancing index and very high Cycle time index. Thus, eliminating or redesigning this task and/ or interaction will enable us to achieve better line balance and will show higher reduction in the total cycle time as compared to component and/ or interactions in other regions of the graph for the specified takt time.

2. High Impact on Line Balancing Region: A point in this region of the graph signifies that the component and/ or interaction has a very high Line Balancing index but not so high Cycle time index as compared to a task and/ or interaction in the High Impact region. Thus, eliminating or redesigning this task and/ or interaction will enable us to achieve better line balance but will not significantly affect reduction in the total cycle time as compared to the task and/ or interaction in the High Impact region for the specified takt time.

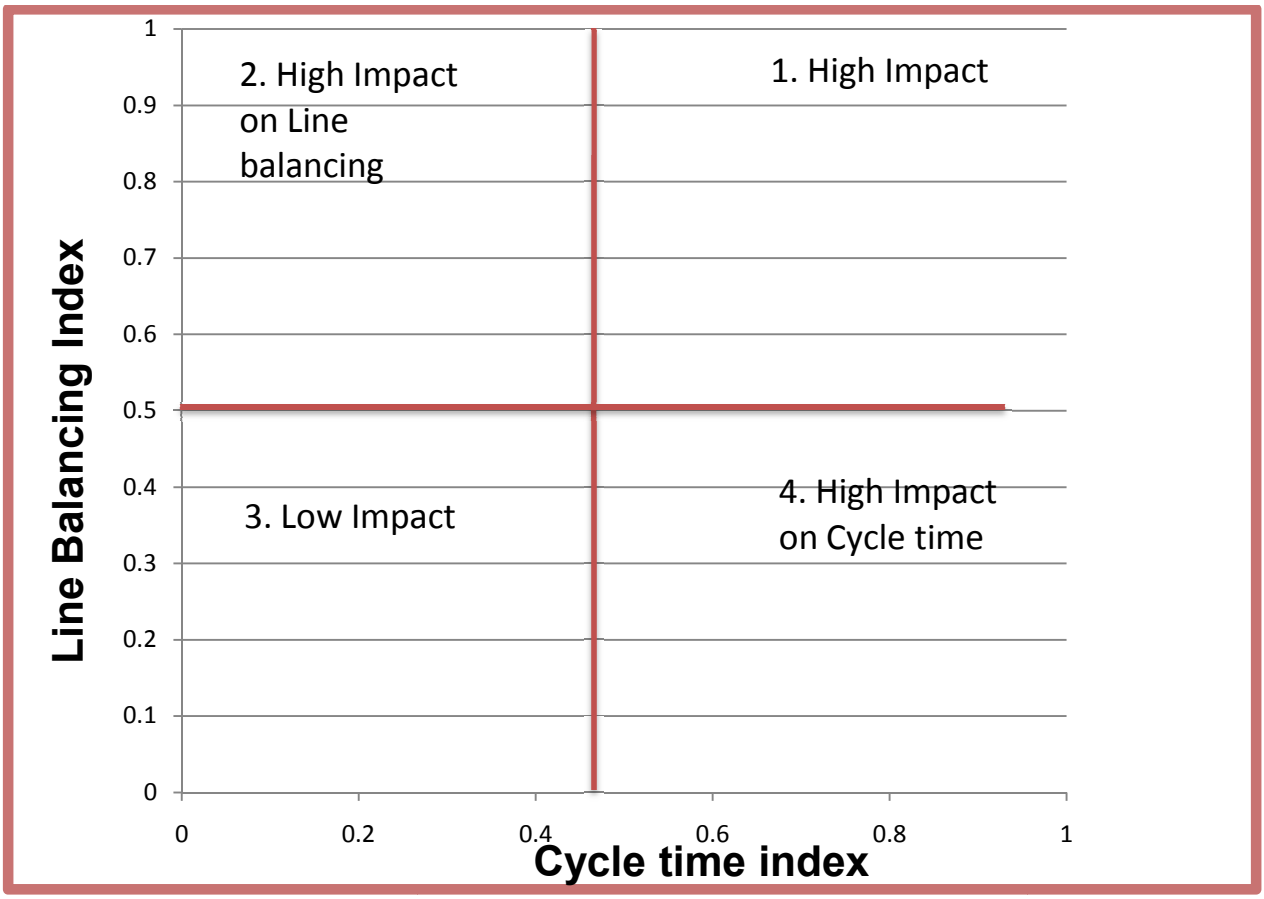


Figure 12: LBI vs. CTI graph

3. Low Impact Region: A point in this region of the graph signifies that the component and/ or interaction has a very low Line Balancing index and a very low Cycle time index as compared to all other regions in the graph. Thus, eliminating or redesigning this task and/ or interaction will not show a significant effect on line balance and total cycle time as compared to task and/ or interactions in other regions of the graph for the specified takt time.

4. High Impact on cycle time region: A point in this region of the graph signifies that the component and/ or interaction has a very high Cycle time index but not so high Line Balancing index as compared to a task and/ or interaction in the High Impact region. Thus, eliminating or redesigning this task and/ or interaction will enable us to reduce total cycle time but will not significantly affect line balance as compared to the task and/ or interaction in the High Impact region for the specified takt time.

Now the cycle time index is a comparison of the baseline cycle time with the new cycle time index obtained after the change in parameters. Similarly the Line Balancing index is a comparison of the baseline with the new line balance obtained after the change in the parameters. Thus, the graph helps in improving the cycle time index and the line balance index by identifying the components which affect these parameters the most. The high impact area in the graph presents the candidates which need to be considered first for redesign.

Thus, the development indices are developed from this concept of evaluating the line balance index versus cycle time graph.

- Cycle time Index: From the definition of Cycle time:

$$Cycle\ time\ (CT) = \ Neck\ time(NT) X\ Number\ of\ Workstations \quad \text{(Equation 2)}$$

Where, Neck time is the time at the workstation, which takes highest time.

The number of workstations is obtained from the line balance. Hence the Cycle Time Index is calculated as follows:

$$Cycle\ Time\ Index\ = \left(1 - \left(\frac{CT}{CT\ Baseline}\right)\right) \quad \text{(Equation 3)}$$

Where CT Baseline = Cycle time of assembly line considering all components , i.e., the original design.

The higher the index the more the improvement in Cycle time meaning a reduction in cycle time. The Cycle time index is bound between 0 and 1. However, the Cycle time index may be negative. This negative index indicates an increase in cycle time as compared to the baseline, i.e., the original design which is a negative effect.

- Line Balance Index: From the Line Balancing result,

SS takt time error = $\sum_{wk=1}^{n} ((\mathbf{TT - wk1})^2 + \cdots . + (\mathbf{TT - wkn})^2)$ (Equation 4)

The sum of squares takt time error is the sum of squares of the difference between the work station time and the takt time.

SS total = $\sum_{wk=1}^{n} ((\mathrm{TT - wk1})^2 + \cdots . + (\mathrm{TT - wkn})^2)$ (Equation 5)

Where TT = takt time

wkn = assembly time at Workstation n

SS = sum of squares

The sum of squares total is the sum of squares of the workstation time. Thus, now we calculate the Line balance Index as follows:

Line Balance Index = $1 - (SS\ takt\ time\ error/SS\ total)$ (Equation 6)

These calculations are done for each of the tasks by relaxing the precedence constraint on one task at a time.

3.2.5. Step 5: Relaxing Precedence Relationships

After the Line Balancing output is obtained considering all precedence constraint, we now repeat the procedure by systematically relaxing all the precedence constraint on each operation of the assembly. This is done in sequence, i.e., the precedence constraint is relaxed on only one operation at a time and the line is balanced to observe the effect of the part on the entire assembly without considering precedence. A constraint on a component, in an assembly, restricts the freedom of the component to be used anywhere in the assembly sequence.

If we relax the constraint on a component we can see the effect of that component on the assembly sequence and therefore on the line balance and cycle time. The relaxing of the precedence constraint enables us to identify the components for redesign. This can be further explained considering the assembly in

Figure 13.

The Allen screw (3) has two precedence constraints, (1) the motor and (2) the center plate. The arrows show the precedence of the motor and the center plate on the Allen screw. This means that the motor must be assembled with the center plate before we fasten the Allen screws on the motor.

Figure 13 shows the Allen screw motor with and without precedence constraints. The feasible assembly sequences considering precedence constraint are:

1->2->3 or 2->1->3

Now completely relaxing the precedence constraint on the Allen screw the feasible assembly sequences will be:

1->2->3 or 1->3->2 or 2->1->3 or 2->3->1 or 3->2->1 or 3->1->2

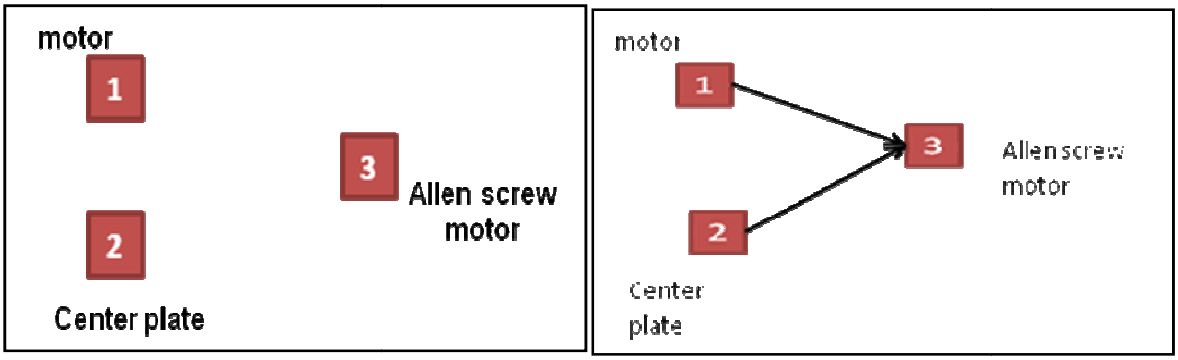


Figure 13: a) Allen screw with precedence constraints b) without precedence constraints

With the aim of creating the different scenarios to compare with the baseline relax the precedence constraints on each of the tasks systematically. Consider the main components at the receiving end of the precedence constraints and the precedence at the supplying end. This leads into the two permutations: the row permutation and the column permutation respectively.

3.2.5.1. Row Permutation

The initial permutation is the row permutation where the thought is to relax the precedence relationships of each task systematically. In order to recognize the effect of interaction of task on Line Balancing and cycle time, we consider all combination of tasks and systematically relax constraint on each task, as well as all the combination of tasks, and then run

the Line Balancing algorithm for each combination. This is further explained with the help of the following example:-

Consider an assembly A with 4 tasks as shown in Table 3.

Table 3: Assembly A

Task	Task Time	Precedence
1	6	
2	5	
3	4	1,2
4	2	1,2,3

Here you can see that task 1 and task 2 have no precedence constraint. The task 3 has two precedence constraints and the task 4 has three precedence constraints.

Now to understand row permutation, consider task 3 using, the following procedure:

Step 1: Relax all precedence constraints on task 3 as shown in

Table **4**.

Table 4: Assembly A with all precedence on task 3 relaxed

Task	Task Time	Precedence
1	6	
2	5	
3	4	
4	2	1,2,3

Step 2: Run the line balance algorithm and obtain the results

- Number of workstations for the given takt time.
- Takt time
- Assembly time at each workstation

Step 3: Now relax only precedence 1 on task 3 as shown in Table 5 and then run the line balance and obtain the result.

Table 5: Assembly A with only one precedence on task 3 relaxed

Task	Task Time	Precedence
1	6	
2	5	
3	4	2
4	2	1,2,3

Step 4: Then relax only the next precedence constraint on task 3, i.e., "precedence 2" as shown in Table 6 and then run the line balance algorithm and obtain the result.

Table 6: Assembly A with next precedence on task 3 relaxed

Task	Task Time	Precedence
1	6	
2	5	
3	4	1
4	2	1,2,3

Step 5: Repeat this procedure for each task and its corresponding precedence constraints in the assembly and obtain the result for the generation of the metrics.

3.2.5.2. Column Permutation

Similarly in column permutation, to recognize the effect of interaction of precedence on Line Balancing and cycle time we consider all combination of precedence on a task and then run the Line Balancing algorithm. This can be further demonstrated with the help of the following example.

Consider an assembly B with 4 tasks as shown in

Table **7**.

Table 7: Assembly B

Task	Task Time	Precedence
1	6	
2	5	
3	4	1,2
4	2	1,2,3

Here the task 1 and task 2 have no precedence constraint. Task 3 has two precedence constraints, and task 4 has three precedence constraints.

In order to better understand column permutation, we follow the following procedure:

Step 1: Relax the precedence constraint 1 on all the tasks as shown in Table **8**.

Table 8: Assembly B with precedence 1 on all tasks relaxed

Task	Task Time	Precedence
1	6	
2	5	
3	4	2
4	2	2,3

Step 2: Run the line balance algorithm and obtain the result

- Number of workstations for the given takt time.
- Takt time.
- Assembly time at each workstation.

Step 3: Now relax only precedence 2 on all the tasks as shown in Table 9, and then run the line balance and obtain the result.

Table 9: Assembly B with precedence 2 on all tasks relaxed

Task	Task Time	Precedence
1	6	
2	5	
3	4	1
4	2	1,3

Step 4: Then relax the combination of precedence 1 and 2 on all tasks as shown in Table 10 and then run the line balance algorithm and obtain the result.

Table 10: Assembly B with precedence 1 and 2 on all tasks relaxed

Task	Task Time	Precedence
1	6	
2	5	
3	4	

4	2	3

Step 5: Repeat this procedure for each precedence constraint and the combination of all precedence constraints in the assembly and obtain the result.

The row and column permutations thus facilitate the creation of all possible scenarios of relaxing precedence relationships. Once all this data has been generated it can be compared with the baseline. Also, by systematically considering all permutations and combinations of the precedence constraints on each of the tasks, the significant precedence constraints and their interactions which affect the line balance and cycle time can be identified. These row and column permutations help to create the graph of Line balance index versus Cycle time index.

3.2.6. Step 6: Redesign Action

3.2.6.1. Analyzing the Data

Using the procedure defined in the previous section, all combinations of row and column permutations are generated, which need to be further analyzed to identify the components for redesign. This is accomplished by a statistical analysis which identifies the individual components, or combination of components which are responsible for largest decrease in overall performance of the manufacturing parameters. Or stated another way, have the largest impact metrics. The statistical analysis is done using a Design for Experiments approach in which the significance of component main effects and interaction effects if formally examines. Further details on the specific analyses conducted are in section 4.6.

3.2.6.2. Identifying Components for Redesign

The action table is developed with the aim of identifying the component for redesign and assisting the product designer. This action table summarizes the types of precedence constraint, the analysis to identify these precedence constraints and the design recommendations.

The action table is shown in Table 11. According to the action table there are four types of constraints namely:-

1. Over-constraint
2. Under- constraint
3. Properly constrained
4. Mistake

The constraint analysis action table makes it easier for the designer to comprehend the various constraints. For an over-constrained motion it is recommended to redesign the component or combine it with other component. If the component is under-constrained it is recommended that the component be redesigned or eliminated. It is not necessary to redesign a properly constrained component as it meets the criteria of constrained motion. A mistake should be eliminated as it affects the proper functioning of the component and the entire assembly.

Table 11: Action table for design recommendations

Symbol	Constraint	Interpretation	Analysis	Recommendations
U	Under-constrained	Degree of freedom has no value and it is required or necessary	Motion Analysis	Redesign or combine components for making the assembly properly constrained
O	Over-constrained	Degree of freedom has more than one value creating locked in stress	Constraint analysis	Redesign or eliminate the component for making the assembly properly constrained
P	Properly Constrained	The part is neither over constrained nor under constrained	not required	If all constraints are properly constrained then analyze the assembly of the part and the mating parts as a whole for opportunities of redesign
M	Mistake	Non functional over constraint or under constraint	not required	Eliminate

The product designer can now make decisions based on the LBI vs. CTI graph as referred to in Table 11. The graph will help to identify possible components for redesign which can optimize the process parameters of Line Balancing and cycle time depending on the desired takt time.

4. Case Study: Brake Assembly

Following the development of the methodology, it was implemented on a case study in order to test the methodology and improve it. The Brake Assembly used here is a widely used component in automobiles and was easily available.

4.1. Step 1: Design Candidate: Brake Assembly

The brake assembly commonly used in automobiles is an Antilock Brake system. The exploded view of the brake assembly is shown in Figure 14. The first step is to create a table of the components of the Brake assembly. The Brake assembly has the following parts as shown in Table 12. Table 12 also, gives the quantity of parts and a brief description of the application of the part in the working of the Brake assembly.

Table 12: Parts of the Brake Assembly

Part number	Part Name	Quantity	used for
1	Motor	1	for rotational movement to activate the system
2	Center Plate	1	regulates fluid to brake system lines
3	Flange	1	protection between side plate and center plate
4	Electric controls box	1	housing for electrical connectors
5	head pegs	3	to be inserted in bushes for mounting purposes
6	Allen screw	4	fasteners for fixing center plate to electric controls box
7	Allen screw motor	2	fasteners for fixing motor to center plate
8	thread cover small	2	for separation of center plate and side plate
9	cylindrical pegs	4	covering for threads (storage)
10	thread cover large	1	covering for threads (storage)
11	bushes	3	to absorbs shocks(preassembled with Center Plate)

The most functional parts of this assembly are the motor, the center plate and the electric controls box. The motor plays a major role in engaging and disengaging the brake. The center plate consists of the necessary valves that open and close in order to regulate the flow of the brake fluid to the system lines. The electric controls box is a circuit for human-machine interface which converts the electrical signals into the mechanical movements of the motor thus resulting in the braking action.

