Brigham Young University

BYU ScholarsArchive

2009-10-06

Automated Data Import and Revision Management in a Product Lifecycle Management Environment

Brad Walton Brooks
Brigham Young University - Provo

Automated Data Import and Revision Management

in a Product Lifecycle Management Environment

Brad Brooks

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

C. Greg Jensen, Chair
Spencer P. Magleby
Jordan J. Cox

Department of Mechanical Engineering

Brigham Young University

December 2009

ABSTRACT

Automated Data Import and Revision Management

in a Product Lifecycle Management Environment

Brad Brooks

Department of Mechanical Engineering

Master of Science

A method has been created that addresses the issues that prohibit the conversion of product artifacts into a secure, efficient and reliable Product Lifecycle Management (PLM) environment. These issues include automatic import of data into a PLM system and revision control of such data. A test case is shown which specifically addresses these issues as they pertain to the management of both legacy and new engineering data in a PLM system.

ACKNOWLEDGMENTS

TABLE OF CONTENTS

LIST OF TABLES

x

LIST OF FIGURES

1 Introduction

In 2006, Airbus announced its second delay for deliveries of the highly anticipated Airbus A380 airliner. The delay caused several cancellations for orders, reducing the amount of planned deliveries from 120 to fewer than 80 aircraft. With each aircraft valued at over 300 Million US dollars, the losses amounted to over 12 billion dollars. [35] As a result, Boeing was able to convince Airbus Customers to order their smaller 787 to supplant the role that the A380 would have otherwise filled. Airbus in turn suffered a great loss which had a large impact on the company and its employees.

The delay was attributed to the highly complex wiring configuration of over 100,000 wires which, when stretched out, would span for over 330 miles. Additionally, not all sites working on the wiring harness were using the same design software. Some facilities in Europe which were working on the wiring harness were using different software than the main design sites in France and Great Britain. This resulted in overall configuration management problems. Required design criteria, such as the bend radii, were not all transferred between design and manufacturing facilities. This lack of communication regarding the engineering data communication resulted in an electrical harness that was several feet short of the required length. As a result, the wiring had to be completely redesigned, causing a large delay in product deliveries.

It is astounding to think that a simple error in managing engineering data caused a 12 billion dollar problem in this example alone. The landscape of engineering design has changed drastically over the past few decades. As engineering designs, such as the A380, increase in complexity, the tools used to manage the corresponding design data must continually evolve to handle this increased complexity. For this reason, special tools have been developed which aid companies in dealing with these issues of engineering data management.

1.1   Problem Statement

Product data management (PDM) systems came about in the early 1980's primarily to archive production information which generally included only final drawings and not the supporting design artifacts. This pattern continued for many years. In the late 1990's when various innovations led to more sophisticated systems which would control more design artifacts even including product-related marketing and financial information. These newer, further-reaching systems have earned the name of "Product Lifecycle Management" (PLM). The enterprise-wide benefits of PLM have resulted in its increased popularity.

Despite the rapid advancements in PLM system technology, many companies—particularly the larger, more established firms—find themselves struggling to adapt their cultures to take advantage of the new capabilities. They purchase and use the new systems, but typically they are deployed and used in the same way that their old PDM systems were used. Using the new systems is difficult for established companies because:

- PDM software is changing too fast for them to keep up

- How to use and adapt the new functionality is not clear

- Migrating legacy data is tedious because there is a greater amount of information about files than was previously unavailable

Because of these issues, the older, well-established companies are finding it difficult to remain competitive and as a result are struggling due to a lack of engineering innovation in the area of data management. Current methods to manage data, as it is imported into PLM, lack a solution which is secure, efficient, and reliable. To date, there has been no research addressing and resolving these issues in an effective manner.

1.2 Thesis Objective

This thesis will develop a method which addresses the issues that prohibit a secure, efficient, and reliable import and management of data into PLM. In order to develop such a method, this thesis will attempt to answer the following questions.

- Can an automated method be developed to import legacy data, including files and metadata?

- Can this automated method be run with a single command to save time over current best practices? If so, how much time does this save?

- Is there a way to improve reliability of the import by performing real-time error checking?

After these questions have been addressed, the results of the proposed method will be shown demonstrating the improvements in efficiency, reliability, and security.

1.3 Delimitation of the Problem

This study will be done in a general way and then applied to a common PLM system called Teamcenter to demonstrate that the study can be performed in the most widely used PLM software. A user should be able to apply this study to other PLM, systems such as ENOVIA or WINDCHILL. Any code or pseudo code will be done in the C programming language.

A test case will be performed based on real data provided by BE aerospace. Design attributes associated with this product data will be taken from data logs contained in several spreadsheets. These attributes will be associated with data such as engineering reports, supplier information, and test documentation. This research will not deal with CAD, CAE, or other forms of engineering data. The majority of the files will be in PDF form as it allows for electronic signatures. Results will be generalized as the data is proprietary.

1.4 Proposed Benefits

The method shown in this thesis will indicate an improvement in data security. Data which is managed outside of PLM is susceptible to operations which violate the artifacts security. Security vulnerabilities can occur when users accidentally delete or move product files. PLM increases security by handling user rights and revision control of product data.

Another benefit of this research will be improved reliability. By passing product data into PLM in an automated fashion, human interaction is eliminated and product data integrity is better maintained.

The final proposed benefit of this research will be an increase in data import efficiency. The new process will show that the time to import data can be reduced by over 30 times compared to previous methods. These time savings and improvements in efficiency will be discussed in the Results Chapter.

2 Literature Review

Recent advancements in CAx technologies have paved the way for faster, more efficient engineering. Companies have reported lower design costs and increased engineering productivity due to advancements in the arena of CAx. [10] The tool which has evolved, more than any of the CAx tools in the past decade, is PLM. Advancements in PLM have driven the concept of product design to a new level by allowing data to be stored more securely while allowing permitting new forms of collaborative design. Such advancements have redefined the idea now considered to be PLM.

Before one can fully appreciate and understand the methods involved with improving PLM software, it is necessary to lay a foundation in this area of research. This foundation will serve as a background for the literature review and is based on prior research conducted in the following areas.

- Product Development Process
- PLM Background

Once these two areas of background have been discussed, a review of literature related to these topics will be given. The literature review will show specific advancements in PLM, and how these advancements relate to the security, efficiency, and reliability in storing and managing product artifacts. Only the most advanced methods of managing information within a PLM system will be studied. The literature will focus on the following areas in PLM:

- Collaborative work in PLM

- Data Mining

- Knowledge Based Engineering in PLM

- Automation in PLM

2.1 Background: Product Development Process

In order to comprehend how this thesis will improve the way products are designed, an understanding of the product development process is necessary. This section will give an overview of product development and identify areas for improvement. It is essential to grasp these areas of improvement since they are the focus for research discussed in the methods chapter (see Chapter 3).

The first concept in the area of product development is the idea of product artifacts. During the product development process, artifacts are created to support the product from its initial concept to its physical manufacture. Examples of these artifacts include 3D models, drawings, images, documentation, requirements and specifications. All the stages of product development—using customer inputs and company knowledge, developing predictive models, product testing, vaulting, and packaging and shipping—involve the creation of product artifacts. Before a final design is completed and is ready for production, the product may pass through many iterations of each of the design stages. Because these repetitions can be very time consuming, research efforts are currently being directed toward the optimization of each of the stages in the product development process.

Recent developments have been made that attempt to automate the process of generating product designs. A "design generator" (Roach et. al) automatically creates "all of the design

artifacts and supporting necessary information for the design of a customized product to meet the specific customer's needs." [12] As a result of innovative methods such as the design generator, increasingly complex products are being designed and created faster and more efficiently. This increase in design speed creates an increasingly heavy burden on those who are required to manage the associated product artifacts. The method shown in this thesis will attempt to address this need by automating the management of importing product artifacts into a PLM system.

In order to better understand how product data can be managed more effectively, it is important to map out each stage in the product development process. Roach et al. developed the ontological levels to better demonstrate how information passes through the product development process. These levels are shown in Figure 1.
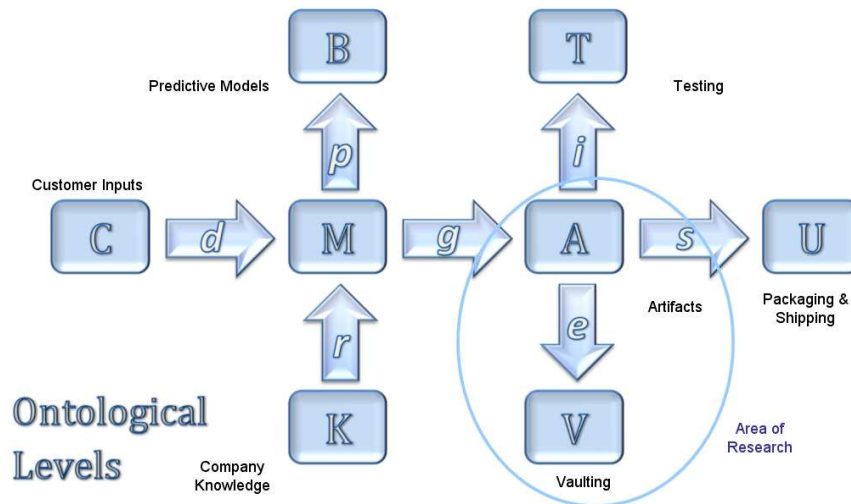
Figure 1 - Product Development Process [12]

Figure 1 shows that customer inputs and company knowledge ("C") drive models ("M") which are used to create product artifacts ("A"). These artifacts are then vaulted ("V"), and this vaulting process ("e") is the topic of this research.

9

Companies spend large amounts of time and resources vaulting and managing artifacts. Artifacts created in the product development cycle are not limited to engineering. They may include other information such as marketing documentation, instruction, warranty, packing and shipping documentation, specification, cost model, and bill of materials. All of these documents have historically been stored in an inconsistent fashion, varying greatly from company to company. Some prefer to store artifacts on network drives, others on local hard drives, and some use a combination of the two. Vaulting these artifacts in a centralized data repository for later use has become more common in industry today. [21] A centralized location enables all users to know that they are working with the latest revision. This eliminates much of the confusion caused when dealing with products which have multiple revisions. PDM and PLM are ideal tools to fill the role of having a centralized data repository to vault artifacts in the product development process.

2.2 Product Lifecycle Management (PLM) Background

PLM originated from PDM developments in the mid 1980's. [27] Robert Johnson was one of the first to reference the benefits of PDM in relation to solid modeling in 1986. [29] During this time, companies such as American Motors Corporation (AMC) began to develop systems to manage their CAD data in order to compete against other automotive manufacturers. Shortly after these initial PDM developments, AMC produced a new design which would later be called the "Jeep Grand Cherokee." Because of the success AMC had experienced, Chrysler purchased AMC and incorporated their PDM technologies. Soon after, Chrysler attributed PDM/PLM technologies in making them the lowest-cost producer, recording development costs which were half of the industry average by the mid-1990s. [27] Companies such as these began

using PLM software packages from IBM and Dassault later on in the late 1990's. These

packages would develop into a PLM system called "ENOVIA." PLM evolved from custom, in-

house implementations, to standardized packages which were offered by software vendors.

Figure 2 shows a timeline of PLM developments as they relate to increased business value.
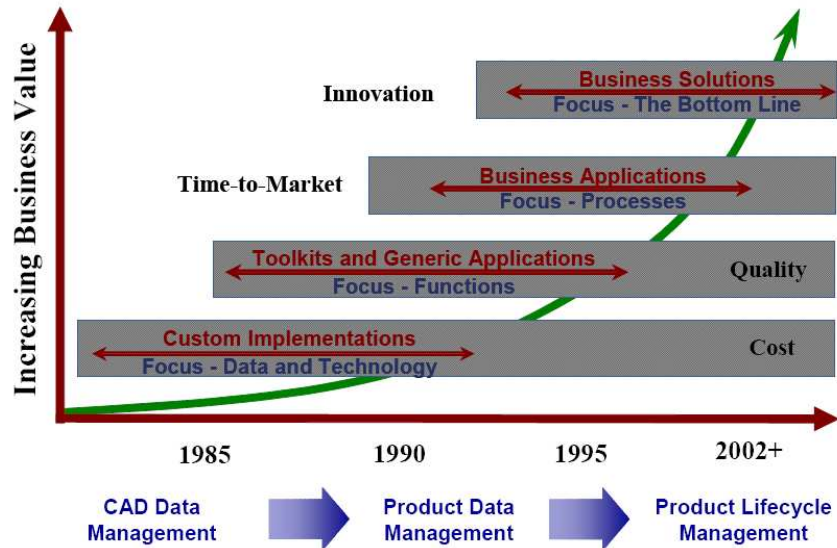
Figure 2 - The Evolution of PLM (Adapted from CIMdata, Inc.) [28]

As shown in the figure above, CAD data management evolved into PDM which evolved

into PLM. Just as CAD companies offer lower cost products such as Solidworks and SolidEdge,

Mid-Market PLM software such as SmarTeam by Dassault Systemes, are aimed at mid to

smaller size engineering companies. These lower end PDM systems typically do not offer the

full range of functionality provided in the more expensive PLM systems. Teamcenter Express,

for example, does not offer the full capabilities of Teamcenter Engineering which allows for full

customization of its PLM environment through a programming interface. Thus, the methods

described in Chapter 3 are currently reserved for only high end PLM products. Examples of these

high end PLM systems in 2008 were Siemens with their product Teamcenter, Dassault who makes ENOVIA, and PTC who develops their product called WINDCHILL.

As the concept of PLM evolves, the capabilities of these commercial PLM systems will naturally increase. This increase in capabilities within PLM will revolutionize the way products are developed and bring the full benefits of PLM to the masses. The research described in this thesis will build upon these capabilities to ease the transition companies have in importing legacy data.

2.2.1   Benefits of PLM

In today's engineering world, people are searching for a way to utilize computing technologies to solve complex problems. The choice to utilize advanced computing technologies often makes the problem more complex than it need be. In fact, the incorrect use of technology can cause a product to be designed slower than when engineers used drafting boards. The concept of PLM attempts to resolve these issues of added complexity by reducing wasted information, resources, and materials. [31] Other notable benefits of PLM have been mentioned in recent research and are outlined in the list below. [26]

- Reduced time to market

- Improved product quality

- Reduced prototyping costs

- Savings through the re-use of original data

- A framework for product optimization

- Reduced waste

- Savings via integration of engineering workflows

PLM also improves collaborative design tasks by simplifying the handling and passing of tasks from one site to another via process automation. PLM integrates people, data, processes and business systems, and provides a product information backbone for companies and their extended enterprise. Also, PLM serves as a central hub to the engineering wheel as products pass through designers, analysts and manufacturers. Recent PLM research by Sudarsan, Fenves, and Siriam has shown that PLM can "facilitate the semantic interoperability of next-generation CAD/CAE/CAM systems… The relevance of our framework to PLM systems is that any data component in the framework can be accessed directly by a PLM system, providing fine-grained access to the product's description and design rationale." [11] The interoperability of PLM to manage information is demonstrated in Figure 3. This information is vital to tools such as CAD, CAD, and CAM.
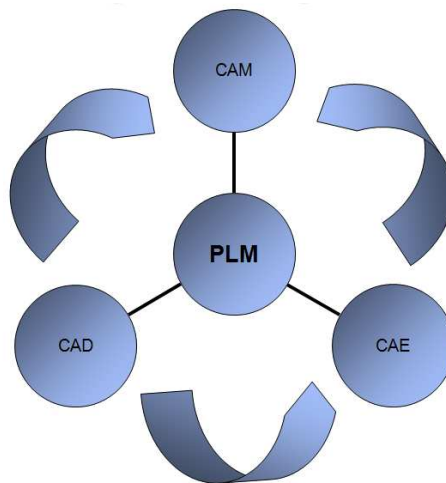
Figure 3 - PLM as Knowledge Based Engineering Hub

As previously stated, engineering competitiveness drives the need for faster design cycles and lower costs. By serving as the central hub to connect software and data to people within a company, research has shown that PLM has the capacity to meet the needs of this competition.

[25] Never before has there been a greater need for a tool, such as PLM, to aid in product development than today. [31]

Unfortunately, because of its rapid advancements, it is difficult for companies to use PLM to its full potential. As a result, its use is commonly limited to being a vaulting database that just stores product data. [21] However, these advanced systems have a far greater potential [1], as shown by Cochran and Hong, who performed automated manufacturing simulations that allowed for various classifications of experimental conditions to be defined in database tables. [13] Another example is the use of PLM as a knowledge based engineering (KBE) portal to store all relevant metadata concerning a product to be used by automation downstream in the design cycle and upstream in business applications such as Enterprise Resource Planning (ERP). [21]

In most of these cases, PLM systems must be customized though the use of programming techniques. These programming methods will be referred to as "PLM automation." When performed correctly, this provides a "highly sophisticated management of documents." [1] Several companies observed in this research, including Boeing, B/E Aerospace, General Motors, and General Electric, are striving to push the capabilities of PLM via automation. These companies are using features such as workflows, KBE, and automation to leverage more fully their PLM systems. The research proposed in this thesis builds on these advancements made over the past decade and attempts to take PLM a step further.

2.2.2 PLM Architecture

In order to comprehend how to take PLM a step further, it is important to understand how the architecture of a PLM system works. This is done by uncovering the essential components behind this technology. PLM consists of two main elements, namely a Relational Database Management System (RDBMS) and a File System. The RDBMS typically manages artifact

attributes while the file system manages product artifacts. A diagram showing how metadata and

artifacts are stored in a PLM data server is shown in the figure below. There are two main

computing components to a PLM system: a data server and the PLM clients. The data server is

the computer system that controls the File Server and the RDBMS. A data server may exist over

several locations or may be a combination of servers. The PLM clients are the individual

workstations which access the information on the data server. A diagram showing how metadata

and artifacts are stored and accessed in a PLM Client-Server architecture is shown in the figure

below.
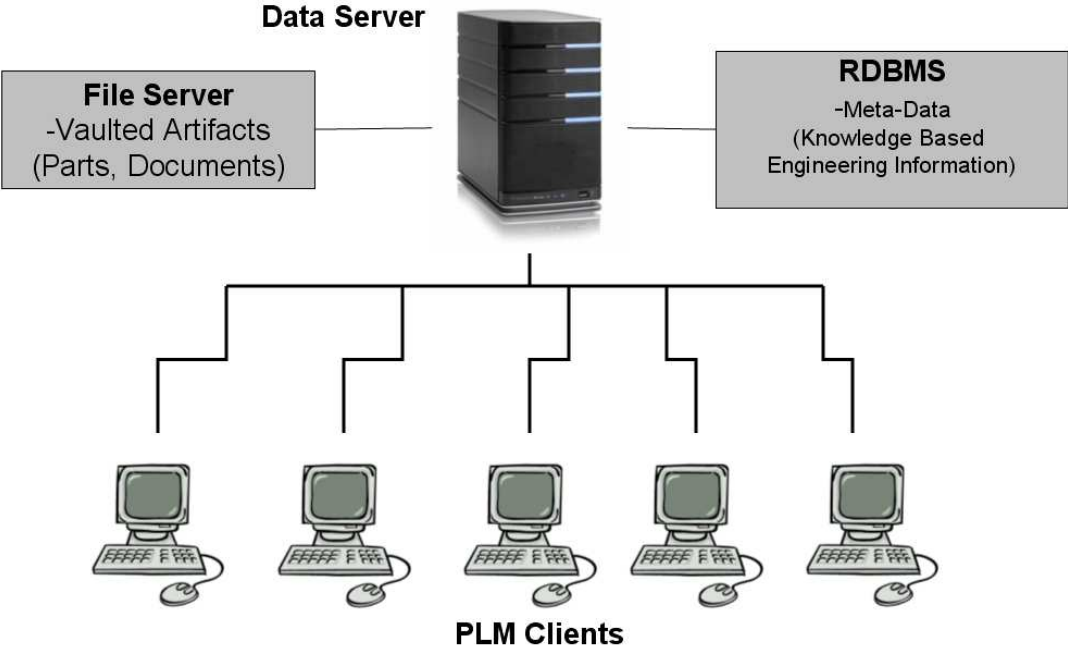
Figure 4 - PLM Client-Server Architecture (Adapted from Xu & Liu 2003 [2])

Figure 4 shows that a PLM data server consists of two main elements, namely a

Relational Database Management System (RDBMS) and a File System. The RDBMS typically

manages artifact attributes while the file system manages product artifacts.