The Allen screws form the support structure of the Brake Assembly. The flange is used as a spacer between the electric controls box and the center plate. The thread covers are used for protection of the threads during storage. Thus, the brake assembly is a complex system of parts put together to perform a single function, i.e., the braking system of an automobile.

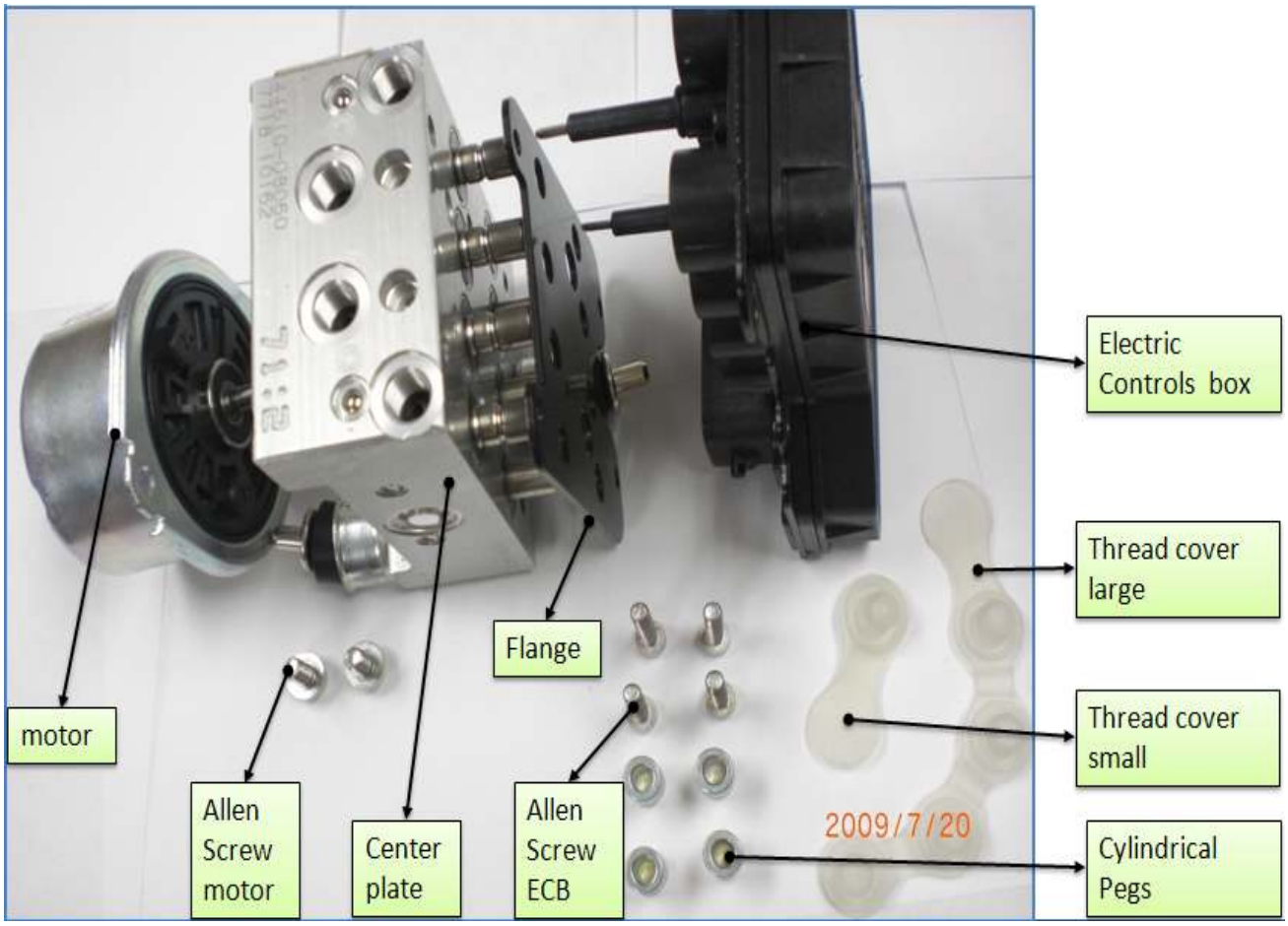


Figure 14: Exploded view of the Brake Assembly

4.2. Step 2: Fishbone Diagram: Brake Assembly

Now, following the parts, list the next step is to draw the assembly fishbone diagram as shown in Figure 15 which helps in understanding the flow of assembly and precedence constraints. The fishbone diagram helps in visualizing the assembly sequence of the brake assembly. It also enlists the tasks necessary for assembly which is useful for the next step of Design for Assembly.

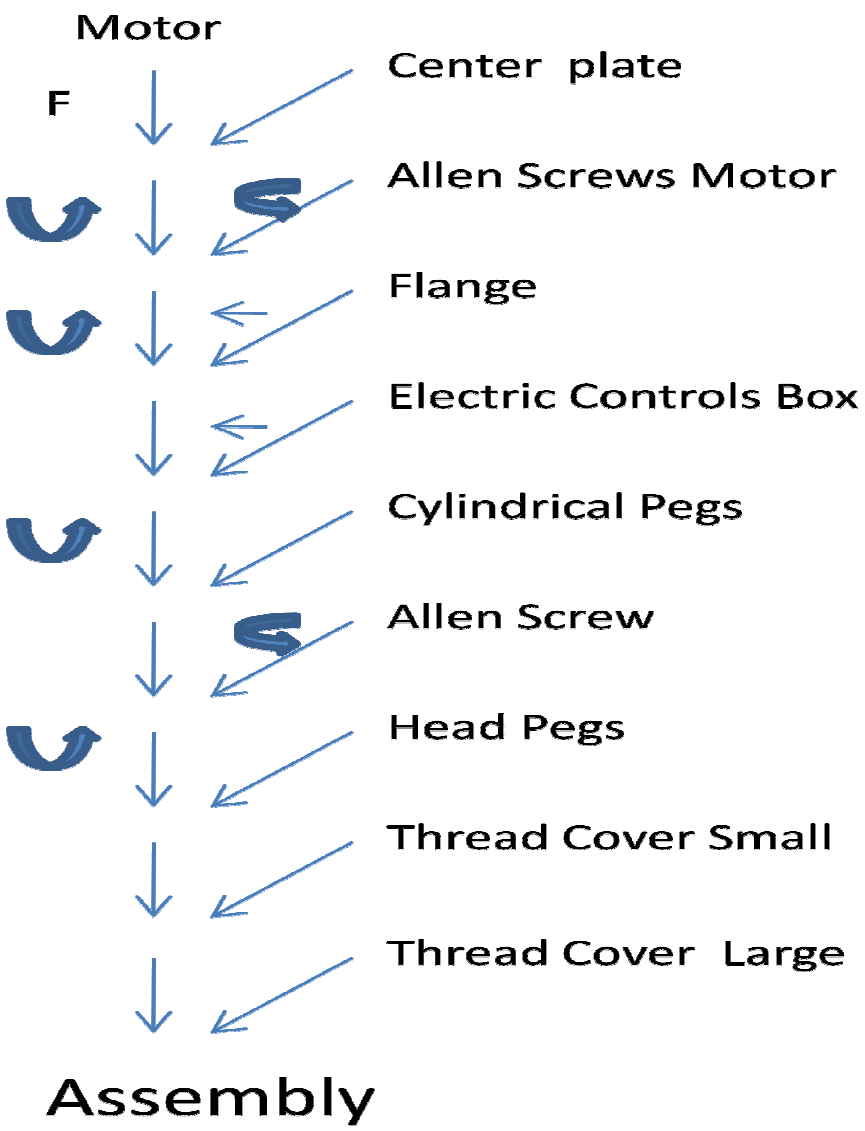


Figure 15: Fishbone Diagram of Brake assembly

The Fishbone diagram is shown in Figure 15. The motor is first assembled with the center plate. The next task or operation is to fix the motor on the center plate by means of screws. The circular arrows shown in Figure 15 depict the fastening operation of the Allen screws. After this operation the next operation is to insert the flange onto the center plate. The Electric controls box succeeds in this task followed by the cylindrical pegs. The next step is to fasten the Electric controls box with the center plate by means of four Allen screws. The thread cover small and thread cover large, which protect the threads are snapped into place at the end of

the assembly. Therefore, the assembly sequence and can be easily visualized by building the fishbone diagram.

4.3. Step 3: DFA Analysis

The next phase is to carry out the DFA analysis. The Westinghouse DFA method is used here to calculate the total assembly time. From this method we also, obtain the time for each operation and the number of repetitions for each operation which is further useful for balancing the workstations.

Table 13: Westinghouse DFA

		Time Factors (seconds)														
		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
No.	Part/Operation Description	End-to-End Orientation	Rotational Alignment	Part Size	Part Thickness	Insertion Clearance	Insertion Direction	Insertion Condition	Fastening	Fastening Process	Handling Condition	Time/Each Operation (T••)	Number of Repetitions (N••)	Repetition Time (K*L) (T••)	Insert Part (1 = Yes; 0 = No)	Eliminate Part (1=yes; 0 = No)
1	Motor	0.8	1.5	0.0	0.0	1.6	1.4	1.3	0.0	0.0	0.0	6.6	1	6.6	1	
2	Center Plate	0.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	1	0.5	1	
3	Flange	2.3	1.0	0.0	0.2	0.0	1.4	0.0	0.0	0.0	0.0	4.9	1	4.9		
4	Electric controls box	0.8	1.0	0.0	0.0	0.0	1.4	0.0	0.0	0.0	0.0	3.2	1	3.2	1	
5	head pegs	0.8	0.5	0,1	0.0	0.9	0.6	0.0	1.0	1.0	0.0	4.8	3	14.4		
6	allen screw	1.8	0.0	0.1	0.0	0.3	0.6	1.3	4.0	4.0	0.0	12.1	4	48.4		
7	allen screw motor	1.8	0.0	0.1	0.0	0.3	0.6	1.3	4.0	4.0	0.0	12.1	2	24.2		
8	thread cover small	1	1	0	0	0	1	0	1	1	0	4.6	2	9.2		
9	cylindrical pegs	1	0	0	0	0	1	1	0	0	0	3.6	4	14.4		
10	thread cover large	1	0	0	0	0	1	0	1	4	0	6.6	1	6.6		
													20	132.4	3	
													TOP	TAT	NUP	

Step 1: Draw the Assembly Sequence Diagram
Step 2: List Parts & operations in order (left column)
Step 3: Enter times from Estimated DFA Time Chart
Step 4: Sum time per part/oper. in column K
Enter no. of repetitions for each operation in col. L
Enter K*L in col. M
Step 5: Enter a 1 in col. N if a part was inserted during operation
Enter a 1 in col. O if part or operation can be eliminated
Step 6: Calculate Summary Statistics

Summary Statistics		
NUP	3	= number of unique parts (Sum of Column N)
TOP	20	= total number of operations (sum of Column L)
TAT	132.4	= total assembly time (sum of Column M)
NP	3	= number of parts = sumproduct(L,N)
$T_{...}$	6.6	= avg time/operation = TAT/TOP
$P_{...}$	3.0	= min # parts = NP - sumproduct(L,N,O)
AR	0.05	= Assembly rating = 2.35 * NP /TAT
PE	1.00	= Part Effciency = Pmin/NP
C	84.40	= Assembly complexity = TAT - (2.4*TOP)
OR	2.76	= Operation difficulty rating = TAT/(2.4*TOP)

As shown in Table 13 for all the parts of the Brake assembly various time factors such as end to end orientation, rotational alignment, part size, etc. are considered for calculating the time for each operation. The time for each operation when multiplied by the number of repetitions for that operation gives us the repetition time for that particular operation. The summation of all the repetition time for each operation is the total assembly time for the Brake assembly.

The total assembly time as shown in the Table 13 is 132.4 seconds. Also, time for each operation and number of operations required to assemble the part are obtained from the DFA.

As a summary the following are the inputs to the DFA analysis from the Fishbone diagram:

1. Tasks or operations
2. Time factors calculated using tables.

The following are the outputs obtained from the DFA analysis:

1. Total assembly time
2. Number of repetitions of each task or operation

The next step is to distribute these tasks among workstations and balancing the assembly line according to the takt time.

4.4. Step 4: Manufacturing Analysis

After the DFA analysis phase, the assembly steps are distributed among the workstations depending on the demand time, i.e., the takt time. Here we apply the COMSOAL algorithm. Considering the precedence constraints and to distribute the work equally among workstations, in this case we assume takt time to be 32 seconds.

No	Workstatio	Elem Task Description	ST	Elem Task #	Prec
1	1	motor	6.6	1	
2		centerplate	0.5	2	
3		allen screw	24.2	3	1,2
4	2	electric controls box	3.2	5	2,4
5		flange	4.9	4	2
6		cylindrical pegs	14.4	6	1,2,3,4,5
7	3	allen screw ECB	24.2	7	2,4,5
8	4	allen screw ECB	24.2	8	2,4,5
9	5	head pegs	14.4	9	1,2,3,4,5,6,7,8
10		thread cover small	9.2	10	1,2,3,4,5,6,7,8,9
11		thread cover large	6.6	11	1,2,3,4,5,6,7,8,9

Figure 16: Flexible Line Balancing software

The Flexible Line Balancing software is used for the purpose of Line Balancing. This software utilizes the COMSOAL algorithm to compute the line balance and the output is in a graphical format as shown in

No	Workstatio	Elem Task Description	ST	Elem Task #	Prec
1	1	motor	6.6	1	
2		centerplate	0.5	2	
3		allen screw	24.2	3	1,2
4	2	electric controls box	3.2	5	2,4
5		flange	4.9	4	2
6		cylindrical pegs	14.4	6	1,2,3,4,5
7	3	allen screw ECB	24.2	7	2,4,5
8	4	allen screw ECB	24.2	8	2,4,5
9	5	head pegs	14.4	9	1,2,3,4,5,6,7,8
10		thread cover small	9.2	10	1,2,3,4,5,6,7,8,9
11		thread cover large	6.6	11	1,2,3,4,5,6,7,8,9

Figure 16.

No	Workstatio	Elem Task Description	ST	Elem Task #	Prec
1	1	motor	6.6	1	
2		centerplate	0.5	2	
3		allen screw	24.2	3	1,2
4	2	electric controls box	3.2	5	2,4
5		flange	4.9	4	2
6		cylindrical pegs	14.4	6	1,2,3,4,5
7	3	allen screw ECB	24.2	7	2,4,5
8	4	allen screw ECB	24.2	8	2,4,5
9	5	head pegs	14.4	9	1,2,3,4,5,6,7,8
10		thread cover small	9.2	10	1,2,3,4,5,6,7,8,9
11		thread cover large	6.6	11	1,2,3,4,5,6,7,8,9

Figure 16 shows the user interface of the software. The following are the inputs to the user interface from the DFA analysis:

1. Elemental task description
2. Task time denoted as “ST”
3. Precedence constraint for each task denoted as “Prec”

The number of operations from DFA and the operation time is then entered into the Flexible Line Balancing software. Also, the precedence constraints are entered into the model before Line Balancing. After we input the desired model the algorithm is executed to obtain the line balance. The algorithm decides the optimum assembly sequence based on the inputs and the optimum line balance is obtained according to takt time.

Figure 17 below shows the model of the Brake assembly considering precedence constraints. The arrows show the order of precedence. Here we see that the motor which is part number 1 has no precedence constraint, but the Allen screw motor, which is part number 3, is preceded by the motor and the center plate (part number 2) as shown by the arrows in Figure 17. Similarly, all other parts are depicted with arrows showing their preceding constraints.

Thus, the software now has a range of feasible assembly sequences given the takt time of 32 seconds. When the model is run with the takt time of 32 seconds the following Line Balancing output is obtained as shown in Figure 17.

- Number of workstations = 5
- Takt time = 32seconds
- Neck time , i.e., workstation which takes the highest time = 31.3 seconds
- Number of operators = 5 (This is the default as the algorithm assumes one operator per workstation.)

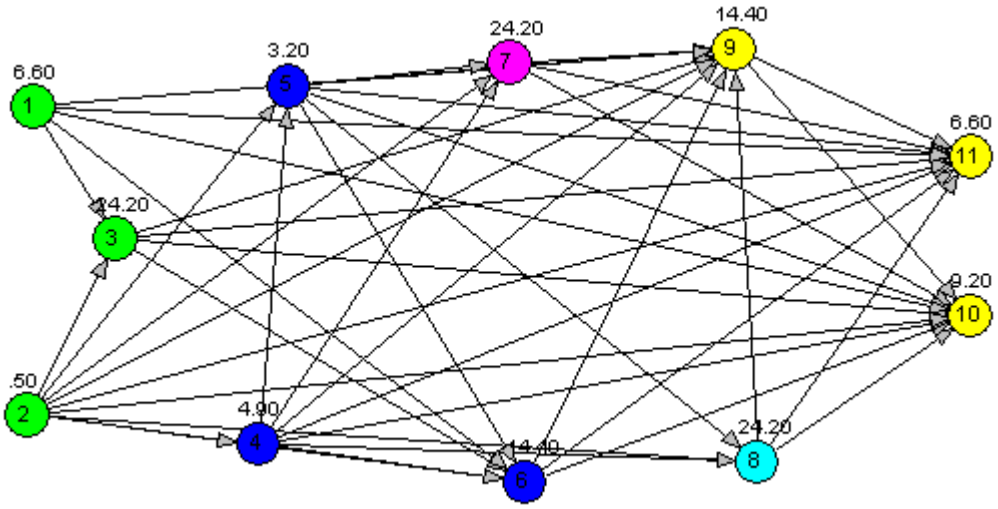


Figure 17: Brake Assembly precedence diagram (Flexible Line Balancing software)

It is interesting to note that the optimum assembly sequence in this case is:

1->2->3->5->4->6->7->8->9->10->11

Also, the task distribution at each workstation is shown in Figure 18.At workstation 1 task 1; task 2 and task 3 are performed. Similarly, the other tasks are distributed among the workstations as shown in Figure 18. The line balance efficiency, which is calculated using the neck time, is 84.6% as shown in Figure 18.