Metadata may be taken from information stored in documents or spreadsheets in order to be stored in the RDBMS. Legacy files are typically located on a shared network drive or locally on a hard drive. Metadata and files may be spread throughout various locations inside a company. Each piece of data is relevant to the product and can be used later in the design process.

Inefficiencies exist when the same data is scattered throughout multiple locations within a company. Collecting this metadata into a central RDBMS is necessary to avoid inefficiencies. The following will show an example of this inefficiency. Suppose that there are two users working on the same artifact. This artifact could be a text document or a 3D model with its associated metadata (costs, analysis, manufacturing data, etc.). The artifact is currently on revision C. If User-1 is working on revision D of this artifact and completes his changes to revision D on his computer, another user in another location could also make different changes to revision C and call it revision D. Thus there exist two conflicting revisions of the same artifact. This conflict can take significant time and money to track and correct. PLM exists to address this issue and save money by storing all revisions on a central location. This allows User-1 to make changes to revision D, commit those changes before User-2 makes changes and releases revision E.

To avoid the aforementioned inefficiencies, Rolls Royce is devoting over 10 million dollars for the initial planning, setup and installation of their complex PLM architecture, which is to be completed by 2010. They found that PLM would serve as the mechanism to provide "one data source" to fuel all their business operations. Companies worldwide are now realizing the need to have product metadata and artifacts stored in a PLM system. As a result, many

companies find it worthwhile to make the transition to PLM. The research discussed in this

thesis is aimed at addressing this need, by simplifying a company's transition to PLM.

This shift in the trend of data management redefines the way products are developed.

Figure 5 shows an example of how different artifacts appear as they are stored within PLM.
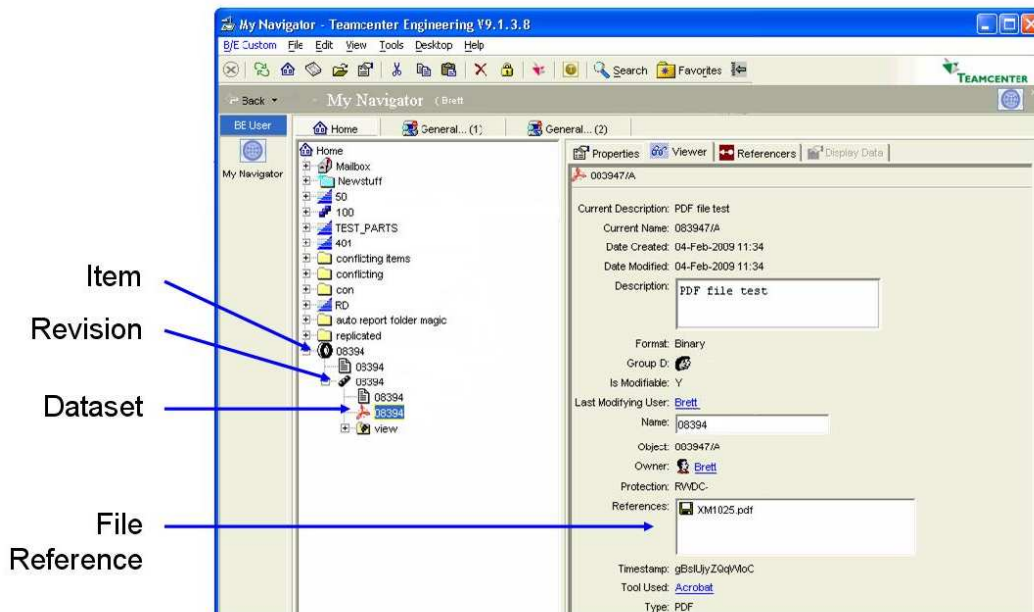
Figure 5 - Teamcenter Artifact Structure

This figure shows how files are stored in a common PLM system called Teamcenter. An

artifact (documents, metadata, files…etc.) is represented as an item. Each item has a revision,

dataset, and file reference. Items are at the high end of this hierarchical structure and may be

stored in folders or home directories. Each item may contain at least one revision. Revisions

may contain multiple datasets. Each dataset is an individual file type such as a document,

drawing, or any other file that pertains to the revision. A file reference is the actual filename of

the dataset as it is defined on the File Server. In Figure 5, the dataset pointed to is a PDF, and the

file reference is the name of that PDF as stored in the File Server. These names (item, revision,

dataset, file reference, etc.) may vary between PLM systems but the concept and their relationships are similar.

The understanding of PLM artifact structure is vital in understanding the method described in Chapters 3 and 4. At this point, the reader should be familiar with the basic elements and architecture involved with a PLM system.

2.3   Concurrent Work in PLM

Concurrent communication is the enabler of global business. [21] One of the methods of working concurrently in PLM is through the use of workflows. Workflow in a PLM environment is represented as a map with nodes that represents tasks and lines between the nodes that represent actions. Workflows allow tasks to move through processes in order to achieve an objective.

Computerized workflows were first used in the 1980's as companies began to model business processes electronically. The benefits of PLM workflows have been referenced by experts in the field. Qiao and Zhang state that, "On the basis of workflow technology and version rules, problems of data incompleteness and inconsistence are resolved." [23]However, workflows are expensive to develop and implement correctly. For this reason, workflows are currently not being implemented on all types of engineering data. Legacy data is generally handled manually without the aid of workflows. Research in this thesis will show how workflows may be called automatically after legacy data is brought into the PLM database to improve data handling efficiency.

Design processes must be kept consistent and well defined in order for a workflow to aid in automating a particular process. An example of a well defined engineering change process model is shown in Figure 6.
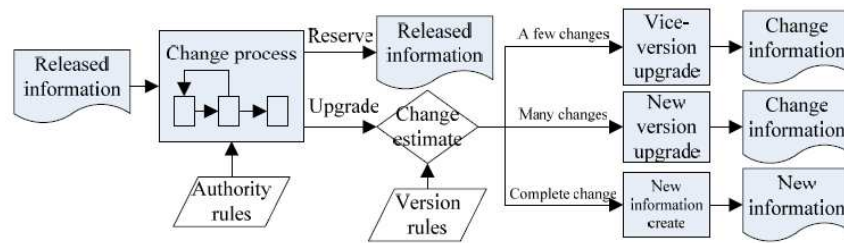
Figure 6 - Engineering Change Process Model [23]

Teamcenter Engineering, allows users to set up custom workflows. These workflows can be customized to allow legacy artifacts to be stored and managed. Engineering experts commented that setting up workflows to automate the way their PLM system manages legacy data, would dramatically improve their product development process. The research in this thesis will demonstrate that workflows can be used to address this need in industry. A further discussion of legacy data management will be presented in Chapters 3 and 4.

One of the main benefits associated with PLM on a global setting is discovered by the use of workflows. With workflows, tasks can be easily transferred to individuals around the globe to accomplish a given task. This means tasks can be performed more efficiently since a work day is no longer limited to eight hours. This process of collaborative engineering around the globe is referred to as "Following the Sun." A real-time collaborative environment presents many benefits. According to Barbosa one benefit is that it, "…should allow teams to work cooperatively, accessing and changing information in distributed engineering data systems at run time." [14] When viewed in this manner, globalization is a benefit to the design process as,

"work functions are no longer limited by organizational, geographic, cultural or time barriers."
[16] This research will build on the idea of this globalized collaborative engineering environment by demonstrating how template workflows can be populated with the use of an algorithm which sends tasks automatically to individuals around the globe.

The globalization and decentralization of companies today may be viewed as a burden which slows product design efforts due to the barriers involved in this collaborative environment. However, updates in PLM and computer technology allow for work to be performed more efficiently in a global collaborative setting. One example of a global company which is using PLM to improve collaborative product development is BAE Aerospace. While developing the Eurofighter Typhoon aircraft they have devoted three full time employees to help their engineers understand the benefits of PLM.

Design efforts at BYU have demonstrated how to effectively work between multiple groups around the world. During these global collaboration projects, multiple universities, which span the globe, designed a vehicle for General Motors. These projects demonstrated effective means of exchanging engineering data using PLM tools, as these students were able to design the vehicle in approximately one academic year. One suggestion from this research is that PLM tools, such as Teamcenter Engineering should, "be put in place well before the project begins."
[24] By placing PLM in the picture before project initiation and establishing engineering processes as workflows, the full benefits of PLM may be experienced in a concurrent environment. In order for PLM to be well established before a project starts, PLM software companies utilize highly trained individuals to assist in on-site customization to meet the needs of a project. Global engineering efforts require additional resources to complete this customization and setup. The PACE vehicle collaboration project had multiple PLM servers in

different locations around the globe. A figure showing the server locations for the PACE project is shown in Figure 7.
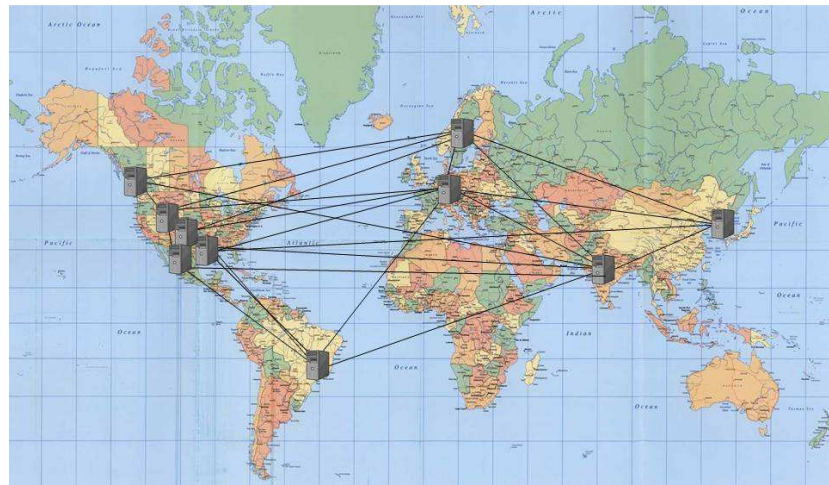
Figure 7 - Proposed PLM Server Locations for PACE Collaborative Project [24]

PLM deployments can vary according to company needs. For instance a complex global product development project will vary greatly from a small company. Lund and Fife mentioned that PLM systems "are not standard equipped to satisfy the diverse needs of every company." [10] Common variants of PLM installations for concurrent engineering companies are three-tier and four-tier configurations. A diagram of a basic PLM three-tier structure which is commonly used in a concurrent working environment is shown in Figure 8. After installing a user interface front end on each application client, a user is able to access data from the server. However, recent developments allow for users to log into the database via a web portal over HTTP. Liu and Xu suggest that a web browsing client should be used as the optimal client for PLM systems because of its standardized and globalized nature. [7]
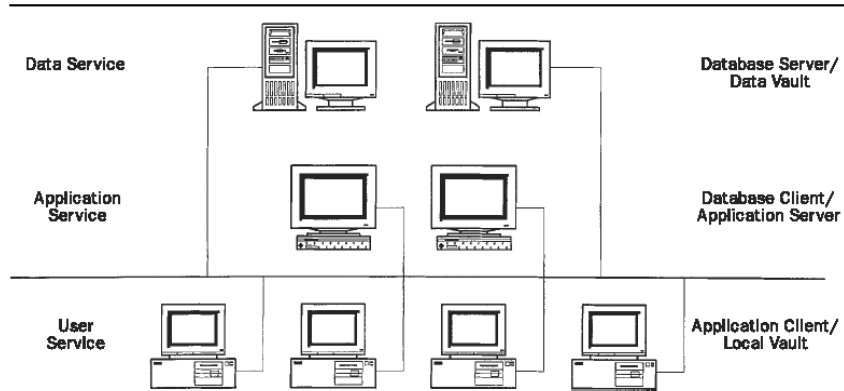
Figure 8 - Three-Tier PLM Architecture [4]

This web based client connection has been referred to by Liu and Xu as a "Web-Enabled PLM System in a Collaborative Design Environment." [8] Depending on company needs, the architecture of PLM may vary. Regardless of architecture, the way concurrent work is performed in an ideal PLM remains the same.

By introducing an automated import of legacy data and automatically calling workflows, a step toward this ideal collaborative environment is taken. Engineering data can be easily accessed from within the PLM software to all sites from within a company, increasing design productivity. The concept of how engineering workflows improve collaborative engineering is fundamental to understanding the benefits of the research proposed in this thesis.

2.4   Data Mining

Data mining is the process of extracting data by the use of patterns. Tools commonly used to uncover these patterns are Neural networks, Clustering, Genetic algorithms, decision trees and vector machines. [32] The objective of these data mining is to extract data, clean the extracted data, and interpret the results. [33]

Several standards for data mining have been established for turning data into useful knowledge. The most prevalent of these standards are the 199 European Cross Industry Standard Process for Data Mining (CRISP-DM1.0) and the 2004 Java Data Mining standard (JDM 1.0). The CRISP standard divides data mining into 6 main stages. [34]

- Business Understanding

- Data Understanding

- Data Preparation

- Modeling

- Evaluation

- Deployment

These stages will be applied to this research and slightly modified to adapt to an engineering environment. These data mining stages will serve as a framework to convert data into useful knowledge and therefore serves as an integral part of this thesis. This knowledge can then imported into PLM and used throughout the rest of the product development process. The application of these stages will be discussed more thoroughly in chapter 3.

2.5 Knowledge Based Engineering in PLM

The origin of knowledge based engineering (KBE) closely tracks CAD developments, and has done so over the past few decades. In the late 1980's, research done by Kaiser, Barghouti, Feiler, and Schwanke, was done to show how database design could be implemented to support KBE in an engineering environment. [30] At the same time, companies began to look for methods to store product knowledge with the CAD models and drawings. A popular method to accomplish this was to store the knowledge as attributes. Ideally, these attributes are created

when the product concept and prototype are in the early phases and follow the product all the way through its lifecycle.

Attributes are typically used in PLM as parameters to define the product. Standard PLM attributes for artifacts are typically item description, date, owner, group, item type, etc. The list of custom attributes can be quite extensive depending on company needs. Custom attributes may be created which are searchable from within the PLM user interface. Examples of custom attributes include origination date, author, engineer, job number, customer, and the design system or assembly.

Recent research done by Bowland et al. [17] demonstrates a method using the database of a PLM system to maintain associative data in assemblies for features not offered by the CAD system. Also, Lund showed a similar method to store modeling and assembly parameters, which have been stored as attributes to reduce the size of items within PLM. [21] A layout of this research as a Unified Modeling Language (UML) diagram is shown in Figure 9. Customization of PLM attributes can be an arduous process. According to Chu and Fan, PLM customization can be more time intensive than the development of the PLM system itself. [4] With a foundation of a PLM system customized with attributes which fit a given product design, a company can customize a solution to fit their individual needs.

One way to customize PLM through the use of attributes is to automate the direction of process workflows based on product attributes. For example, if an artifact contains an attribute that indicates it is associated with a particular airframe such as the A380, the file can be automatically sent for review to the engineering manager associated with that airframe. To support this concept, one manager stated: "In today's distributed manufacturing environment,

how to manage the product data and distributing it to the right people is critical to the success of

a company." [5]
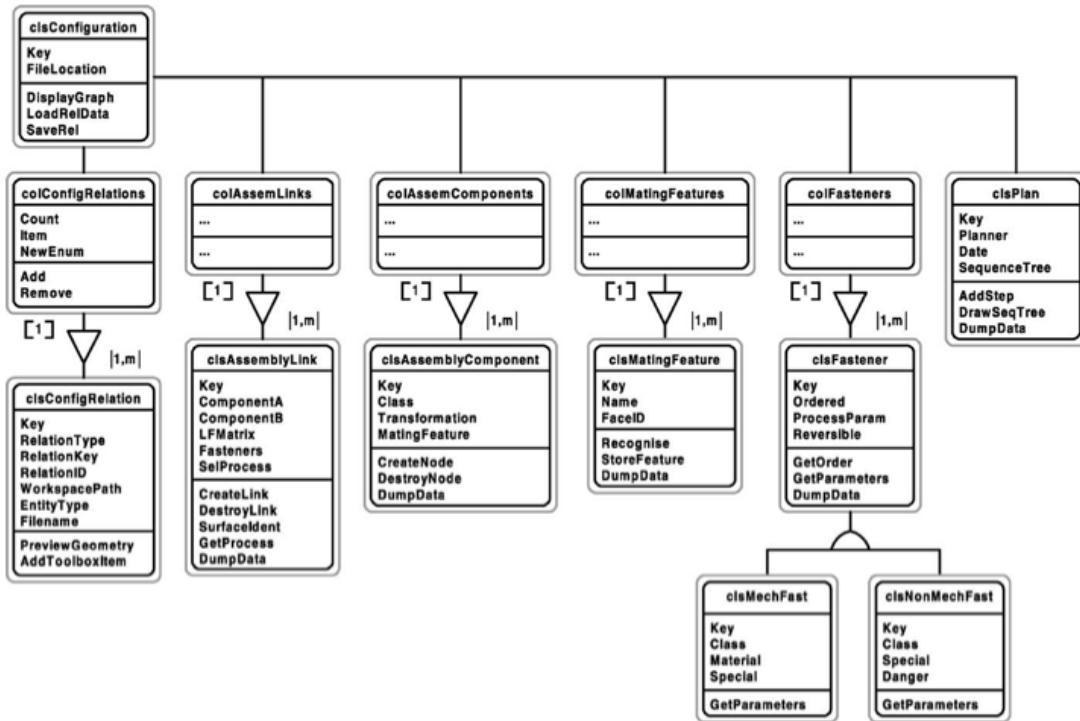
Figure 9 - UML of Assembly Data Stored in PLM [1]

Another way to use attributes in PLM is in the storage and reuse of geometric data. Peng

developed a custom solution which used geometric data inside databases for rapid spring design.

[15] In the study presented in this thesis, a similar method will be used to manage engineering

attributes for use within PLM.

The use of attributes such as defining features, parameters and relations are essential to

improving product reusability. To demonstrate this concept, an example of a reusable CAD

assembly will be given. The assembly shown in Figure 10 is an inhaler used to keep breathing

air temperature elevated in cold conditions. The features involved with each part in the CAD

model must be defined along with the parametric relations. If the model is well defined with attributes, rebuilding another instance of the assembly shown below is simplified. By changing a few parameters, such as the inner and outer diameter, the assembly features are modified and can be easily used to quickly design another instance of this product. This re-usability is a major advantage of using attributes in a CAD environment. Figure 10 shows an example of a reusable design for a CAD inhaler assembly.
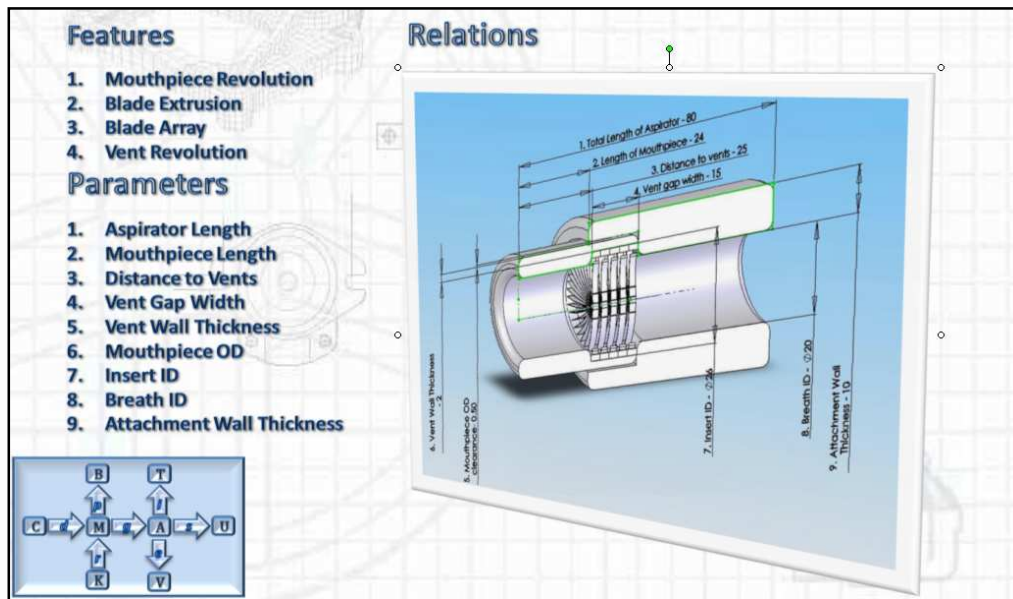
Figure 10 - Reusable CAD Model Via the Use of Attributes

This figure shows the parameters, relations and features of a CAD assembly. The relations section of the figure shows an envelope for the design. Each dimension, such as length, ID and OD, is labeled and numbered, this allows each relation to be easily identified and used in parametric equations. A typical parametric equation is shown by

$$P(t) = P_0 (1 - 2t) + P_1 t \tag{1}$$

where the output in this function P(t) is shown as a function of the two input points P_0 and P_1. These points vary as a function of t which typically is given a range from 0 to 1. This equation represents a simple function which provides insight into the concept of a parametric relationship.