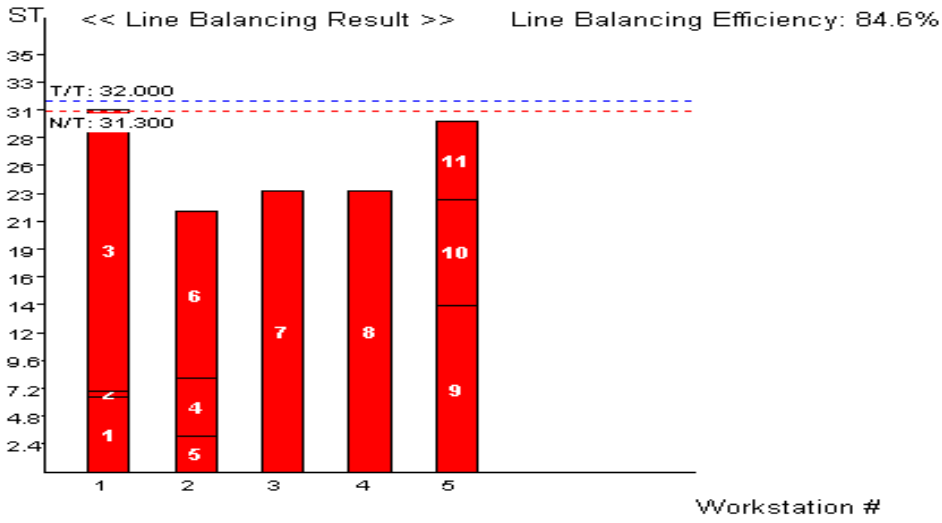


Figure 18: Line Balancing output of Brake Assembly (Flexible Line Balancing Software)

Thus, after the Line Balancing we obtain the following optimum outputs according to the takt time:

1. The number of workstations
2. Time at each workstation
3. Optimum sequence of the assembly
4. Distribution of task at each workstation.

Consider the Allen Screw for illustration. We have 6 feasible assembly sequences when we relax the precedence on Allen screw motor. Therefore, the Line Balancing result obtained from the Flexible Line Balancing software by relaxing the constraint on the Allen screw is shown in Figure 20. When the model is run considering the takt time of 32 seconds, the following Line Balancing output is obtained as shown in Figure 20.

- Number of workstations = 5
- Takt time = 32 seconds
- Neck time , i.e., workstation which takes the highest time = 30.2 seconds
- Number of operators = 5

It is interesting to note that the optimum assembly sequence in this case is

3->2->4->1->5->6->8->7->9->10->11

The precedence constraint diagram for the brake assembly with the relaxation of the precedence constraint on the Allen screw motor is shown in Figure 19.

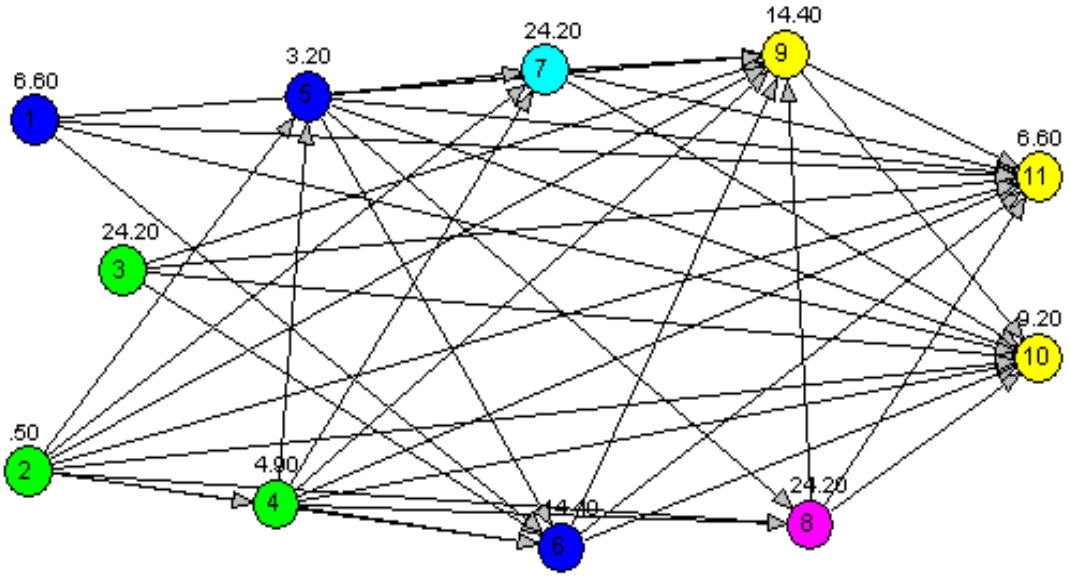


Figure 19: Precedence diagram relaxing precedence on Allen screw motor

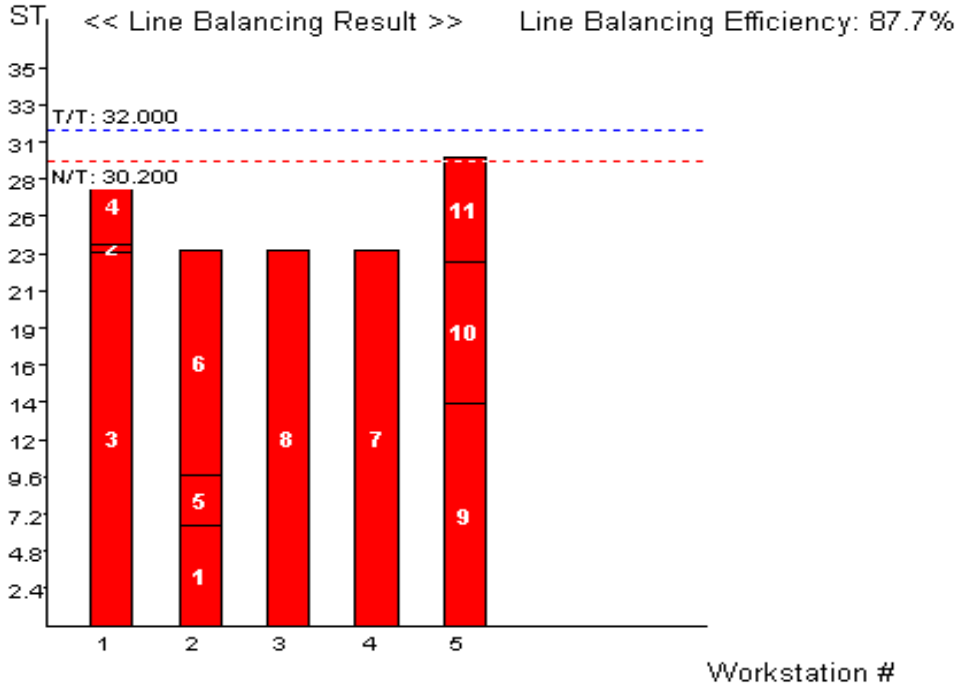


Figure 20: Line Balancing output relaxing precedence on Allen screw motor.

Here we have successfully relaxed the precedence constraint on the Allen screw motor and seen its effect on the Line Balancing result. Similarly, we now relax the precedence constraint on each component of the DFA analysis to obtain the metrics in order to identify the

components which have significant effect on Line Balancing and Cycle time. Thus, this procedure is iterated for each operation in the DFA analysis for generation of the metrics, which is explained in the next section.

The analysis is done using MS Excel. When the procedure for Line Balancing is iterated using the Flexible Line Balancing software the following data is obtained:

- Number of workstations for the given takt time of 32 seconds
- Neck time
- Takt time
- Assembly time at each workstation

This data is obtained for each of the iterations by systematically relaxing precedence constraint on one operation at a time. Now this data is used for calculation of two indices:

1. Cycle Time Index
2. Line Balance Index

The Table 14 shows the entire metrics for the brake assembly. The baseline run considers all components of the brake assembly with their precedence constraints this is the original design. Also, in this metrics you can see that after relaxing precedence constraints on each of the tasks, the time at each workstation changes accordingly.

Table 14: Brake Assembly Metrics

			workstation time in seconds							
			1	2	3	4	5	6	Cycle Time Index	Line Balancing Index
Baseline		ALL COMPONENTS i.e. original design	31.30	22.50	24.20	24.20	30.20		0.00	0.0604
relaxing all precedence on	1	Motor	31.30	22.50	24.20	24.20	30.20		0.00	0.0604
	2	Center Plate	31.30	22.50	24.20	24.20	30.20		0.00	0.0604
	3	Allen Screw	29.60	24.20	24.20	24.20	30.20		0.35	0.0540
	4	Flange	31.30	22.50	24.20	24.20	30.20		0.00	0.0604
	5	Electric Controls Box	31.30	22.50	24.20	24.20	30.20		0.00	0.0604
	6	Cylindrical Pegs	29.60	24.20	24.20	24.20	30.20		0.35	0.0540
	7	Allen ECB 1	29.60	27.40	30.80	28.80	15.80		0.16	0.0824
	8	Allen ECB 2	29.60	27.40	30.80	28.80	15.80		0.16	0.0824
	9	Head Pegs	29.60	24.20	24.20	24.20	30.20		0.35	0.0540
	10	Thread cover small	24.40	24.20	24.20	14.40	24.20	21.00	0.65	0.2236
	11	Thread cover Large	21.80	24.20	24.20	14.40	24.20	23.60	0.72	0.2226

After generating the metrics for the brake assembly we then plot a graph of the Line balance Index versus the Cycle time index as shown in Figure 21. From this graph the product designer can then choose which component will be redesigned. As shown in Figure 21, the Line balance indices are plotted on the Y axis, and the corresponding Cycle time Indices are on the X axis. Thus, each task is a point on the graph.

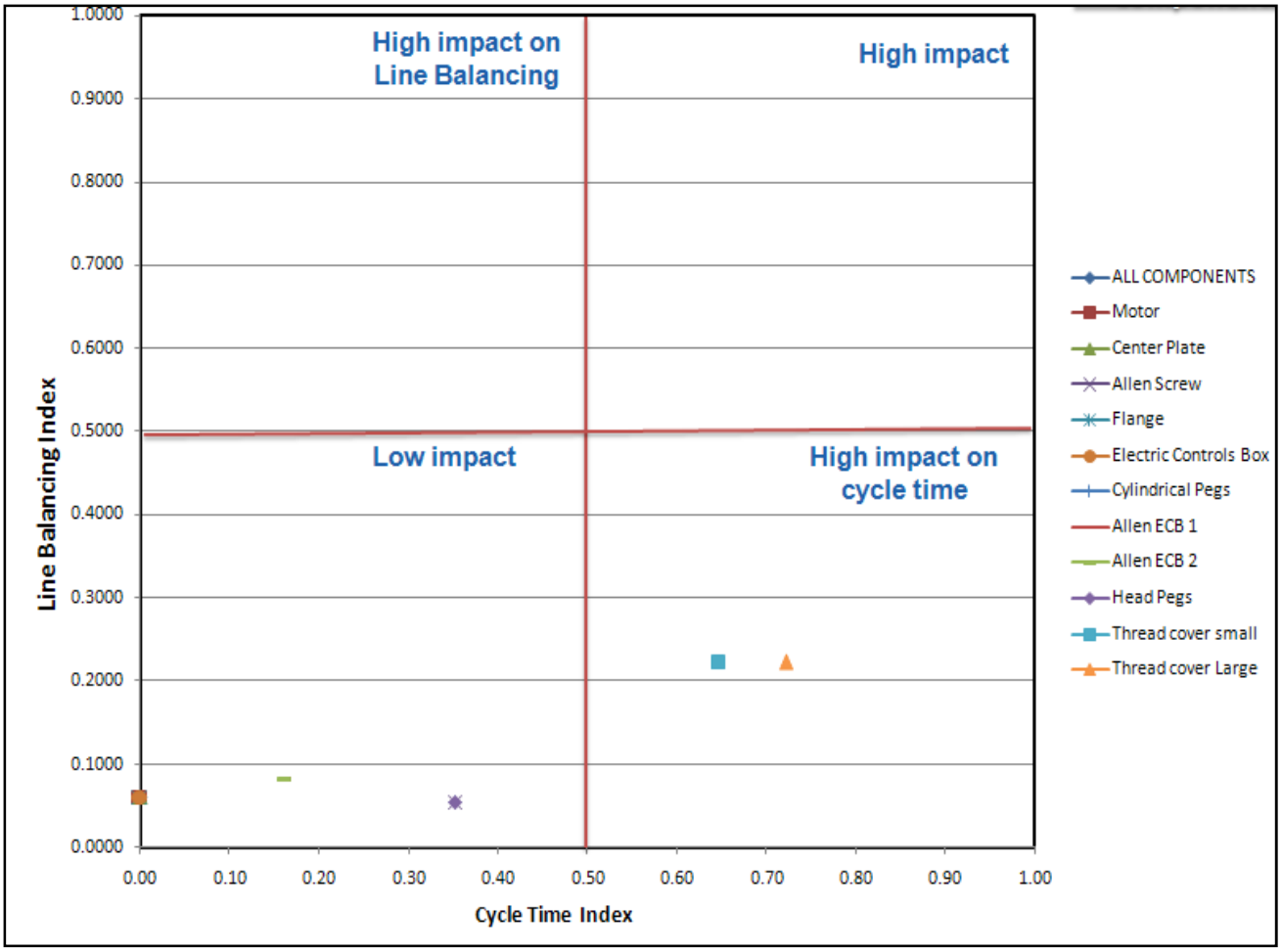


Figure 21: Line Balancing Index versus Cycle time Index for Brake assembly

From Figure 21 we see that thread cover large gives us the maximum reduction in cycle time whereas thread cover small affects the Line balance index the most. Thus, the decision to select a component for redesign depends on the product designer and the designer may choose from a task from any of the four regions. However the components or tasks in the High impact zone in Figure 21 must be considered the primary components for redesign as they have the most significant affect on the line balance and cycle time.

4.5. Step 5: Relaxing Precedence Relationships

The Line Balancing is done by relaxing all precedence's on each task. This gives a good observation on the potential components for redesign. As we further focus on the relaxing of the precedence constraints, the following questions arise:

- What are the interaction effects if we systematically relax constraints on various combinations of the task?
- What are the interaction effects of the precedence constraints if we consider various combinations of precedence on each task?
- What can we interpret from doing both combinations?

To answer these questions, we relax precedence on all the permutations of the tasks and then run the Line Balancing algorithm for each permutation, i.e., the Row Permutation. Also, we consider all permutations of the precedence on each task and then run the Line Balancing algorithm for each permutation, i.e., the Column Permutation. This is discussed in section 3.2.5.

These permutations were done using C++ programming. The C++ codes developed for this algorithm as well as the Row permutation and Column permutation are in Appendix B.