The use of parametric relationships in this manner is helpful in CAx models and allows for greater flexibility to design in a dynamic fashion. Similarly, attributes allow for re-usability within PLM in that design attributes from previous designs may be searched and then re-used. However, this functionality does not come as a standard feature in PLM installations. Speaking on this subject, Gao and Aziz mentioned, "there are a number of shortcomings in the current crop of commercially available systems, such as the lack of design knowledge sharing" [22] To remedy this issue, Oh et al. [19] demonstrated an example of integrating knowledge based engineering into a PLM system using the unified modeling language (UML).

The application of KBE also applies to product artifacts such as instructions, warrantee information, and packaging. These artifacts can be re-used as templates to simplify new designs by only requiring the user to modify associated parameters. Re-usability is a major advantage of knowledge based engineering and results in cost and time savings.

The use of attributes within PLM is directly linked to a full understanding of the methods section of this thesis. Once the reader can integrate the concept of attribute usage in PLM, the vision of this thesis becomes clear.

2.6   PLM Automation Research

With research being performed in the areas of attribute and data organization within PLM, there is still much work to be done in the area of PLM automation. As mentioned previously,

PLM automation is the customization of PLM though the use of programming techniques. The main goal of automation in PLM is to improve the speed and efficiency of common operations.

The metric of computational cost will be evaluated to quantify the improved speed and efficiency of PLM automation over traditional methods. Even though computers have increased dramatically in speed over the past few decades, client-server connections can still be easily overwhelmed. PLM automation reduces the overhead involved with these connections, and as a result, reduces both server and local computational costs. These costs are the main factors in PLM efficiency according to research performed by Leong and Lee. [3] The sum of computational costs are represented as

$$\sum_{i=1}^{m}(x_{ij}^{a}r_{ij}C_{local} + x_{ij}^{a}s_{ij}) = \sum_{i=1}^{m}[x_{ij}^{a}(r_{ij}C_{local} + u_{ij}C_{local} + s_{ij})] \tag{2}$$

where i represents the data being retrieved or updated, $x_{ij}^{a}r_{ij}C_{local}$ represents the local retrieval cost, $x_{ij}^{a}u_{ij}C_{local}$ represents the local update cost, and the local storage cost of object i in site j is represented by $x_{ij}^{a}s_{ij}$. Note that local update cost goes to zero as it is eliminated from the left side of the equation. This is because there is no local update cost involved in terms of PLM efficiency. Similarly, when an object i is updated via sites j and k, the remote retrieval, update and storage cost is

$$\sum_{k \neq j, k=1}^{n}x_{ijk}^{r}\left(r_{ij}C_{remote} + u_{ij}C_{remote} + s_{ik}\right) \tag{3}$$

In order to reduce the overall bandwidth consumption, both local and remote costs may be reduced. By doing this the total cost of PLM operations decreases exponentially. As a result, the user will notice a dramatic change in the speed of common PLM tasks.

Because of this improvement, research in automation has been applied to various areas in the product development. One area of research which attempted to increase efficiency is the automation of analysis runs on systems with PLM-like architecture. Carroll and Hawkins [18] created a software package to allow multiple computers to run analysis using a three-tier web system. Additionally, Fife et al. [24] demonstrated the automation of process control and optimization routines inside PLM. These examples of PLM software customization are aimed at resolving the shortcomings of PLM as they apply to design, analysis, and optimization techniques.

Because PLM is a relatively new technology, several areas of PLM automation are not yet developed. Automation of artifact import with KBE attributes is one of these undeveloped areas in PLM research. This type of PLM automation will reduce the amount of local and remote requests by removing the interactive steps required in a given task. For example, the steps required to import a single artifact into PLM averages around 11 steps. The steps involved with manually inserting an artifact into Teamcenter Engineering are listed below:

1. Item creation

2. Specify Item type

3. Gather product data (revision, name, itemID)

4. Insert Item

5. Revision creation

6. Specify revision level

7. Enter revision data (description of changes)

8. Insert Revision

9. Create Dataset

10. Select File type (PDF, Word Document, Etc…)

11. Collect and insert metadata associated with the dataset

These 11 operations which were recorded interactively can be greatly simplified through PLM automation to a single button click. By doing this, the resulting number of local and server requests associated with creating an item, revision, and dataset is simplified by representing these interactive steps as programmatic function calls. As a result, the application of PLM automation to legacy data import is a solution which will save valuable time during the product development process. Thus the concept of PLM automation is essential to realizing the goals presented in this thesis: to perform a reliable, efficient, and secure automated import of legacy artifacts. Also, by understanding the current state of PLM automation, the reader can more fully comprehend the steps which will be discussed in further detail in Chapter 3 - Method.

3 Method

3.1 Strategy for Importing Product Artifacts

This chapter will discuss the method developed to automate the import of product artifacts. As mentioned before, the methods used in this thesis are applicable to other database and PLM systems and will be generalized for this purpose. The following are steps were chosen to approach an automated import in a generalized manner.

1. Preprocess data to ensure proper metadata exists in the filename structure and files are properly organized.

2. Create proposed metadata table indicating what files need to be imported into the PLM system and identifies metadata to be inserted.

3. Generate master metadata table from parsing proposed metadata table and checking the actual files.

4. Create metadata genetic code for each item stored in PLM.

5. Store genetic code and link to item

6. Append completed code to master metadata table

7. Run the code using the automated import algorithm

8. Resolve any errors

3.2   Critical Data Structures

In order to accomplish the steps of the proposed strategy, several critical data structures are necessary. The following is a list of these critical data structures.

- Filename Structure

- Proposed Metadata Table

- Master Metadata Table

- Genetic Code

The next sections will explain these critical data structures and why they are required.

3.2.1   Filename Structure

The filename structure establishes a starting point for the initial metadata. The metadata found inside the filename structure are required to perform database queries after the files have been imported. A generic representation of the filename structure is

$$[Metadata_1] - [Metadata_2] - \ldots - [Metadata_n] . [File Extension] \tag{4}$$

where the file extension indicates the file type (i.e. PDF is an adobe acrobat file). The significant metadata are determined depending on query structure of the database system. The following metadata are required to formulate queries to an engineering PLM database component.

- Part number

- Version/Revision

- Name of part

The resulting file structure for an engineering PLM system might be as follows [Part Number] – [Revision] – [Name] . [File Extension]. An example of this data structure is shown in Figure 11.
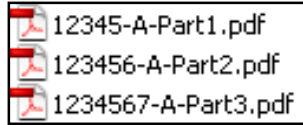
Figure 11 - Standard File Naming Scheme

3.2.2 Proposed Metadata Table

Another critical data structure is the proposed metadata table. A company's proposed metadata table will depend on what metadata they want in the PLM system. This includes file info and necessary metadata not included in the filename. The structure of an example proposed metadata table is shown in Table 1. The columns of Item and Revision in the table must match filename structure for the algorithm to work correctly.

Table 1 - Organized Metadata

Document	Item	Rev	MetaData1	MetaData2
1	12345	A	Part1	Author1
2	123456	A	Part2	Author2
3	1234567	A	Part3	Author3

3.2.3 Master Metadata Table

The master metadata table contains the parsed file structure data along with the associated file path. The primary purpose of this table is to serve as a key which links the files to the company's metadata located inside the proposed metadata table.

This table ideally duplicates the proposed metadata table if all files and data are present and match. Otherwise the item in the master metadata table that does not match the proposed metadata table will be skipped or ignored.

3.2.4   Genetic Code

The genetic code contains all of the relevant metadata to describe a particular file. This includes all of the data inside the proposed and master metadata tables. The order of the data contained in the code must be kept consistent so that data is inserted in the correct order in the database. To accomplish this level of consistency an algorithm is used to extract the code from the tables. The information and structure of the code is similar to the filename structure and metadata table format. The code is defined as

$$[\text{Metadata}_1] \; \% \; [\text{Metadata}_2] \; \% \; \dots \; \% \; [\text{Metadata}_n] \qquad\qquad (5)$$

where the % symbol indicates a delimiter whereby parsing is accomplished. All queries will be based on this code which is richer than both the filename and the metadata tables because it contains all the relevant data to be stored in the database.

By utilizing these critical data structures, the import process is greatly simplified. These structures eliminate import redundancies, and therefore serve as essential elements which allow this method to achieve a more efficient, secure, and reliable import of artifacts.

3.3   Application of Strategy

In order to apply the aforementioned strategy and critical data structures, the following stages will be utilized:

- Data Understanding

- Artifact Preparation

- Data Collation

- Error checking

These steps are slightly modified from the CRISP-DM 1.0 data mining standard to describe the tool that was created to organize and import data into PLM. This tool will be called the Automated Artifact Import Tool (AAIT). An overview of this tool will be presented, for a more complete overview of the code required in the AAIT, please refer to Appendix A: Automation Source Code.

3.4   Data Understanding

The first stage in a data mining process is to gain an understanding the data involved. Metadata will be taken from an engineering spreadsheet which contains a log of various product artifacts. The product artifacts are stored in folders which are organized according to their respective product. A graphical representation of this data structure is shown in Figure 12.
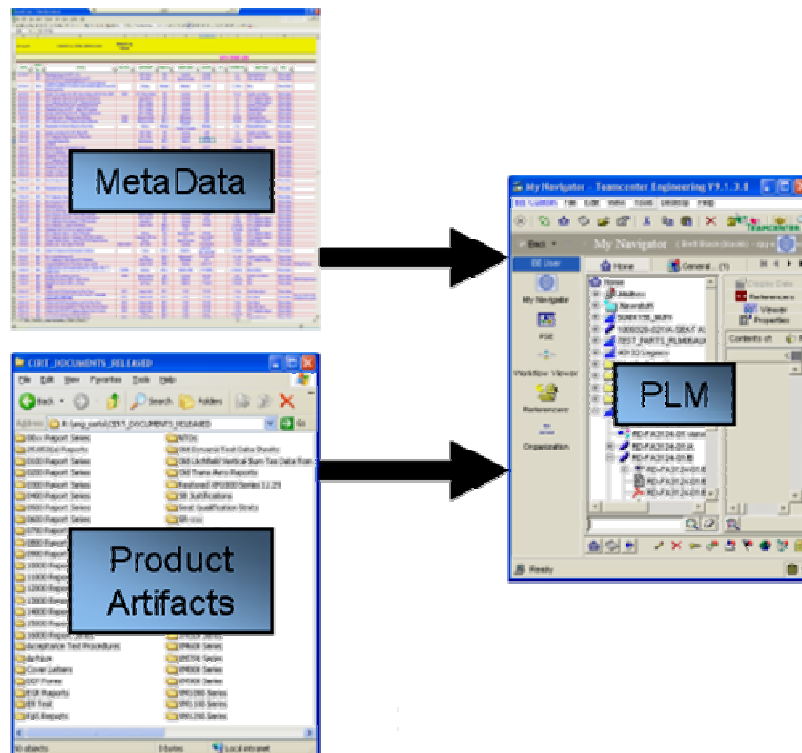
Figure 12 - Graphical Flow of the Automated Import Method

As seen in Figure 12, Metadata and Product artifacts will be merged into a central repository, PLM.   The next sections will discuss the steps required in the preparation of both the metadata and artifacts.

3.5   Artifact Preparation

It is imperative that the method for data preparation be carefully considered when gathering data related to each artifact. Prior to import, artifact preparation must be reviewed on a case by case basis for each artifact type. As each artifact is reviewed, it is necessary to filter out any extraneous data or files not related to the intended product. Also, in order to ensure a reliable upload, data associated with each artifact must remain consistent. The data associated with the artifact preparation will be:

- Item name

- Item revision

- Item number

- Description

- File name

- File path

The first three metadata (item name, revision, and number) will be collected and stored inside the file name. The remainder of the metadata (such as company specific data), is stored separately in a spreadsheet.

The process of collecting the first three metadata is a simple but repetitive task. Therefore employees will often not update these items, creating errors that can create problems in the process workflow later on. This is because updating the metadata involves recording the new

revision number each time a change to the artifact is made. This can be a tedious process, but can save time in the product development workflow. Fortunately, many companies already have this data gathered in a central location such as a spreadsheet.   Gathering the data into a consistent and standard format will be shown as the most vital step in preparation for artifact import.   Because this vital step is prone to human error, error checking measures will be demonstrated later on in this chapter.

The standard file naming structure will be utilized. As mentioned before, artifact files will be named with the following syntax: [Item Number]-[Revision]-[Item Name].[File Extension].
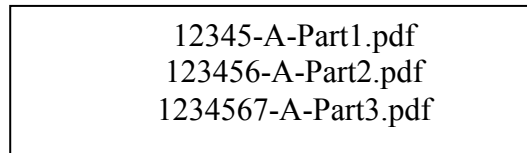
Before the revision number, item number and name are extracted from the filenames, the files need to be located in the same directory. Often, employees have not updated the folder structure when creating new revisions. In this case, the files need to be moved into a common folder before import. Moving these files is a manual or interactive process of organizing file and folder structure. This process is typically not time consuming as most legacy files will already be stored in a common folder.

By naming the files in a standardized fashion and moving them into a central location, as described above, the associated metadata must be parsed (extracted) from the files for use in the PLM system. To parse data from the files, a command may be run from the command prompt to write the file names to a text file. The command line shown below extracts all file names, in a given directory, with their associated path.

F:\Temp> dir /b > output.txt

Command Line 1 - Command Line File Name Extraction

37

Similar commands exist under alternate operating systems or programming languages. This command will return a text file containing all the file names from a given directory. Continuing with this example, Command Line 1 when performed on the files in this example would return the text information shown in Figure 13.

```
12345-A-Part1.pdf
123456-A-Part2.pdf
1234567-A-Part3.pdf
```

Figure 13 - Extracted File Names

In order to separate the item number, revision, and description, the text file in Figure 13 needs to be parsed. Parsing filenames is a simple process as long as the naming structure is kept consistent. Thus while the method is robust enough to handle inconsistencies in the data, maintaining a consistent artifact naming structure can decrease the time required to develop a parsing method. Once the results file has been parsed, the data is ready to be combined with any additional product metadata.

3.6   Data Collation

The product metadata related to a particular artifact can be stored in various locations, files, and formats. Collating all of the product metadata related to each artifact is the next step in this method. By assembling all of the product data into one centralized location company-wide redundancies and inefficiencies are eliminated. In order to accomplish this, the data must be organized, delimited, and migrated to a PLM environment.

Organization is the first step of data collation. Depending on the culture of a given company, artifact metadata may be stored in individual documents, spreadsheets or a database. In order to organize the data, one must identify where the data is stored, filter the data, and gather the data into one source.

Identification requires company expertise and knowledge to correctly assess where the product metadata is located. In order to do this, it is necessary to consult with the individuals who are working with this metadata on a regular basis. Once these metadata locations are identified, the data sources must be evaluated for extraneous data that needs to be filtered out. The filtering can be automated by using a program to identify patterns in the source data. The code used to filter the data will take the relevant data and gather it into one source.

To demonstrate this concept of data organization, we will continue from the example in section 3.5. The parsed file information (master table) will be combined with the data in an engineering log (proposed table) which contains information relevant to the product design to formulate the genetic code. Table 1 shows a sample of the parsed artifact data to be merged with this spreadsheet.

Table 2 - Organized Metadata
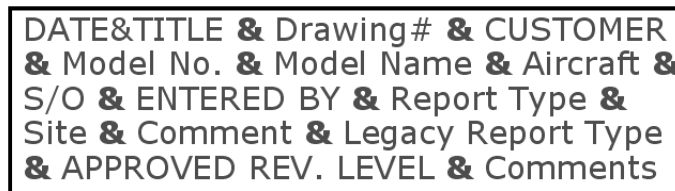
Path	File	Item	Rev	Description
F:\temp	12345-A-Part1.pdf	12345	A	Part1
F:\temp	123456-A-Part2.pdf	123456	A	Part2
F:\temp	1234567-A-Part3.pdf	1234567	A	Part3

Now the path, filename, item, revision, and description are contained in separate columns. The description column contains the company metadata such as cost, supplier info, and manufacturing data. If more company metadata is needed to describe the artifact, more columns

are added for use as custom searchable fields within the PLM database component. This entails

filtering redundant information from the metadata. A generalized way used to organize the

metadata's genetic code into a single string is shown as:

$$D = di + di_1 + di_2 + di_3 + di_4 + \ldots + di_n \qquad (6)$$

where D is the description and each di_i is a piece of metadata. An example of this would be

"part1&cost&supplier". Note that the metadata are joined and delimited by an ampersand

symbol. This allows the database to know that the artifact not only contains information about

the part, but also about the supplier and cost as well. Once organized, the data is joined into a

simple query line to be inserted into the PLM database. This query will contain actual pieces of

metadata for each artifact. A real world example of the genetic code when applied to a single

string of metadata on an engineering drawing is shown in Figure 14.

```
DATE&TITLE & Drawing# & CUSTOMER
& Model No. & Model Name & Aircraft &
S/O & ENTERED BY & Report Type &
Site & Comment & Legacy Report Type
& APPROVED REV. LEVEL & Comments
```

Figure 14 - Example of Concatenated Metadata for an Engineering Drawing

The next step is to build a command structure in the form of a database query which

inserts attributes such as file, path, revision, item type, and description string into columns of a

database. This is done by compiling the genetic code in Figure 14 with the file name, path, and

item number. Table 2 shows an example of this command structure for each of the part artifacts.

Table 3 - Command Structure

Command
Part1&12345&A&ItemType&Item Revision&PDF&pdf&Acrobat&F:\temp\12345-A-Part1.pdf
Part2&123456&A&ItemType&Item Revision&PDF&pdf&Acrobat&F:\temp\123456-A-Part2.pdf
Part3&1234567&A&ItemType&Item Revision&PDF&pdf&Acrobat&F:\temp\1234567-A-Part3.pdf

Once organized as shown in the table above, the queries will be tested before the import occurs. The data is now prepared and ready to be imported. A more thorough discussion of the import phase will be discussed in the implementation chapter as it relates to specific software and functions within the PLM system.

After the data is imported, a workflow must be called to release the newly imported artifacts. However COTS (Current Off The Shelf) PLM systems do not allow datasets to be eliminated as the item is revised. This is sometimes necessary as certain artifacts are not needed in future revisions. For example, Figure 15 shows a scenario where a PDF document that exists in revision A but is no longer required in revision B.