4.6. Step 6: Redesign Action

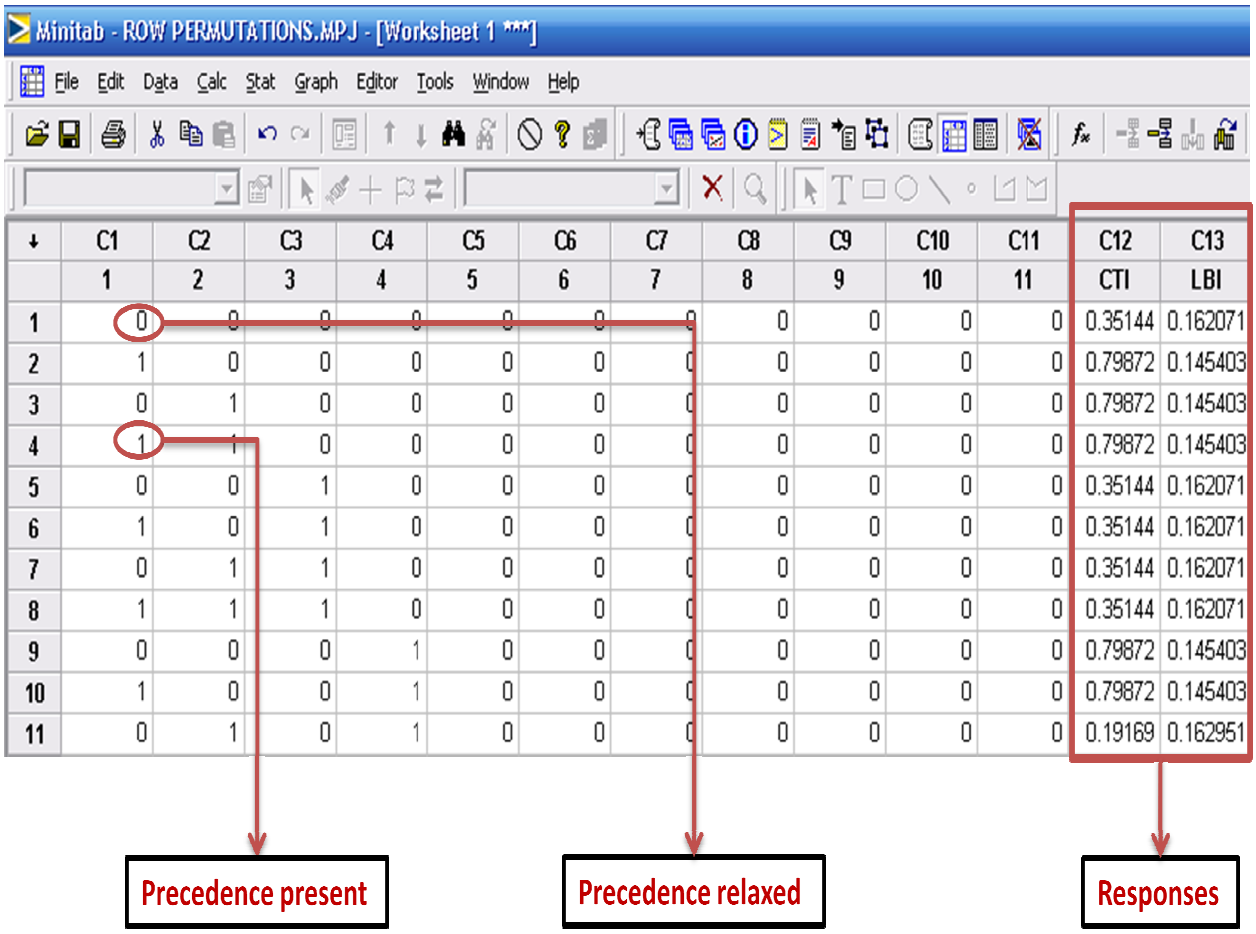
4.6.1. Analyzing the Data

The Row Permutation and the Column Permutation for the Brake assembly is done using C++ programming. The result is obtained in an MS Excel file for each case. As the Brake assembly has a total of 11 tasks, all combinations of these task result in 2^{11} combinations of the tasks and similarly 2^{11} combinations of the precedence constraints. Hence we have a total of 2048 observations for each permutation. Thus, for the brake assembly we now have a 2^{11} factorial experiment for each of the cases where:

1. Factors: 11 tasks of the Brake assembly
2. Responses: Line Balancing Index and Cycle time Index
3. Levels: 0 and 1, i.e., precedence relaxed and precedence present

This data is then analyzed in the statistical software Minitab to obtain the significant factors and their interaction for each case, i.e., the Row Permutation case and the Column Permutation case.

Figure 22 below shows the sample worksheet for the data analysis of the Row permutation. Similarly data is analyzed for the column permutation using Minitab.



	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13
	1	2	3	4	5	6	7	8	9	10	11	CTI	LBI
1	0	0	0	0	0	0	0	0	0	0	0	0.35144	0.162071
2	1	0	0	0	0	0	0	0	0	0	0	0.79872	0.145403
3	0	1	0	0	0	0	0	0	0	0	0	0.79872	0.145403
4	1	1	0	0	0	0	0	0	0	0	0	0.79872	0.145403
5	0	0	1	0	0	0	0	0	0	0	0	0.35144	0.162071
6	1	0	1	0	0	0	0	0	0	0	0	0.35144	0.162071
7	0	1	1	0	0	0	0	0	0	0	0	0.35144	0.162071
8	1	1	1	0	0	0	0	0	0	0	0	0.35144	0.162071
9	0	0	0	1	0	0	0	0	0	0	0	0.79872	0.145403
10	1	0	0	1	0	0	0	0	0	0	0	0.79872	0.145403
11	0	1	0	1	0	0	0	0	0	0	0	0.19169	0.162951

Figure 22: Analyzing Data using Minitab

We then analyze the factorial design using Minitab and plot the significant effects plot for both the cases, i.e., Row Permutation and Column Permutation. For each case we plot the significant effects plot for each of the two responses, i.e., Line Balancing Index and Cycle time index.

After the data analysis we now have four plots of significant effects as follows:

Row Permutation:

- Significant effects plot of Line balance index
- Significant effects plot of Cycle time index

Column Permutation

- Significant effects plot of Line balance index
- Significant effects plot of Cycle time index

Row Permutation: The significant effects plot of the response variable for row permutation helps identify the task and the interaction of the tasks that are significant.

The significant effects plot for the response of Line balance index is shown in Figure 23. We can see that the interactions that are marked red are significant. These include KL; F and J interactions, which are most significant.

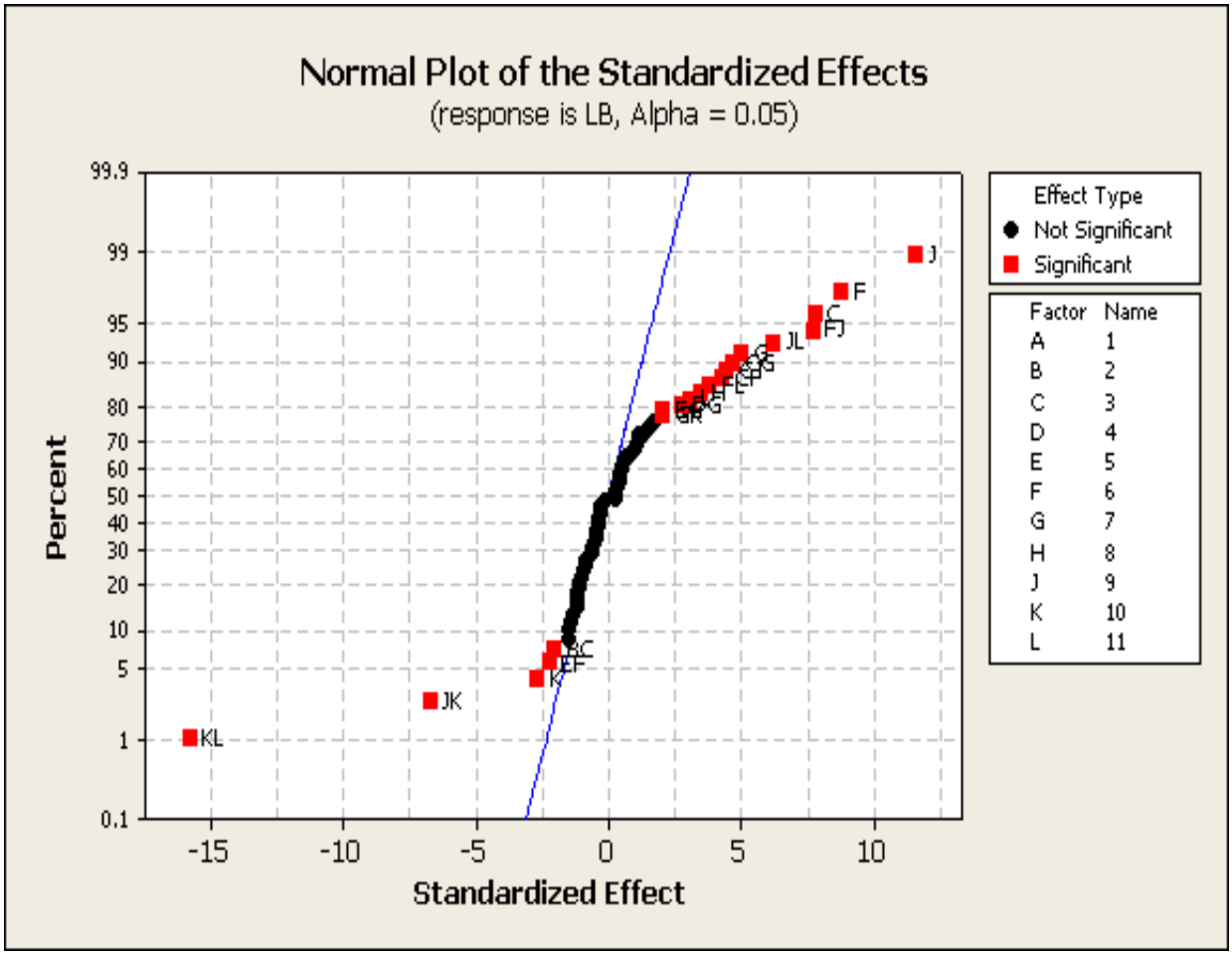


Figure 23: Significant effects plot of Line balance index for row permutation

Similarly, as shown in

Figure 24 for the response of the Cycle time index we can identify interactions like FJ; EJ; FJ and EK, which are most significant.

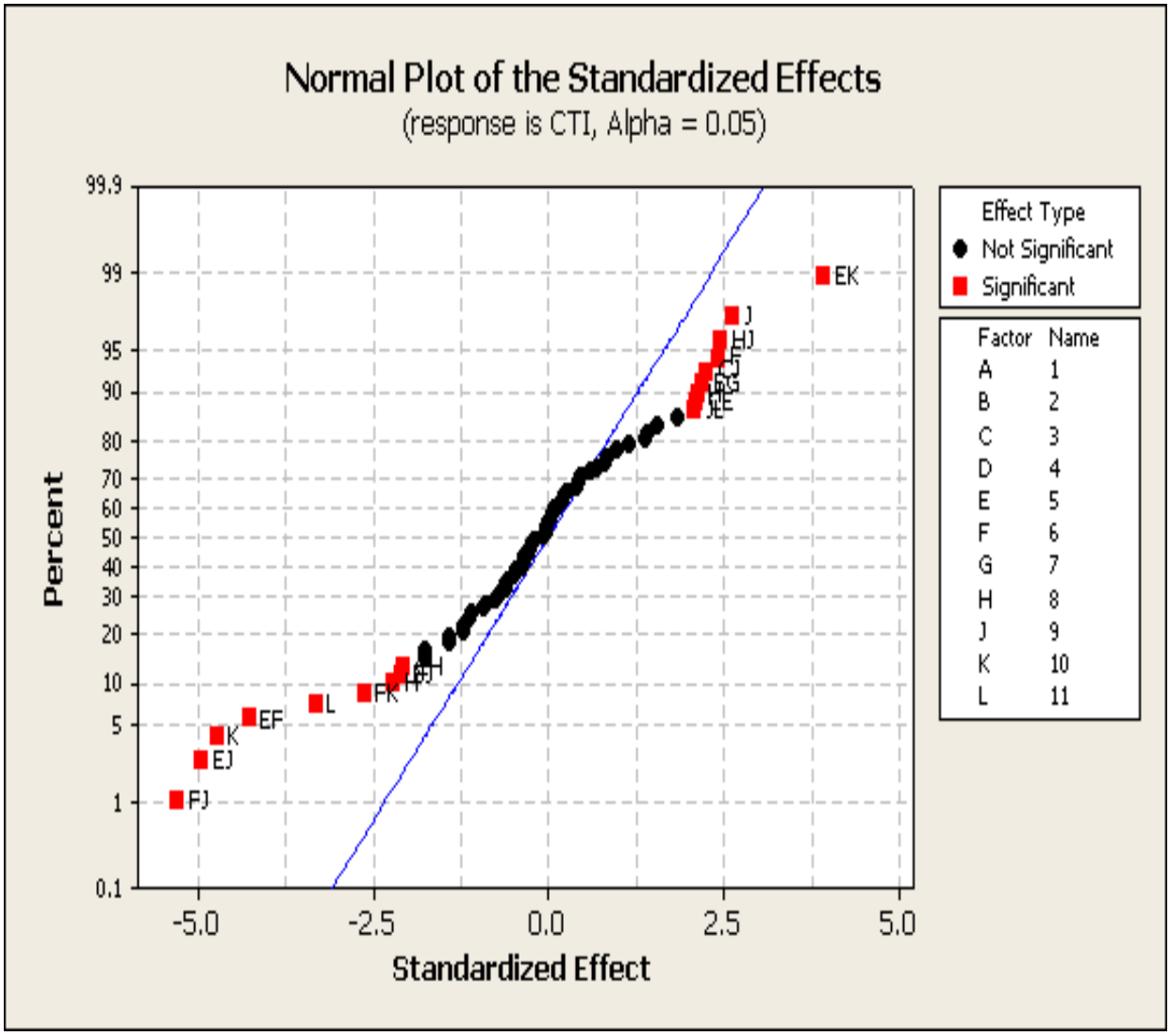


Figure 24: Significant effects plot of Cycle time index for row permutation

Column Permutation: The significant effects plot of the response variable for column permutation helps identify the precedence and the interaction of the precedence constraints that are significant.

The significant effects plot for the response of Line balance index is shown in Figure 25. We can see that the interactions that are marked red are significant. These include AC, A and C interactions which are most significant.

Similarly as shown in Figure 26 for the response of the Cycle time index we can identify interactions like AG; AH; C and A which are most significant.

The plots are as shown below:-

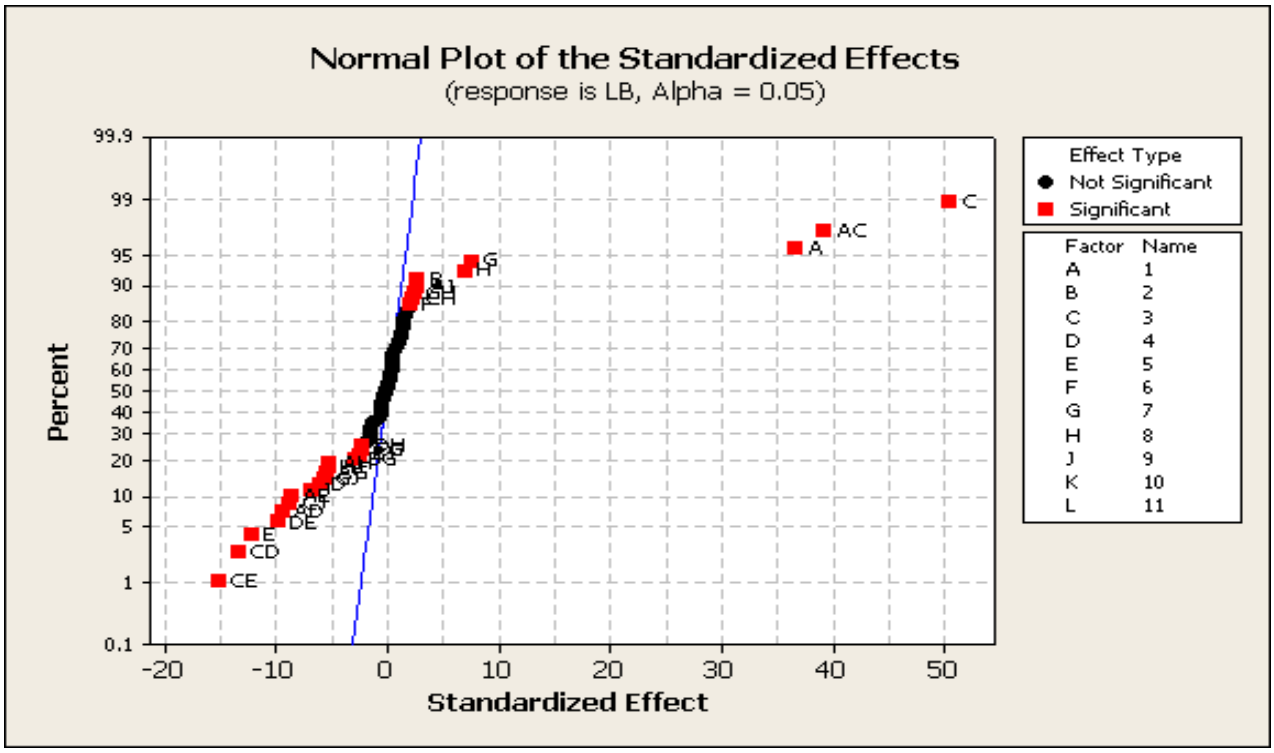


Figure 25: Significant effects plot of Line balance index for column permutation

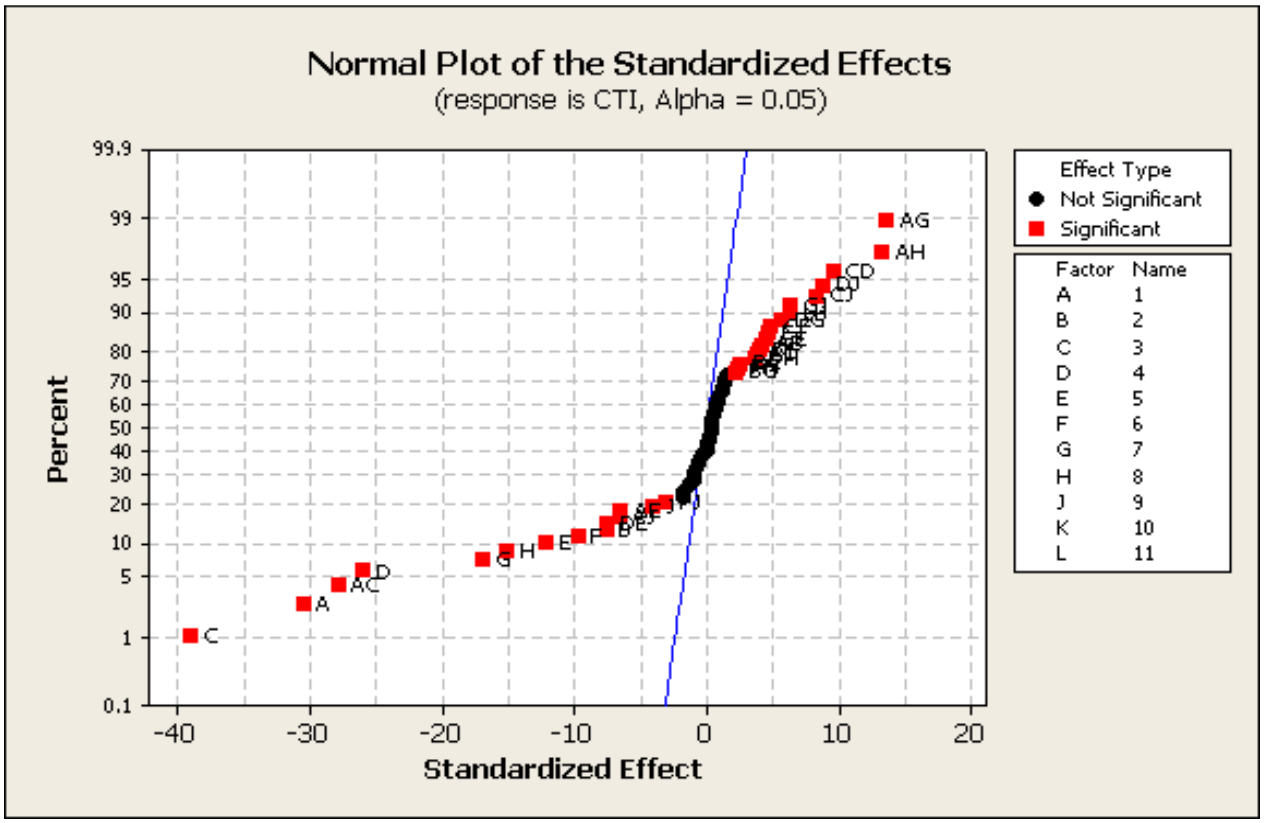


Figure 26: Significant effects plot of Line balance index for column permutation

4.6.2. Identifying Components for Redesign

The significant effects plot from Minitab indicates the following:

- The tasks that is significant.
- The interaction effects of these tasks that is significant.

Now to identify the components for redesign we have to plot the graph of Line balance Index versus Cycle time Index. This is achieved by the following:

- Take the coefficients of significant effects of Line balance index.
- Take the coefficients of significant effects of Cycle time index.
- Plot a graph in Excel using these coefficients where:
  X values are the cycle time index coefficients.
  Y values are the line balance index coefficients.

This graph is obtained for each case of the Row Permutation and the Column permutation.

- Row permutation

Figure 6 shows the graph of the Line balance index versus the Cycle time Index, i.e., LBI versus CTI for the row permutation.

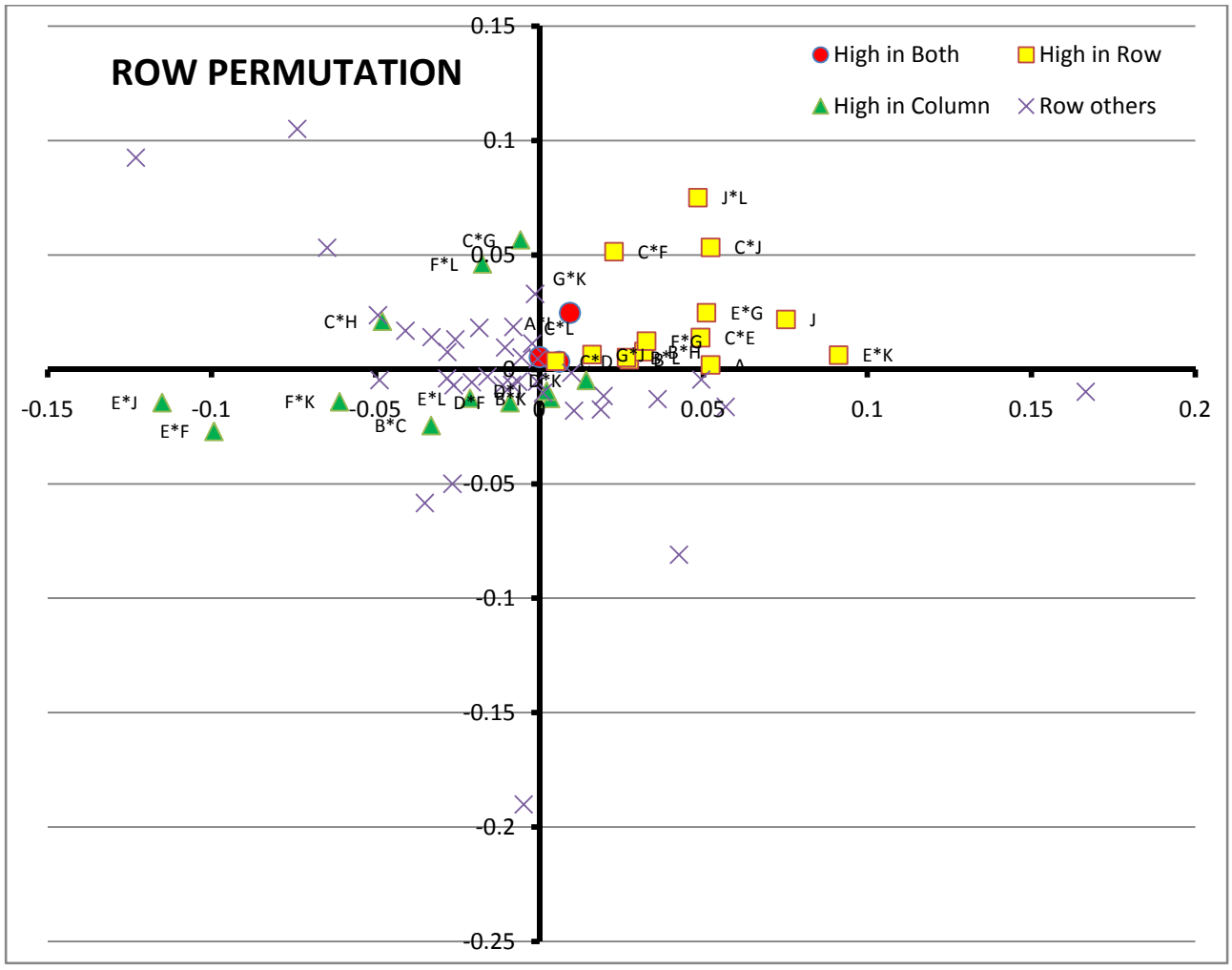


Figure 27: LBI versus CTI for row permutation

As there are numerous points on the graph let us now decrease the scale of the axis in order to clearly see the points lying in the positive region, i.e., the High impact region. These points, i.e., tasks affect the Line balance and cycle time the most.

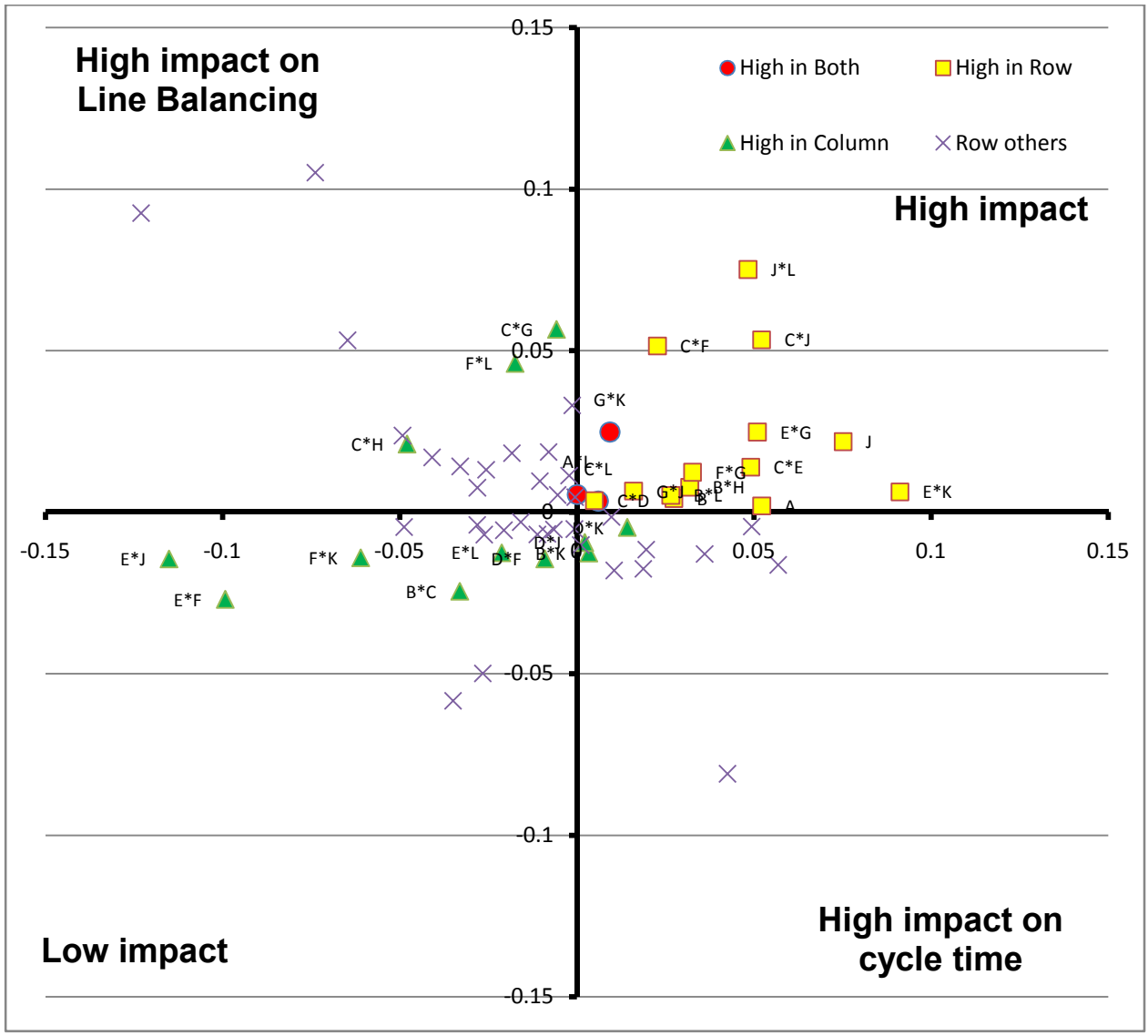


Figure 28: LBI versus CTI (ROW)

Figure 28 shows a clear view of the LBI versus CTI graph for the Row Permutation. Here you can see that the points marked red mean that these points have positive coefficients in both the Row Permutation and the Column Permutation. The points marked in yellow are the ones which have positive coefficients only in the Row Permutation. The green points are the ones which have positive coefficients only in the Column Permutation and are correspondingly here in the Row Permutation. The points marked as "x" are all the other points in the Row Permutation and are not significant.

- Column Permutation

Similar to the Row permutation, Figure 30 below shows the LBI versus CTI graph for the column permutation.

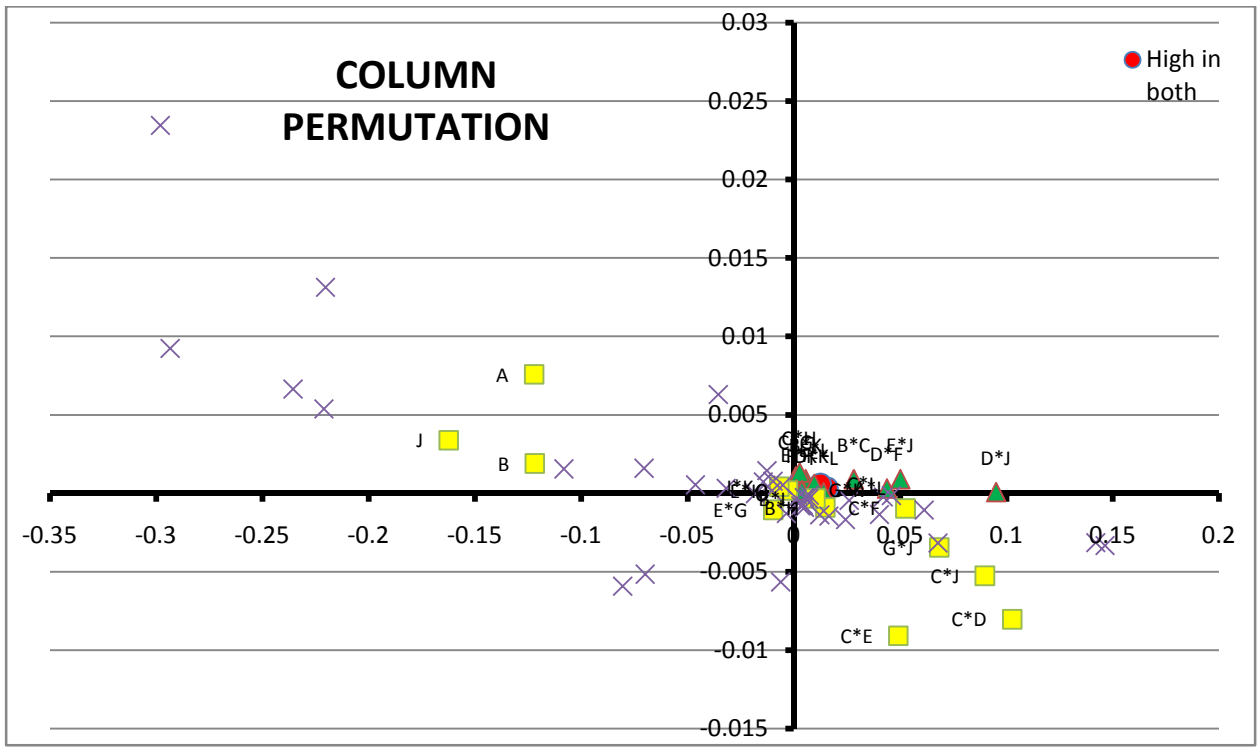


Figure 29: LBI versus CTI for column permutation

As there are numerous points on the graph, let us now decrease the scale of the axis in order to clearly see the points lying in the positive region, i.e., the High impact region. These points, i.e., tasks affect the Line balance and cycle time the most.

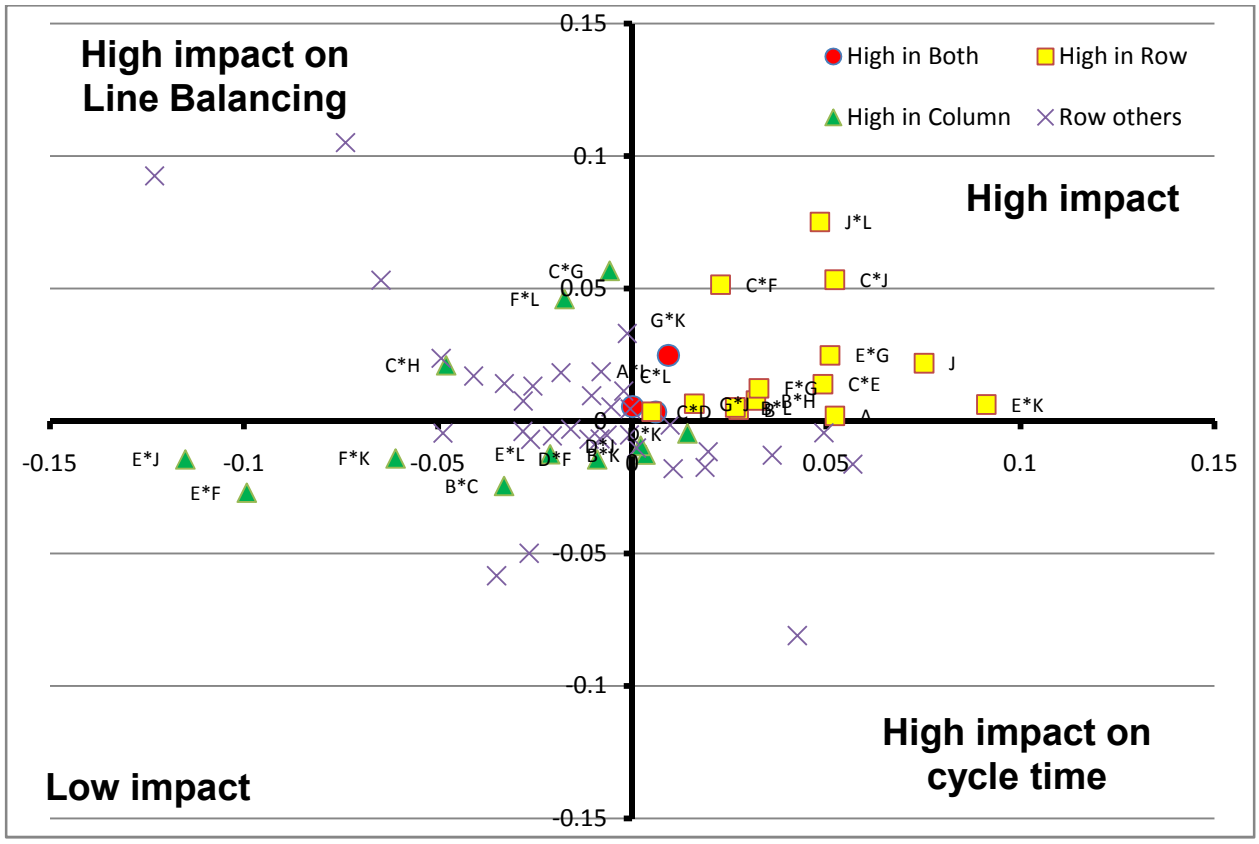


Figure 30: LBI versus CTI (column)

Figure 30 shows a clear view of the LBI versus CTI graph for the Column Permutation. Here you can see that the points marked red mean that these points have positive coefficients in both the Row Permutation and the Column Permutation. The green points are the ones which have positive coefficients only in the Column Permutation. The points marked in yellow are the ones which have positive coefficients only in the Row Permutation and are correspondingly here in the Column Permutation. The points marked as "x" are all the other points in the Column Permutation and are not significant.