The AAIT was developed with this functionality, and allows the user to revise item revisions while retaining only the necessary datasets. This functionality prevents users from accessing datasets which are not relevant to a future revision.
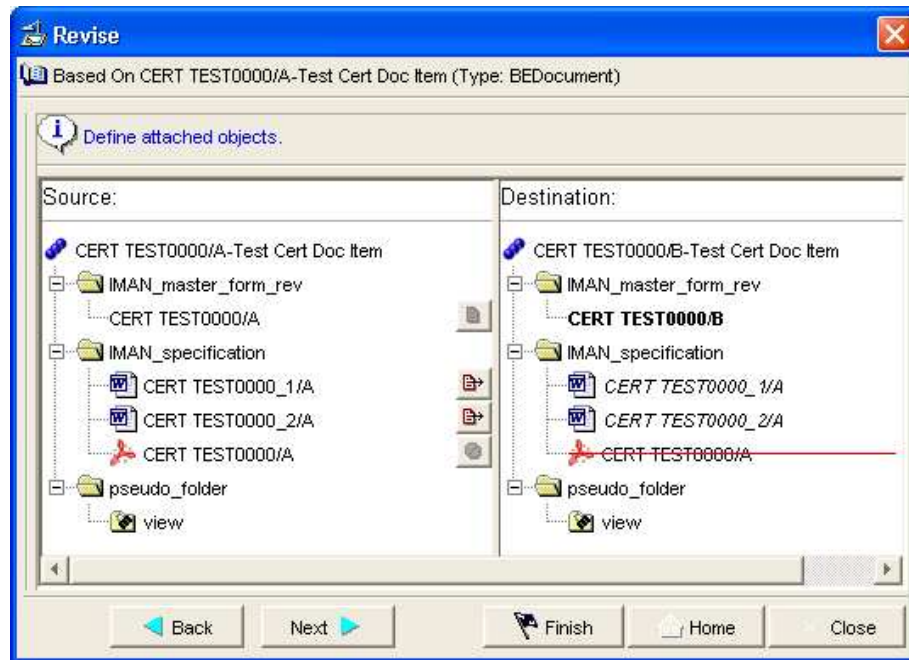
Figure 15 - Elimination of Datasets through Custom Revision Workflows

An additional preventative measure is that of freezing the data during the import. During a data freeze, engineers or employees working on the items inside PLM must halt work on particular files as they are being imported. A freeze indicates that all files in a particular folder or location become unavailable for modification. With the AAIT, this downtime period is significantly minimized saving the company time and money. In this way, employees are able to more quickly continue their work on secure files.

3.7   Error Checking

Once items have been imported, error checking is essential to show which artifacts failed to import correctly. Error checking during the automated import is done by using status codes. As status codes are displayed on the screen they identify trends and issues as the program is running. This can be extremely valuable in saving time as the entire run can be cut short if a

group of errors are found. A group of these codes which were created to show the status of each item as it is imported are shown in the list below.

- 999 - Item and revision uploaded successfully

- 888 - Item already existed, revision uploaded

- 777 - Item and Revision already exist, dataset not uploaded

- 1 - Ready for Process Algorithm

The codes listed above were created for use with the Automated Artifact Import Tool (AAIT). If the item failed to import correctly, an error message was displayed on the screen. Some of these errors will be discussed in the next sections, including: dealing with maximum revisions, invalid revision, and invalid item numbers.

3.7.1   Maximum Un-Released Revisions Reached

In PLM terms, once an item is released, the user is no longer allowed to edit the artifact revision within that item. It is important to release all previous revisions of an item before introducing new revisions to the PLM database component. Because of this, certain database schema rules prohibit the number of unreleased revisions. This limit imposed by the PLM schema for a given revision will be denoted by R_L. This revision limit can cause an issue when attempting to import multiple revisions which belong to the same item. For example, if an item is currently on revision C in the PLM system, and a user creates revisions D and E, E cannot be imported until revisions C and D are released (if $R_L = 1$). This is because the revision limit will only allow one unreleased revision. If $R_L = 2$, then revisions D and E could be imported as

unreleased revisions, and only revision C would need to be released. This is possible because the PLM schema would only allow 2 unreleased revisions because $R_L = 2$. If the number of revisions to be imported (R_n) exceeds the unreleased revision limit (R_L), no revision will be imported.

$$R_n > R_L \qquad (7)$$

Because of this requirement, additional logic must be applied to each revision. If a revision being imported (R_i) is newer than the latest revision (R_{di}) stored in the PLM database (indicated by the Đ symbol), then the latest revision stored must be released before the newer revision is imported.

$$\text{If } R_{di} \text{ Đ } R_i \rightarrow Rel \left(R_i \right) \qquad (8)$$

The sample database used for this method had the number of maximum un-released revisions set to 1 or 2 depending on the item type. With this revision limit in place, only a small percent of the items reached the revision limit. More details on the results of the test case as it applies to revision limits will be given in the results section.

3.7.2   Item Number Unavailable / Naming Collision

Item numbers can be already reserved by existing product artifacts within the PLM database component. Commonly there are item numbers which already exist in PLM such as supplier parts. These items may have the same item numbers as an item which is to be imported by the import tool. As a result, there are two parts which are entirely different in design intent and function sharing the same item number. To accommodate for this case of an item number collision, an executive decision must be made concerning a new naming strategy. For example, if a part number N_i is already reserved in the database, or exists in the set of items entities E, then a prefix P_i or suffix S_i is attached to the item name to alleviate the collision. To implement the prefix based logic, the part number would then become

44

$$Ni \in E \rightarrow Pi + Ni \tag{9}$$

where P_i is a predetermined prefix that is added to the part number. Alternatively a suffix is added to the desired item number to avoid a naming collision. In this case the part number, N_i is also a subset of the set of item entities, E, and can be modified as

$$Ni \in E \rightarrow Ni + Si \tag{10}$$

where the suffix, S_i, is appended to the part number. According either scenario, a prefix or a suffix can be added to item names inside the artifact set depending on company or user preference.

Another naming solution which can be used to avoid item collisions is the use of a unique naming schema. This schema entails parts of the document metadata. For example, if the artifact contains metadata, such as the a job number, document type, group code, and a unique three digit serialized number then a naming convention for this schema is used. The schema is shown in Figure 16.
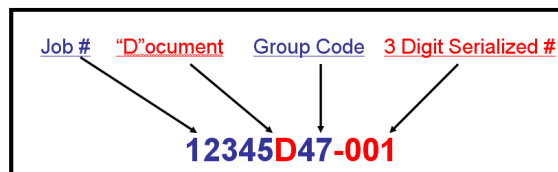
Figure 16 - Custom Naming Schema to Avoid Item Collisions

3.7.3   Invalid Revision Character

On legacy artifacts, a revision character, which exists on a drawing, specification, or technical document, may not be a valid revision character according to predefined rules in a PLM schema. In this case, default alternate revision characters are chosen. This alternate revision character must be unique from other documents as it can be used as a flag to denote that it is a pseudo-revision level. This will indicate to any future users that the revision on the item in

45

the PLM system is different from the revision on the document. For example, if the revision on a particular product drawing is a character such as α, the PLM system may not consider this to be a valid entry. A new default revision is proposed, such as a number 47. Items in the PLM system with a revision of 47 will now serve as a flag to the user that a potential revision conflict may have occurred. The figure below shows how a simple "if statement" can be used to replace an invalid or non-existent revision character with a predetermined revision flag.

$$\text{If } R_i \in \text{NULL} \rightarrow R_i = A$$

$$\text{If } R_i \in \text{Invalid} \rightarrow R_i = 47$$

Figure 17 - Invalid Revision Characters Conditional Statements.

In this figure, the conditions for an invalid revision are shown. If a revision, R_i, is a subset of the NULL set or the Invalid set (such as α), these entries would not be acceptable, so they are replaced with A and 47 respectively. These methods to overcome invalid revision characters are simple but are universal and can be applied to other PLM systems.

3.7.4   Manual Data Validation

The majority of the issues mentioned in the three previous sections can be resolved via automated algorithms. Occasionally, there is a case which requires the user to manually fix an error. The frequency of a manual item or revision fix will be more thoroughly described in the results section.

The goal of a manual data validation is to raise the import success to 100 percent for all associated artifacts. Data validation serves as an excellent method to account for all imported

items and ensure that a reliable import process has occurred. A successful data validation stage

will result in a report indicating any errors that occurred during the import. The specific

implementations of data validation will be discussed in chapter 4, as they relate to a specific

PLM software package.

4 Implementation

This section will show implementation of the methods described in Chapter 3. A discussion of several algorithms will be presented, along with the steps required to complete the artifact import. The following items, associated with the implementation of the AAIT, will be outlined in this section.

- PLM application programming interface (API)

- Programmatic Authentication

- Item Import

- The ITK Code Generation Tool

- Workflow instantiation - Revision Release

- Data Validation

- Error Handling

4.1 PLM Application Programming Interface (API)

The first step of implementing the AAIT is learning how to use the API associated with the PLM system. These learning steps may be easily applied to other PLM programs if they allow customization via an API. PLM software such as Teamcenter, Windchill and ENOVIA have their own API's with supporting documentation to help users get started.

An API is a programming environment that has function calls which, among other things, allows the user to programmatically perform all the steps required to import artifacts without the need for user input. This research will focus on Teamcenter's API which is called the Integrated Tool Kit (ITK). The ITK has documentation containing functions, modules, and examples. The user must understand the structure of this documentation in order to implement the API successfully. An excellent way to familiarize oneself with the documentation is to try to create a simple program. For example, attempt to display "hello world" by implementing the most basic functionalities within the API. Once a simple program is created, this program serves as the template which will be used in more complex programs. This template is vital in applying the ITK in this research because it serves as a starting point. Ultimately it will evolve into an implementation which will save valuable time by eliminating the need to process operations that would otherwise be required, such as mouse clicks and un-needed key strokes. In an import, thousands of operations are eliminated via the use of the API code. By eliminating redundant operations, PLM automation serves as a mechanism to leverage the proposed benefits of PLM, to reduce development time and cost.

4.2   Programmatic Authentication

The ITK is divided into sections called modules. Modules are divisions within the API to distinguish sets of functions. For example, the ITEM module is used to distinguish function calls related to the storage and modification of items. Examples of functions inside the ITEM module are: create_item, delete_item, and update_item. Modules allow the API documentation to be organized so the user can quickly find the function needed for a desired operation. Valuable time

50

in automating a process can be saved by knowing where to look for the required functions within a given API.