Now we have identified the factors which have high impact on both Line Balancing and cycle time. In order for us to suggest design recommendation, we further follow the steps below to interpret the LBI versus CTI graph. The steps for interpretation are as follows:

1. Prioritize the parts.
2. Draw the input/ output diagram of the most critical part.
3. Analyze this diagram for under constrained and over constrained precedence.
4. Refer to action table for design recommendations.

Let us now see each of these steps in detail.

Step1: Prioritize the parts

For prioritizing the parts, use the following approach:

1. From the permutation take the points which are high in both the row and column permutation.
2. Compute their radius from the center of the LBI versus CTI graph.
3. Hence Radius = $\sqrt{(LBI)^2 + (CTI)^2}$
4. Compute the sum of the Row radius and Column radius.
5. The higher the sum the greater the distance. Therefore select the task with the highest sum for further analysis.

For the brake assembly the interaction G*K has the highest sum hence it is selected for further analysis as shown in the Table 15 below. Also, you can see the highest radius points in the row permutation and column permutation. E*K is the highest radius point in the row permutation and the interaction D*J is the highest radius point in the column permutation.

Table 15: Brake Assembly radius calculation

High in Both	Radius Row	Radius Column	Sum
A*L	5.477925123	15.66567963	21.1436
C*L	6.945215908	12.55504788	19.50026
G*K	26.49728046	4.061611253	30.55889

Highest	Task	Row Radius	Radius Column
In Row	E*K	91.41	
In Column	D*J		95.16

Step 2: Draw the input/ output diagram of the most critical part

From the network diagram as shown in Figure 17, we analyze the inputs and outputs of the task. For the brake assembly for the interaction G*K, we can see the input and output precedence constraints as shown in Figure 31. Let us now visualize the input/output diagram for the task G and the task K.

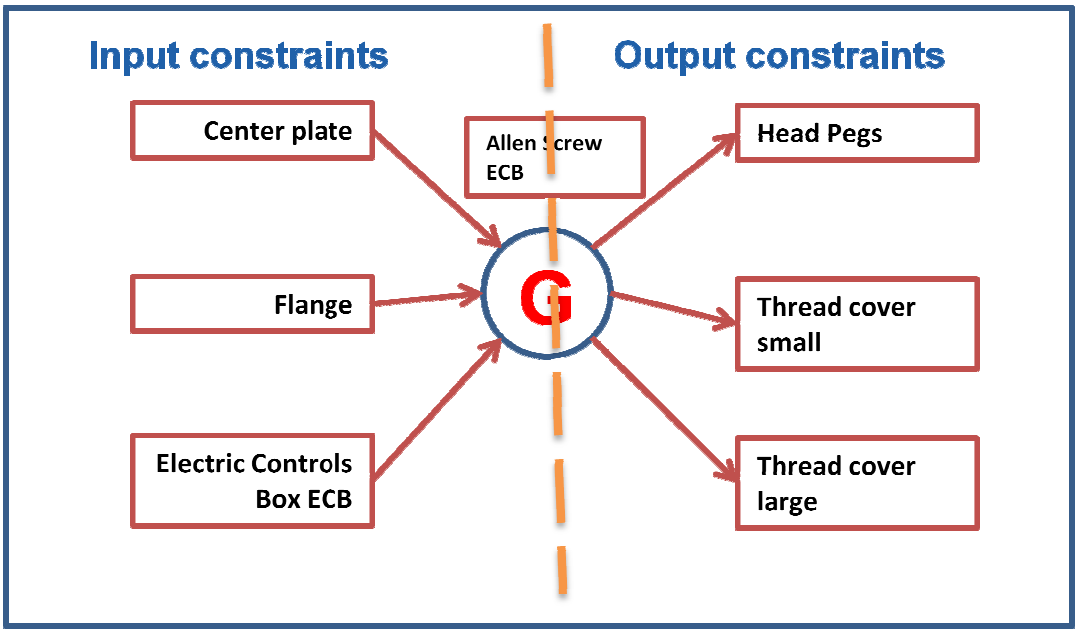


Figure 31: Input / Output precedence constraint for task "G"

As shown in Figure 31 the input precedence constraints for the task G, i.e., Allen screw ECB is center plate, Flange and the Electric Controls box (ECB). The output constraints of task G are head pegs, thread cover small and thread cover large.

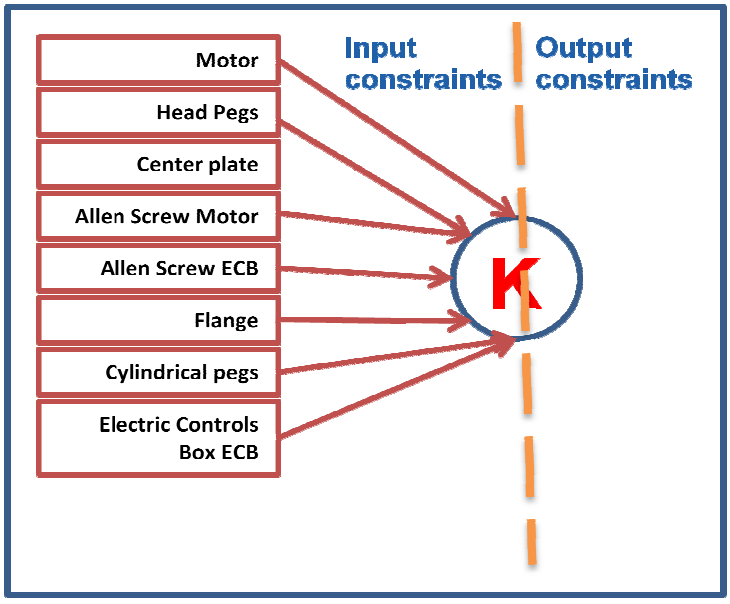


Figure 32: Input / Output precedence constraint for task "K"

Similarly in Figure 32 you can see that the task “K”, i.e., thread cover small has 9 tasks as input precedence constraints and no output constraints.

Step 3: Analyze this diagram for under constrained and over constrained precedence

After we draw the input/ output diagram we analyze each of the constraints in detail. Let us now take a look at the action table as shown in Table 11.

Now for the brake assembly, from Figure 31 and the action table above we analyze each of the precedence constraints on the task “G.” We observe that task G is causing over-constraint on the center plate, Flange and ECB. Hence referring to the action table it is necessary to redesign or eliminate task G, i.e., the Allen Screw ECB. Now eliminating “G”, i.e., the Allen screw ECB, we consider snap fit as the redesign action. Again going through the DFA procedure considering snap fit we get the new set of results. Thus, let us now compare the results from the DFA and Line Balancing before and after the new design. From this redesign we can compare the following results at the design stage of the product.

1. Total time from DFA
2. Line Balancing Efficiency
3. Number of workstations
4. Total Cycle time
5. Highest workstation time

The Table 16 shows the comparison of the new design to the original design of the brake assembly. The new DFA shows 42.6 seconds of reduction in the total assembly time when we eliminate the Allen screw ECB and use snap fit. Also, from the Line Balancing aspect of the design 2 workstations are reduced and efficiency increases by 11%. The total cycle time shows 62.6 seconds reduction as compared to the original design with a takt time of 32 seconds.

Table 16: Before and After (comparison of redesign)

Parameters		BEFORE (With Allen Screw ECB)	AFTER (New Design with Snap fit)	Improvement
DFA	**Total time from DFA**	132.4 seconds	89.8 seconds	42.6 seconds reduction
Line Balancing	**Number of workstations**	5	3	2 stations reduced
	Efficiency	84.60%	95.60%	11%
Cycle time	**Total Cycle time**	156.5 seconds	93.9 seconds	62.6 seconds reduction
	Highest Workstation time	31.3 seconds	31.3 seconds	no difference

Thus, by using the above procedure at the design stage we can help optimize Line Balancing and cycle time in the manufacturing stage. This helps in completing the loop and improving efficiency when implementing the new design on the real assembly line.

5. Discussion and Conclusion

In this discussion we will assess the improvement made towards the objective described in the motivation section. We would like to suggest some future research opportunities that could lead to a new DFX methodology.

In the methodology section, we developed a framework to link the DFA methodology to the manufacturing assembly line. This method was further implemented on the Brake Assembly as a preliminary case study. Also, during implementation many of the links were thoroughly reviewed and changes were made to improve the methodology. The brake assembly is the first

step towards linking the DFA to real assembly line performance as outlined in this work. The flow of this method is unique and simple to comprehend and implement.

First, the fishbone diagram gave us the precedence constraint which was then directly used with the DFA analysis as an input to the Line Balancing algorithm for row and column permutations. The COMSOAL algorithm was used here but there are other heuristics which can be explored for the purpose of Line Balancing.

Also, for relaxing the precedence constraints, we have used a brute force approach of considering all the combinations when relaxing precedence constraints this can be streamlined by using a different approach, for example, a weighted average method.

Next, the algorithm was programmed in C++, and the input to this program was done using a comma-separated values (CSV) file format. One problem is that the program consumes a lot of memory and processing power. The computation level for the brake assembly was fairly simple being 2048 permutations, but with change in the number of tasks from 11 to 21, the computational complexity increases and requires a longer time to simulate the row and column permutations for the algorithm. Also the amount of output data is very high to further calculate the indices. Thus, this method has limitations where the number of tasks computed can be anywhere between 0-15, which yields more stable results and it is easier to compute.

The output of the C++ program yields the cycle time and time at each workstation but the program can be tweaked further to output the indices directly in order to minimize time required to calculate these indices in a separate Excel file. Thus, in summary, the following objectives were fulfilled in this work:

1. We have successfully created a link between DFA and assembly line performance.
2. The model shown here and the steps create a link between DFA and assembly line performance.
3. We help the product designer to identify components for redesign from the Line balance index versus the Cycle time index graph to optimize cycle time and line balance according to takt time.

6. Suggestions for Future Work

A number of open problems must be solved to allow the development of a new DFX methodology in order to link the DFA to real assembly line performance. These problems suggest a variety of research directions that need to be pursued to make such a system feasible.

One such direction would be to apply this method to a new case study in order to enhance the functionality. We can take an older generation of a product, apply the methodology and then compare whether the design changes from this methodology are similar to the new generation of that particular product. For example, a complex case study like the one-time-use cameras can be used. Here we can use the methodology on generation 1 of the camera and see if the suggested design changes are similar to the one in the next generation. This is a good way to validate the method. Also, we may face new challenges with the increased number of tasks, and the program may be further enhanced to accommodate large amounts of data.

Another possibility would be to create multiple replicas of the DFA analysis for the same product. This will give us the range of task times as we can create multiple responses for the same tasks to run the design for experiments methodology. The variation may give us an accurate understanding of the design changes necessary.

Also, we might consider developing some software to simulate this DFX methodology. This software will directly output the LBI versus CTI graphs from a certain DFA analysis input. This will make it easier for the designer to analyze products faster and in a reliable manner as the human interaction with the data analysis is minimized.

Finally, in terms of applications of this framework, there are a plethora of possible products which can be analyzed. My particular interest would be to use the framework to study more complex products and enhance this methodology into a new DFX which will be helpful to the product designer.

7. References

Amen, M. 2001. Heuristic methods for cost-oriented assembly line balancing: A comparison on solution quality and computing time. International Journal of Production Economics, 69.

Baykasoglu, A. 2006. Multi-rule multi-objective simulated annealing algorithm for straight and U type assembly line balancing problems. Journal of Intelligent Manufacturing, 17: 217-232.

Bettles, I. Design for Manufacture & Assembly (DFMA) - The Boothroyd & Dewhurst Approach: 316-321. UK: The Smallpeice Trust.

Boothroyd, G., Dewhurst, N., Knight. 2002. Product Design for Manufacture and Assembly (second ed.), CRC Press.

Boothroyd, G., Dewhurst, N..; DFMA cost reduction tools; http://www.dfma.com/software/index.html.

Bralla, J. G. 1986. Handbook of Product Design for Manufacturing: McGraw-Hill Book Company, New York.

Bramall, D. G., Mckay, K. R., Rogers, B. C., Chapman, P., Cheung, W. M., & Maropoulos, P. G. 2003. Manufacturability Analysis Of Early Product. Int. J. Computer Integrated Manufacturing, 16(7-8): 501-508.

Caputo, A. and Pelagagge, P. 2008. Effects of Product Design on Assembly line performances: A concurrent Engineering approach. Industrial Management and Data Systems, 108(6): 726-749.

Chan, Vincent, Salustri, F. A., ; Design for Assembly; http://deed.ryerson.ca/~fil/t/dfmdfa.html, 2003.

Chiang, Wen-Chyuan, Urban, T. L., 2004. The stochastic U-line balancing problem: A heuristic procedure. European Journal of Operational Research, 175.

Chi-haur Wu, Y. X., Swee Mean Mok. 2007. Linking product design in CAD with assembly operations in CAM for virtual product assembly. Assembly Automation, 27: 309–323.

Curtis, Mark, Boothroyd, G, Dewhurst, . Design for Manufacture & Assembly.

Dimitriadis, S. G. 2005. Assembly line balancing and group working: A heuristic procedure for workers' groups operating on the same product and workstation. Computers & Operations Research, 33.

Falkenauer, E. 2000. Line Balancing in the Real World, International Conference on Product Lifecycle Management.

Gunasekaran, A. 1999. Agile manufacturing: A framework for research and development. International Journal of Production Economics, 62: 87-105.

Hinckley, M. Make No Mistake! An Outcome-Based Approach to Mistake-Proofing (1 ed.): Productivity Press.

Hopp, W. J. 2001. Factory physics: foundations of manufacturing management: Boston: Irwin/McGraw-Hill.

Ishii, Kosuke, Lee, B., December, 1995. Reverse Fishbone Diagram: A Tool In Aid Of Design For Product Retirement, ASME Design Technical Conference.

Lambert, A. J. D. 2004. Generation of assembly graphs by systematic analysis of assembly structures. European Journal of Operational Research: 932-951.

LG Electronics Inc., P. T. E. a. M. S.; Flexible Line Balancing V. 3 - An Exceptional Product for Balancing Assembly Lines; http://www.protech-ie.com/flb.htm.

Miles, B. 1988. Design for Assembly. Automotive Engineer, 13(3): 30-40.

Ohashi Toshijiro, I. M., Arimoto Shoji, Miyakawa Seii. 2002. Extended Assemblability Evaluation Method (AEM): Extended Quantitative Assembly Producibility Evaluation For Assembled Parts And Products. JSME International Journal, 45(2).

Pastor, R., C. A., A Duran, M Perez. 2002. Tabu search algorithms for an industrial multi-product and multi-objective assembly line balancing problem, with reduction of the task dispersion. Journal of the Operational Research Society, 53: 1317-1323.

Peeters, Marc, Degraeve, Z.,. 2006. An linear programming based lower bound for the simple assembly line balancing problem. European Journal of Operational Research.

Ron, G.; Design For Assembly; http://www.engineer.gvsu.edu/vac/.

Sly, Dave, Prem. G.; A Practical Approach to solving Multi-objective Line Balancing Problem; http://www.proplanner.com/Documents/Support/LB/LBTechnical.pdf.

Sprecher. 1999. A competitive branch-and-bound algorithm for the simple assembly line balancing problem. International Journal of Production Research, 37(8).

Stone, Robert B,. McAdams, D. A., Varghese J. Kayyalethekkel. 2003. A product architecture-based conceptual DFA technique. Design studies, 25: 301 - 325.

Swift, Ken, and Graham, J., ; The Designers' Sandpit; http://www.hull.ac.uk/MAPP/sandpit/index.html.

Whitney, D. E., 2004. MECHANICAL ASSEMBLIES Their Design, Manufacture and Role in Product Development. New York: OXFORD UNIVERSITY PRESS.

Xie, X.; Design for Manufacture and Assembly;
http://home.utah.edu/~u0324774/pdf/DFMA.pdf.

Yoosufani, Z. 1983. Effect of Part Symmetry on Manual Assembly times. Journal of Manufacturing Systems, 2(2): 189-195.

Appendix A:

Preliminary Analysis

The preliminary analysis of Toyota Antilock Brake System was done to compare the different DFA techniques and to analyze the results obtained. The following methods were used for comparison.

- Westinghouse Method (manual)
- Boothroyd Dewhurst Method (manual)
- Boothroyd Dewhurst DFMA Software Method

Toyota Antilock Brake System

Table 17: Parts of the ABS System

Part number	Part Name	Quantity	used for
1	Motor	1	for rotational movement to activate the system
2	Center Plate	1	regulates fluid to brake system lines
3	Flange	1	protection between side plate and center plate
4	Electric controls box	1	housing for electrical connectors
5	head pegs	3	to be inserted in bushes for mounting purposes
6	Allen screw	4	fasteners for fixing center plate to electric controls box
7	Allen screw motor	2	fasteners for fixing motor to center plate
8	thread cover small	2	for separation of center plate and side plate
9	cylindrical pegs	4	covering for threads (storage)
10	thread cover large	1	covering for threads (storage)
11	bushes	3	to absorbs shocks(preassembled with Center Plate)

Assembly Fishbone Diagram

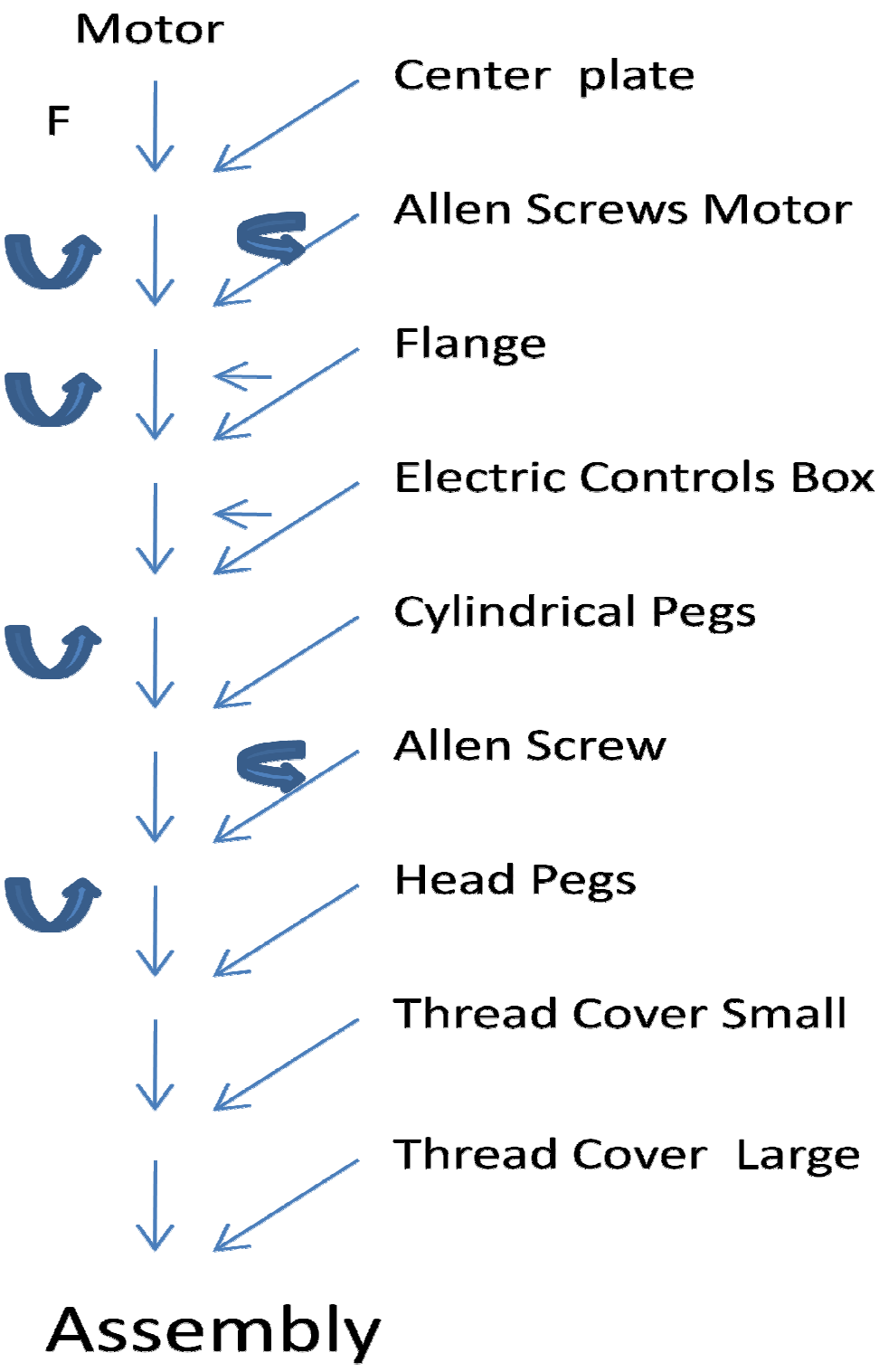


Figure 33: Assembly fishbone diagram of ABS system

Assembly Procedure

1) Assemble the Motor with the Center Plate.
2) Reorient the Center Plate and then secure the motor to the center plate by means of screws.
3) Reorient the assembly and introduce the Flange into the Center Plate.
4) Then assemble the Electric Controls Box (ECB) with the Center Plate.
5) Reorient the assembly and insert the cylindrical pegs between the center plate and ECB.
6) Now secure the ECB to the center plate using the Allen screws.
7) Reorient the assembly and insert the Head Pegs into the bushes.
8) Attach the 2 small thread covers to the thread holes.
9) Attach the large thread cover to the thread holes.

Note:

*The bush is pre-assembled by use of power assisted tools, i.e., it has not been considered for manual assembly.

*The weights were measured using Mettler Toledo weighing scale (PB3001S).

Westinghouse Method

Table 18: Westinghouse DFA

		Time Factors (seconds)														
		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
No.	Part/Operation Description	End-to-End Orientation	Rotational Alignment	Part Size	Part Thickness	Insertion Clearance	Insertion Direction	Insertion Condition	Fastening	Fastening Process	Handling Condition	Time/Each Operation (T_{ea})	Number of Repetitions (N_{rep})	Repetition Time ($K*L$) (T_{rep})	Insert Part (1 = Yes; 0 = No)	Eliminate Part (1=yes; 0 = No)
1	Motor	0.8	1.5	0.0	0.0	1.6	1.4	1.3	0.0	0.0	0.0	6.6	1	6.6	1	
2	Center Plate	0.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	1	0.5	1	
3	Flange	2.3	1.0	0.0	0.2	0.0	1.4	0.0	0.0	0.0	0.0	4.9	1	4.9		
4	Electric controls box	0.8	1.0	0.0	0.0	0.0	1.4	0.0	0.0	0.0	0.0	3.2	1	3.2	1	
5	head pegs	0.8	0.5	0,1	0.0	0.9	0.6	0.0	1.0	1.0	0.0	4.8	3	14.4		
6	allen screw	1.8	0.0	0.1	0.0	0.3	0.6	1.3	4.0	4.0	0.0	12.1	4	48.4		
7	allen screw motor	1.8	0.0	0.1	0.0	0.3	0.6	1.3	4.0	4.0	0.0	12.1	2	24.2		
8	thread cover small	1	1	0	0	0	1	0	1	1	0	4.6	2	9.2		
9	cylindrical pegs	1	0	0	0	0	1	1	0	0	0	3.6	4	14.4		
10	thread cover large	1	0	0	0	0	1	0	1	4	0	6.6	1	6.6		
													20	132.4	3	
													TOP	TAT	NUP	

Step 1: Draw the Assembly Sequence Diagram

Step 2: List Parts & operations in order (left column)

Step 3: Enter times from Estimated DFA Time Chart

Step 4: Sum time per part/oper. in column K

Enter no. of repetitions for each operation in col. L

Enter K*L in col. M

Step 5: Enter a 1 in col. N if a part was inserted during operation

Enter a 1 in col. O if part or operation can be eliminated

Step 6: Calculate Summary Statistics

Summary Statistics		
NUP	3	= number of unique parts (Sum of Column N)
TOP	20	= total number of operations (sum of Column L)
TAT	132.4	= total assembly time (sum of Column M)
NP	3	= number of parts = sumproduct(L,N)
T_{avg}	6.6	= avg time/operation = TAT/TOP
P_{min}	3.0	= min # parts = NP - sumproduct(L,N,O)
AR	0.05	= Assembly rating = 2.35 * NP /TAT
PE	1.00	= Part Effciency = Pmin/NP
C	84.40	= Assembly complexity = TAT - (2.4*TOP)
OR	2.76	= Operation difficulty rating = TAT/(2.4*TOP)

Boothroyd Dewhurst Method

Table 19: Boothroyd and Dewhurst DFA

Part number	Part Name	No. of Items, RP	Tool Acquire Time sec., TA	Handling code	Handling Time sec., TH	Insertion code	Insertion Time sec., TI	Total Time sec. TA+RP(TH+TI)	Minimum Part Count
1	Motor	1	0	30	1.95	14	4	5.95	1
2	Center Plate	1	0	O0	1.13	OO	0	1.13	1
	Reorient	1	0	O0	1.13	0	0	1.13	
7	Allen screw motor	2	2.9	11	1.8	10	3.7	13.9	
	Reorient	1	0	O0	1.13	0	0	1.13	
3	Flange	1	0	20	1.8	OO	1.5	3.3	
4	Electric controls box	1	0	30	1.95	OO	1.5	3.45	1
	Reorient	1	0	O0	1.13	0	0	1.13	
9	cylindrical pegs	4	0	O1	1.43	10	3.7	20.52	
6	Allen screw	4	2.9	11	1.8	10	3.7	24.9	
	Reorient	1	0	O0	1.13	0	0	1.13	
5	head pegs	3	0	O1	1.43	O5	3.3	14.19	
8	thread cover small	2	0	31	2.25	O4	1.8	8.1	
10	thread cover large	1	0	30	1.95	O5	3.3	5.25	
								105.21	3

The ABS Brake the total number of parts and subassemblies is 20 and there are 4 additional operations. The total assembly time is 105.21 seconds. The theoretical minimum number of parts is 3.

DFA index: $E_{ma} = N_{min} * t_a / t_{ma}$ where N_{min} is theoretical minimum number of parts, t_a is basic assembly time for one part and t_{ma} is estimated time to complete the assembly of the product.

Here,

$N_{min} = 3$; $t_a = 3$ seconds; $t_{ma} = 105.21$ seconds.

Thus, $E_{ma} = (3 \times 3) / 105.21 = 0.08 = 8\%$

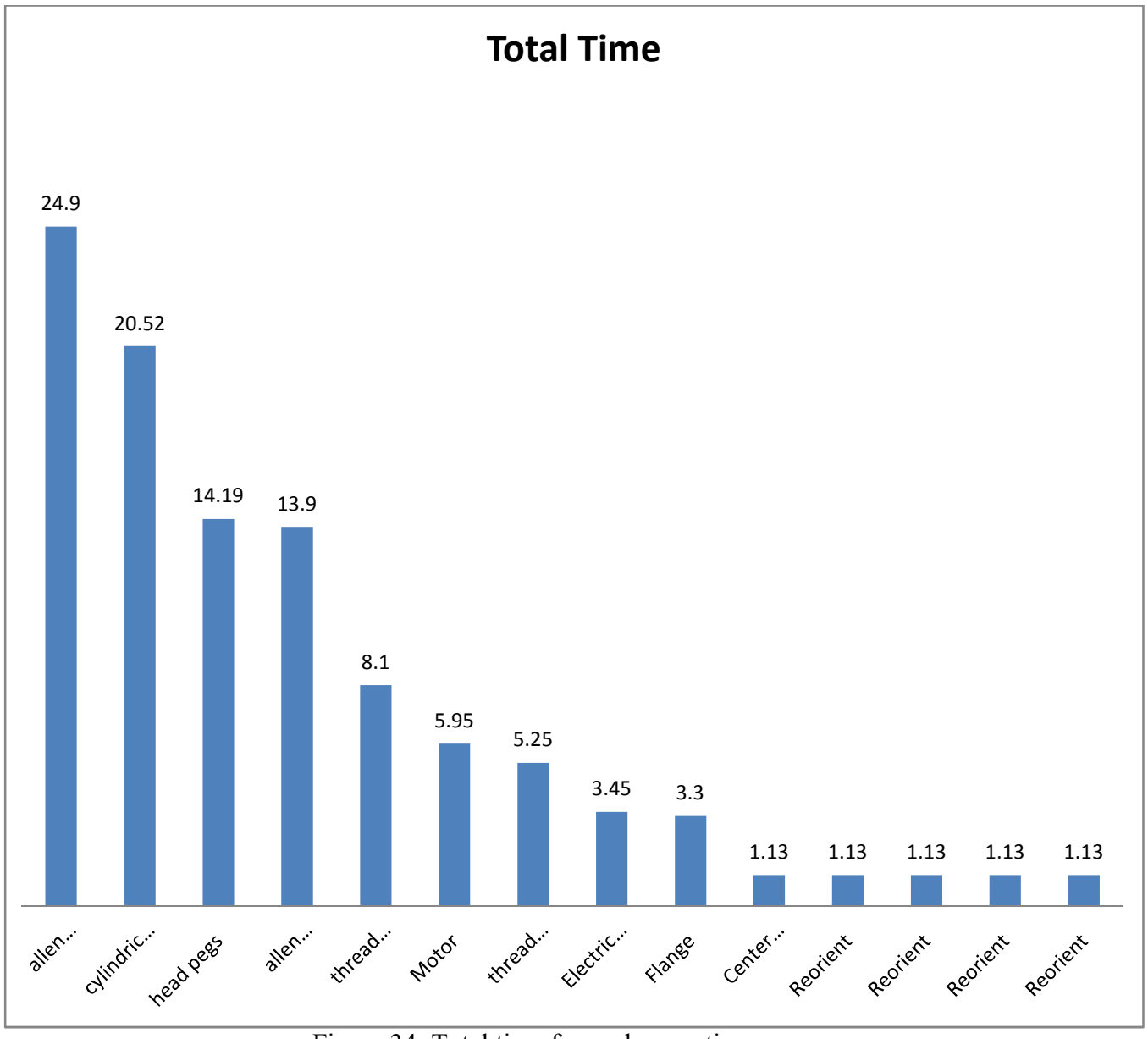


Figure 34: Total time for each operation

Table 20: DFMA Software

DESIGN FOR ASSEMBLY PRODUCT REVIEW TABLE REPORT
Rochester Institute of Technology, Mechanical Engineering
Toyota ABS [ABS.DFA]

Date of printing: 1:30pm Fri Mar 7, 2008

No.	Sub no. entry no.	Type	Name	Repeat count	Minimum items	Tool fetching time, s	Handling time, s	Insertion or op'n time, s	Total time, s
1	1.1	Part	Center Plate	1	1	0.00	1.95	1.50	3.45
2	1.2	Part	Motor	1	0	0.00	1.95	4.00	5.95
3	1.3	Oper	Reorientation	1	--	--	--	4.50	4.50
4	1.4	Part	Allen Screw Motor	2	0	2.90	2.10	14.30	35.70
5	1.5	Oper	Reorientation	1	--	--	--	4.50	4.50
6	1.6	Part	Flange	1	1	0.00	1.95	1.50	3.45
7	1.7	Part	Electric Controls Box	1	1	0.00	1.95	4.00	5.95
8	1.8	Oper	Reorientation	1	--	--	--	4.50	4.50
9	1.9	Part	Cylindrical Pegs	4	1	0.00	1.43	3.00	17.72
10	1.10	Part	Allen Screw	4	0	2.90	1.80	7.90	41.70
11	1.11	Oper	Reorientation	1	--	--	--	4.50	4.50
12	1.12	Part	Head Pegs	3	0	0.00	1.50	1.80	9.90
13	1.13	Part	Thread Cover Small	2	1	0.00	1.95	1.80	7.50
14	1.14	Part	Thread Cover Large	1	1	0.00	1.95	4.00	5.95

Appendix B:

COMSOAL Algorithm Codes

```
//#include "stdafx.h"
#include <iostream>
#include <stdio.h>
#include <set>
#include <list>
#include <map>
#include <string.h>
#include <fstream>
#include <sstream>
#include <iterator>
#include <stdlib.h>
#include <vector>
#include <windows.h>
#include <ctime>
#include <cstdlib>

#include "COMSOAL.h"

double stationtimecap;
bool isfirstseq;
seq_t bestseq;
int numattempts;
int currattempt;
map<int,task_t> taskmap;

double calcIdleTime(seq_t currseq) {
	double totaltasktime = 0;
	list<station_t> currstations = currseq.stations;
	list<station_t>::iterator stationiter = currstations.begin();
	while(stationiter != currstations.end()) {
		list<task_t> currtasks = (*stationiter).tasks;
		list<task_t>::iterator taskiter = currtasks.begin();
		while(taskiter != currtasks.end()) {
			totaltasktime += (*taskiter).time;
			taskiter++;
		}
		stationiter++;
	}
	double idletime = (stationtimecap * currseq.stations.size()) - totaltasktime;
	// printf("calcIdleTime: %f\n",idletime); // DEBUG
	return idletime;
```

```
}

void tryNewSeq() {
	// printf("################ TRY NEW SEQ ################\n"); // DEBUG

	/*
	cout << "PRINT ALL" << endl;
	map<int,task_t>::iterator taskmapiter = taskmap.begin();
	while(taskmapiter != taskmap.end()) {
		task_t tmptask = (*taskmapiter).second;
		int tmpid = tmptask.id;
		double tmptime = tmptask.time;
		set<int> tmpdeps = tmptask.deps;
		cout << "id: " << tmpid;
		cout << " time: " << tmptime;
		cout << " deps: [ ";
		set<int>::iterator tmpdepsiter = tmpdeps.begin();
		while(tmpdepsiter != tmpdeps.end()) {
			cout << (*tmpdepsiter) << ", ";
			tmpdepsiter++;
		}
		cout << "]" << endl;
		taskmapiter++;
	}
	*/

	// create current station
	station_t currstation;
	currstation.time = 0;
	currstation.id = 1;

	seq_t currseq;
	currseq.maxstationtime = 0;
	// currseq.stations.push_back(currstation); // add after mod
	// printf("currseq.size: %d\n",currseq.stations.size()); // DEBUG

	for(int n = 0; n < numattempts; n++) {
		printf(">> ATTEMPT %d out of %d\n",n+1,numattempts); // DEBUG
		extendSeq(taskmap,currstation,currseq);
		currattempt++;
	}

	// print stats
	printf("################ STATISTICS ################\n");
	double cycletime = (bestseq.maxstationtime * bestseq.stations.size());
	printf("cycletime: %f\n",cycletime);
```

```
	printf("numstations: %d\n",bestseq.stations.size());
	list<station_t>::iterator stationiter = bestseq.stations.begin();
	while(stationiter != bestseq.stations.end()) {
		printf("station: %d time: %f [ ",(*stationiter).id,(*stationiter).time);

		station_t stations = (*stationiter);
		list<task_t>::iterator taskiter = stations.tasks.begin();
		while(taskiter != stations.tasks.end()) {
			printf("%d, ",(*taskiter).id);
			taskiter++;
		}
		printf("]\n");

		stationiter++;
	}

}

void extendSeq(map<int,task_t> unassignedtaskmap, station_t currstation, seq_t currseq) {

	/*
	map<int,task_t>::iterator taskmapiter = taskmap.begin();
	while(taskmapiter != taskmap.end()) {
		task_t tmptask = (*taskmapiter).second;
		int tmpid = tmptask.id;
		double tmptime = tmptask.time;
		set<int> tmpdeps = tmptask.deps;
		cout << "id: " << tmpid;
		cout << " time: " << tmptime;
		cout << " deps: [ ";
		set<int>::iterator tmpdepsiter = tmpdeps.begin();
		while(tmpdepsiter != tmpdeps.end()) {
			cout << (*tmpdepsiter) << ", ";
			tmpdepsiter++;
		}
		cout << "]" << endl;
		taskmapiter++;
	}
	*/

	// printf(">> extendSeq\n"); // DEBUG
	// printf("currseq.size: %d\n",currseq.stations.size()); // DEBUG

	// CONSTRUCT FIT TASKS
	double currstationtime = currstation.time;
```

```
	vector<task_t> fittasks;
	map<int,task_t>::iterator unassignediter = unassignedtaskmap.begin();
	while(unassignediter != unassignedtaskmap.end()) { // for each unassigned task
		task_t othertask = (*unassignediter).second;
		// printf("unassigned task: %d\n",othertask.id); // DEBUG
		// Sleep(1000); // DEBUG

		set<int>deps = othertask.deps;
		set<int>::iterator depiter = deps.begin();
		bool found = false;
		while(depiter != deps.end() && !found) {
			int dep = (*depiter);

			// printf("dep: %d\n",dep); // DEBUG
			found = (unassignedtaskmap.find(dep) != unassignedtaskmap.end());
			// printf("dep: %d found in unassignedtasks: %d\n",dep,found); // DEBUG

			depiter++;
		}
		if(!found) {
			// printf("attempt to add to fittasks: %f + %f <=
%f\n",currstationtime,othertask.time,stationtimecap); // DEBUG
			if(currstationtime + othertask.time <= stationtimecap ) {
				// printf("added task %d\n",othertask.id); // DEBUG
				fittasks.push_back(othertask);
			}
		}
		unassignediter++;
   }

	// FIT TASKS POPULATED
	if(!fittasks.empty()) {

		/*
		printf("fittasks contains: [ "); // DEBUG
		vector<task_t>::iterator fititer = fittasks.begin();
		while(fititer != fittasks.end()) {
			printf("%d, ",(*fititer).id);
			fititer++;
		}
		printf("]\n");
		*/

		int numfittasks = fittasks.size();
		int randindex = rand() % numfittasks;
		task_t randtask = fittasks[randindex];
```

```
		// printf("randtask: %d\n",randtask.id); // DEBUG

		unassignedtaskmap.erase(randtask.id);
		currstation.tasks.push_back(randtask);
		currstation.time += randtask.time;

		/*
		printf("currstation: %d time: %f contains: [",currstation.id,currstation.time); //
DEBUG
		list<task_t>::iterator stationiter = currstation.tasks.begin();
		while(stationiter != currstation.tasks.end()) {
			printf("%d, ",(*stationiter).id);
			stationiter++;
		}
		printf("]\n");
		*/

		// printf("currstation.time > currseq.maxstationtime : %f > %f\n",
currstation.time,currseq.maxstationtime); // DEBUG
		if(currstation.time > currseq.maxstationtime) {
			currseq.maxstationtime = currstation.time;
		}

		if(!unassignedtaskmap.empty()) {
			/*
			printf("unassignedtaskmap IS NOT empty: [ "); // DEBUG
			map<int,task_t>::iterator taskiter = unassignedtaskmap.begin();
			while(taskiter != unassignedtaskmap.end()) {
				printf("%d, ",(*taskiter).second.id);
				taskiter++;
			}
			printf("]\n");
			*/

			extendSeq(unassignedtaskmap,currstation,currseq);
		} else {
			currseq.stations.push_back(currstation); // store station after mod
			// printf("unassignedtaskmap IS empty\n"); // DEBUG
			if(isfirstseq) {
				// printf(">> IS firstseq, so make bestseq\n"); // DEBUG
				isfirstseq = false;
				bestseq = currseq;
				bestseq.idletime = calcIdleTime(bestseq);
				// printf(">> %d\n",bestseq.stations.size()); // DEBUG
			} else {
```

```
                                // printf("currseq.maxstationtime: %f\n",currseq.maxstationtime);
// DEBUG
                                // printf("currseq.stations.size: %d\n",currseq.stations.size()); //
DEBUG
                                // printf("IS NOT firstseq, compare currseq to bestseq: %f < %f\n",
                                //      (currseq.maxstationtime * currseq.stations.size()),
                                //      (bestseq.maxstationtime * bestseq.stations.size())); //
DEBUG

                                // compare
                                if( (currseq.maxstationtime * currseq.stations.size()) <
                                        (bestseq.maxstationtime * bestseq.stations.size()) ) {
                                        // printf(">> make bestseq\n"); // DEBUG

                                        bestseq = currseq;
                                        bestseq.idletime = calcIdleTime(bestseq);
                                }
                        }
                }
        } else { // fittasks empty
                // printf("fittasks empty\n"); // DEBUG
                currseq.stations.push_back(currstation); // store station after mod

                // got total task time
                currseq.idletime = calcIdleTime(currseq);
                // printf(">> currseq.idletime > bestseq.idletime: %f >
%f\n",currseq.idletime,bestseq.idletime); // DEBUG
                if((currseq.idletime > bestseq.idletime) && !isfirstseq) {
                        // tryNewSeq();
                } else {
                        station_t newstation;
                        newstation.id = currseq.stations.size() + 1;
                        newstation.time = 0;
                        // printf(">> create new station: %d\n",newstation.id); // DEBUG
                        extendSeq(unassignedtaskmap,newstation,currseq);
                }
        }

}

seq_t run(int mynumattempts, int mystationtimecap, map<int,task_t> mytaskmap) {

        // DEBUG

        map<int,task_t>::iterator taskmapiter = mytaskmap.begin();
        while(taskmapiter != mytaskmap.end()) {
```

```
                task_t tmptask = (*taskmapiter).second;
                int tmpid = tmptask.id;
                double tmptime = tmptask.time;
                set<int> tmpdeps = tmptask.deps;
                cout << "id: " << tmpid;
                cout << " time: " << tmptime;
                cout << " deps: [ ";
                set<int>::iterator tmpdepsiter = tmpdeps.begin();
                while(tmpdepsiter != tmpdeps.end()) {
                        cout << (*tmpdepsiter) << ", ";
                        tmpdepsiter++;
                }
                cout << "]" << endl;
                taskmapiter++;
        }

        seq_t tmpseq;
        bestseq = tmpseq;

        numattempts = mynumattempts;
        stationtimecap = mystationtimecap;
        taskmap = mytaskmap;
        currattempt = 0;
        isfirstseq = true;
        srand((unsigned)time(0)); // rand seed
        tryNewSeq();
        return bestseq;
}

/*
int main(int argc, char* argv[]) {
        run(argc,argv);
        return 0;
}
*/
```

Row Permutation Codes

```
#include "COMSOAL.h"

int numtasks;
list<seq_t> results;

void genoutput() {

	char* filename = "output.csv";
	fstream fout(filename,ios::out);
	fout << "Task Deps,";
	for(int i = 0; i < numtasks; i++) {
		fout << (i+1) << ",";
	}

	fout << "Cycle Time,";
	for(int i = 0; i < numtasks; i++) {
		fout << (i+1) << ",";
	}
	fout << "\n";

	int count = 0;
	list<seq_t>::iterator seqiter = results.begin();
	while(seqiter != results.end()) {
		seq_t seq = *seqiter;

		// deps
		fout << "\"";
		for(int j = 0; j < numtasks; j++) {
			int id = j+1;
			bool hasdeps = (count >> j) & 1;
			if(hasdeps) {
				fout << id << ",";
			}
		}
		fout << "\",";

		for(int j = 0; j < numtasks; j++) {
			bool hasdeps = (count >> j) & 1;
			if(hasdeps) {
				fout << "1,";
			}else{
				fout << "0,";
			}
		}
```

```
                // DEBUG
                double cycletime = (seq.maxstationtime * seq.stations.size());
                fout << cycletime << ",";

                list<station_t>::iterator stationiter = seq.stations.begin();
                while(stationiter != seq.stations.end()) {
                        double time = (*stationiter).time;
                        fout << time << ",";
                        stationiter++;
                }

                fout << "\n";
                seqiter++;
                count++;
        }

        fout.close();
}

void permute(int numattempts, int stationtimecap, map<int,task_t> origtaskmap) {
        // run(numattempts, stationtimecap, origtaskmap);

        numtasks = origtaskmap.size();

        map<int,task_t> currtaskmap;
        int nump = (int)(pow(2,((double)numtasks)));
        for(int i = 0; i < nump; i++) { // DEBUG
                cout << "################# PERMUTATION " << (i+1) << " OF " << nump <<
" #################" << endl; // DEBUG
                currtaskmap = origtaskmap;

                // alter deps
                for(int j = 0; j < numtasks; j++) {
                        int id = j+1;
                        bool hasdeps = (i >> j) & 1;
                        cout << "task: " << (j+1) << " hasdeps: " << hasdeps << endl; // DEBUG
                        task_t t = (currtaskmap[id]);
                        if(!hasdeps) {
                                t.deps.clear();
                                currtaskmap.erase(id);
                                currtaskmap.insert(pair<int,task_t>(id,t));
                        }
                }

                results.push_back(run(numattempts, stationtimecap, currtaskmap));
```

```
                cout << endl << endl;
        }
}

int main(int argc, char* argv[]) {
        // run(argc, argv);

        // INITIALIZE
        if(argc != 4) {
                cout << "Improper cmd line args" << endl;
                exit(1);
        }

        map<int,task_t> taskmap;
        int numattempts;
        int stationtimecap;

        numattempts = atoi(argv[1]);
        stationtimecap = atof(argv[2]);
        char* filename = argv[3];
        string line;
        fstream fin(filename,ios::in);

        int tmpid;
        double tmptime;
        string tmp;

        char* cstr;
        char* token;
        getline(fin,line);
   while (getline(fin,line)) {
                // cout << line << endl;

                cstr = new char [line.size()+1];
                strcpy(cstr, line.c_str());

                // token = strtok(cstr,",");

                // while( token != NULL) {
                //      cout << "token: " << token << endl;
                //      token = strtok(NULL,",");
                // }

                token = strtok(cstr,",");
                tmpid = atoi(token);
                cout << "id: " << tmpid;
```

```
                token = strtok(NULL,",");
                tmptime = atof(token);
                cout << " time: " << tmptime;

                int tmpdep;
                set<int> tmpdeps;
                token = strtok(NULL,",");
                // cout << "tmp0: " << token << endl; // DEBUG
                while(token != NULL) {
                        tmpdep = atoi(token);
                        if(tmpdep == 0) {
                                // cout << "tmp1: " << token << endl; // DEBUG
                                string str(token);
                                strcpy(token, (str.substr(1,str.size())).c_str());
                                tmpdep = atoi(token);
                                // cout << " qdep: " << dep << endl;
                        } else {
                                // cout << " dep: " << dep << endl;
                        }
                        tmpdeps.insert(tmpdep);
                        token = strtok(NULL,",");
                }

                // DEBUG
                cout << " deps: [ ";
                set<int>::iterator tmpdepsiter = tmpdeps.begin();
                while(tmpdepsiter != tmpdeps.end()) {
                        cout << (*tmpdepsiter) << ", ";
                        tmpdepsiter++;
                }
                cout << "]" << endl;

                task_t t;
                t.id = tmpid;
                t.time = tmptime;
                t.deps = tmpdeps;
                taskmap.insert(pair<int,task_t>(tmpid,t));
                delete cstr;
        }
        fin.close();

        permute(numattempts, stationtimecap, taskmap);
        genoutput();
}
```

Column Permutation Codes

```
#include "COMSOAL.h"

int numtasks;
list<seq_t> results;

void genoutput() {

	char* filename = "output.csv";
	fstream fout(filename,ios::out);
	fout << "Task Deps,";
	for(int i = 0; i < numtasks; i++) {
		fout << (i+1) << ",";
	}

	fout << "Cycle Time,";
	for(int i = 0; i < numtasks; i++) {
		fout << (i+1) << ",";
	}
	fout << "\n";

	int count = 0;
	list<seq_t>::iterator seqiter = results.begin();
	while(seqiter != results.end()) {
		seq_t seq = *seqiter;

		// deps
		fout << "\"";
		for(int j = 0; j < numtasks; j++) {
			int id = j+1;
			bool hasdeps = (count >> j) & 1;
			if(hasdeps) {
				fout << id << ",";
			}
		}
		fout << "\",";

		for(int j = 0; j < numtasks; j++) {
			bool hasdeps = (count >> j) & 1;
			if(hasdeps) {
				fout << "1,";
			}else{
				fout << "0,";
			}
		}
```

```
                // DEBUG
                double cycletime = (seq.maxstationtime * seq.stations.size());
                fout << cycletime << ",";

                list<station_t>::iterator stationiter = seq.stations.begin();
                while(stationiter != seq.stations.end()) {
                        double time = (*stationiter).time;
                        fout << time << ",";
                        stationiter++;
                }

                fout << "\n";
                seqiter++;
                count++;
        }

        fout.close();
}

void permute(int numattempts, int stationtimecap, map<int,task_t> origtaskmap) {
        // run(numattempts, stationtimecap, origtaskmap);

        numtasks = origtaskmap.size();

        map<int,task_t> currtaskmap;
        int nump = (int)(pow(2,((double)numtasks)));
        for(int i = 0; i < nump; i++) { // DEBUG
                cout << "################# PERMUTATION " << (i+1) << " OF " << nump <<
" #################" << endl; // DEBUG
                currtaskmap = origtaskmap;

                // alter deps
                for(int j = 1; j < numtasks+1; j++) {

                        // cout << "task: " << j << endl; // DEBUG
                        task_t t = currtaskmap[j];

                        set<int> tmpdeps(t.deps);
                        t.deps.clear();
                        set<int>::iterator depiter = tmpdeps.begin();
                        while(depiter != tmpdeps.end()) {
                                int dep = (*depiter);
                                bool hasdep = (i >> (dep-1)) & 1;
                                if(hasdep) {
                                        t.deps.insert(dep);
```

```
                        }
                        // cout << "dep: " << dep << " : " << hasdep << endl; // DEBUG
                        depiter++;
                    }

                    currtaskmap.erase(j);
                    currtaskmap.insert(pair<int,task_t>(j,t));
                }

                results.push_back(run(numattempts, stationtimecap, currtaskmap));
                cout << endl << endl;
        }
}

int main(int argc, char* argv[]) {
        // run(argc, argv);

        // INITIALIZE
        if(argc != 4) {
                cout << "Improper cmd line args" << endl;
                exit(1);
        }

        map<int,task_t> taskmap;
        int numattempts;
        int stationtimecap;

        numattempts = atoi(argv[1]);
        stationtimecap = atof(argv[2]);
        char* filename = argv[3];
        string line;
        fstream fin(filename,ios::in);

        int tmpid;
        double tmptime;
        string tmp;

        char* cstr;
        char* token;
        getline(fin,line);
   while (getline(fin,line)) {
                // cout << line << endl;

                cstr = new char [line.size()+1];
                strcpy(cstr, line.c_str());
```

```
// token = strtok(cstr,",");

// while( token != NULL) {
//         cout << "token: " << token << endl;
//         token = strtok(NULL,",");
// }

token = strtok(cstr,",");
tmpid = atoi(token);
cout << "id: " << tmpid;

token = strtok(NULL,",");
tmptime = atof(token);
cout << " time: " << tmptime;

int tmpdep;
set<int> tmpdeps;
token = strtok(NULL,",");
// cout << "tmp0: " << token << endl; // DEBUG
while(token != NULL) {
        tmpdep = atoi(token);
        if(tmpdep == 0) {
                // cout << "tmp1: " << token << endl; // DEBUG
                string str(token);
                strcpy(token, (str.substr(1,str.size())).c_str());
                tmpdep = atoi(token);
                // cout << " qdep: " << dep << endl;
        } else {
                // cout << " dep: " << dep << endl;
        }
        tmpdeps.insert(tmpdep);
        token = strtok(NULL,",");
}

// DEBUG
cout << " deps: [ ";
set<int>::iterator tmpdepsiter = tmpdeps.begin();
while(tmpdepsiter != tmpdeps.end()) {
        cout << (*tmpdepsiter) << ", ";
        tmpdepsiter++;
}
cout << "]" << endl;

task_t t;
t.id = tmpid;
t.time = tmptime;
```

```
                t.deps = tmpdeps;
                taskmap.insert(pair<int,task_t>(tmpid,t));
                delete cstr;
    }
        fin.close();

        cout << endl;
        permute(numattempts, stationtimecap, taskmap);
        genoutput();
}
```