The first modules used in authentication are ITK and POM. These modules have functions which initialize the connection and log the user into the PLM server. The PLM system must first be accessed by entering login information before any other operations can be performed.

```
//----------------------------------------
// Function to provide ITK login
//----------------------------------------

int login(void) {

        //IJ_ERROR (ITK_initialize_text_services (ITK_BATCH_TEXT_MODE) ,"",0);
        IJ_ERROR (ITK_auto_login () ,"log in",0);
        IJ_ERROR (POM_set_env_info(POM_log_sql_switch, true, 0, 0, NULLTAG, (char*)NULL) ,"",0);
        return 0;
}
```

Code Fragment 1 - Programmatic Authentication

This login function shows several steps which must be called in order to programmatically authenticate into Teamcenter. First the function ITK_initialize_text_services is called which initializes batch text mode. The next function ITK_auto_login, initiates the log in to the ITK, and finally the PLM_set_env_info sets environment parameters which are needed to modify the session. Once connected programmatically to the PLM system, the artifacts can be uploaded via ITK function calls.

4.3   Item Import

This section contains the functions used to import items into Teamcenter, along with the associated datasets. These algorithms are rather simple and call ITK functions in the ITEM

module. In order to import an item via the ITK, we must first check to see if the item already

exists in Teamcenter. The following code fragment demonstrates how to do this.

```
IJ_ERROR( ITEM_find_item (item_id,&the_item),      "check if item exists",1 );
if(the_item) {
        tag_t *rev_list;
        int count;
        IJ_ERROR (ITEM_list_all_revs(the_item, &count, &rev_list),
        "ask all revs of an item",1);
        int i=0;
        bool hasbeenfound = false;
        for (i = 0; i<count; i++)
        {
                char rev_id[ITEM_id_size_c +1];
                ITEM_ask_rev_id(rev_list[i],rev_id);
                fprintf(revreport,"%s\t",rev_id);
                }
        }
fprintf(revreport,"\n");
fclose(revreport);
return 0;
```

Code Fragment 2 - Find Item and Revisions Prior to Import

The code fragment will scan the database for all items with the same name as the items

that are being imported. If an item with the same name is found, the ITEM_list_all_revs

function will list all the revisions contained inside that item. The results of this query are then

printed to a file. The fclose function then clears up any memory associated with the revision

report.

Now that the database has been queried for all revisions associated with an imported

item, the imported revision must be compared to all revisions associated with a given item inside

the database. The implementation of algorithm is shown in Code Fragment 3.

```
IJ_ERROR (ITEM_list_all_revs(the_item, &count, &rev_list),       "ask all revs of an item",1);
int i=0;
bool hasbeenfound = false;
for (i = 0; i<count; i++)
char rev_id[ITEM_id_size_c +1];
ITEM_ask_rev_id(rev_list[i],rev_id);
printf("\tTeamCenter has REV ID: %s Count %d of %d\n", rev_id,i+1,count);
if(strcmp(rev_id,item_rev) == 0)
{
        hasbeenfound = true;
}

if(!hasbeenfound){
        HANDLE hdnl = GetStdHandle(STD_OUTPUT_HANDLE);
        SetConsoleTextAttribute(hdnl,FOREGROUND_RED | FOREGROUND_INTENSITY);
        printf("\t Warning:  Need to add revision %s to the database\t\n", item_rev);
        SetConsoleTextAttribute(hdnl,FOREGROUND_RED | FOREGROUND_GREEN |
        FOREGROUND_INTENSITY| BACKGROUND_BLUE );
        tag_t dataset = NULL_TAG;
        IJ_ERROR (ITEM_create_rev(the_item, item_rev, &new_rev),"create a revision",1);
        IJ_ERROR (import_to_dataset(dataset_type,item_id, item_id,ds_ref_name,file_path, 1, SS_BINARY,
        tool_name, &dataset),"Import dataset",1);
        IJ_ERROR (ITEM_attach_rev_object(new_rev,dataset, ITEM_specification_atth),
"attach file to rev",1);
        printf("Created a new revision.\n");
}
```

Code Fragment 3 - Import Revision Verification

In this code a Boolean, "hasbeenfound" is used to indicate if the revision has been found

inside the PLM database. If the revision is not found, then a warning is shown to the console

which indicates that a new revision will be imported. The algorithm then creates the revision

and imports the file into the dataset. It is important that the revision is created before the file is

imported, otherwise the artifact cannot be imported. Once this is done, the revision is attached to

its corresponding item.

Once the item has been imported, a status report is generated. In the status report, each

artifact is displayed along with its revision, dataset and file type. This report ensures that all

artifacts have been stored correctly. Table 4 shows an example of this artifact import report**.**

Table 4 - Error Checking Tabular Layout

Item #	Revision	Dataset	File Type	Status
12345	A	12345-B.PDF	PDF	**Uploaded**
12345	B	12345-B.PDF	PDF	**Dataset Already Exists**
12345	C	12345	N/A	**File type Invalid**

These error checks are shown in real-time, as output on the screen, as shown in the status column in the table above. The ability to catch errors is an essential step in PLM import automation. Prior to PLM automation, error checking was much more difficult. For example, the user would be required to go into the PLM system and refresh the import directory to see if the item had been imported. With API function calls, a request may be issued to check if the item has been stored under the correct item name. This is a quick validation step to ensure that items are imported correctly and as a result, is a valuable time saving measure.

The checks mentioned in this section verify that items uploaded successfully in the PLM system. The item number must be evaluated to ensure that it is not already stored in the database. If the item number already exists, a collision can occur. If starting from a fresh PLM deployment, collisions are typically not an issue. In this study, code to expect collisions was created due to the fact that the test environment contained many existing items.

As the reader will see in the results chapter, the automated import and validation code are the major contributors to improved efficiency in the artifact import process. After the item import, artifacts are ready to be directed via release workflows to the appropriate parties involved.

4.3.1 The ITK Code Generation Tool

Before continuing to the next step in the artifact import process, it is important that the reader both appreciate and understand an important tool which was vital in the item import implementation. The ITK Code Generation Tool (ITKgen) is a tool which was used to make this ITK implementation successful by drastically reducing the time required to implement functions from within the ITK. The ITKgen was primarily developed by Jon Lund while working as a graduate student in C. Greg Jensen's ParaCAD Research Lab at Brigham Young University.

The creation of automation algorithms can be simplified by implementing code generators within Teamcenter's API. Code generators, such as the ITKgen, are used to copy non-working functions from a structured documentation source and output working code. This is done by means of parsing the documentation and extracting the necessary components. In order to parse the copied function, a program is executed which parses the contents of the clipboard and replaces the contents of the clipboard with useful code.

The code generator works ideally on the ITK documentation because it has been well structured by developers at Siemens. An example of a well documented function that was used commonly in the item import code is "Create_Revision". The documentation for this function is shown in Figure 18.

Using the code generator on this function will return a the following code to the clipboard "Create_Revision(rev, &created_revision)". This returned function is then directly inserted into the surrounding algorithm saving valuable programming time.

```
Function: Create_Revision

Inputs: rev – revision to be imported (type – char)

Outputs: created_revision – the tag of the revision which has been created

Syntax: Create_Revision (char rev, tag_t &created_revision);
```

Figure 18 - Sample Documentation of a Function

The principles behind the ITK code generation tool are broadly applicable. Code generation algorithms will vary from each application programming interface, but the principles of the algorithms remains the same. The ability to take code documentation and turn it into useful code can reduce development time dramatically.

Examples such as these have been tested on other engineering API environments such as UGOpen, an NX CAD programming interface. It has proven to be a useful tool when developing in an API which has been well documented. As a result of this tool, valuable time and resources are saved when preparing automation code. This technique is in effect automating the automation of engineering tools.

4.4   Workflow Instantiation - Revision Release

Moving the topic back to the artifact import process, once the item has been imported it must be released by calling a workflow. It may be necessary to release the revisions because of the "maximum unreleased revisions" issue mentioned in Chapter 3. There is another reason that artifact revisions must be released in PLM. This is because even though the artifact may be in a released state outside of PLM, the database does not reflect that status until the artifact revision is released via a workflow. Thus a released status must be emulated from within Teamcenter by

calling a revision release process workflow. By releasing an item, the program is then able to add

additional revisions to Teamcenter's Database.

In order to release the revision, a workflow is called directly from a function call. The

code to call a process or workflow from within the ITK consists of several steps. First, a template

workflow must be created interactively. This only needs to be done once. The figure below

shows a "zero-step" release workflow called from the code, which releases the revision of an

item once it has been uploaded.

Figure 19 - Sample Release Workflow

The second step is to find the zero step process template programmatically. Once the

template is found, a new process is created from the existing workflow template. A process will

be created for each artifact as it is imported. In this way, new process workflows are generated

automatically through the Teamcenter API.

As seen in Code Fragment 4, workflow creation is handled through the EPM module in

the Teamcenter API. The workflow must first be found via the EPM_find_template function then

the process is created using the EPM_create_process function. As a result of this code, the steps

required to manually create a workflow instantiation and process are eliminated.

```
EPM_find_template ("Zero-Step Release", PROCESS_TEMPLATE,

&process_task_template), "find_template tag",0);



EPM_create_process("process name", "process description",

process_task_template, 1,attachments, attachment_types,

&new_process),  "create_process",0);
```

Code Fragment 4 - Finding and Calling a Workflow Process From the ITK

By automating the routing of product data to the appropriate parties involved, the

automated import tool saves additional time and improves data synergy and collaboration by

utilizing the full capabilities of PLM. Because of this, all engineers and employees responsible

for the imported artifacts are automatically notified that their data is now located in a central

repository.

4.5   Data Validation

Once the item has been released, the data must be validated in order to ensure the item

has been imported into the PLM database. To do this, the automated process of import and

revision management was applied to an existing PLM installation to simulate any item collisions

or errors that would happen in a real-world environment.

To identify errors in real time, error codes were shown on the screen, allowing the import

administrator to validate the imported artifacts. Some of the errors which can occur in an

automated import process are listed below:

- Maximum un-released revisions reached for an item

- Item number unavailable / Naming Collision

- Invalid revision character

As items were imported into Teamcenter, several error handling methods were used to overcome these issues. The figure below demonstrates an example of an error which can occur in a manual import as the maximum un-released revisions have been reached.
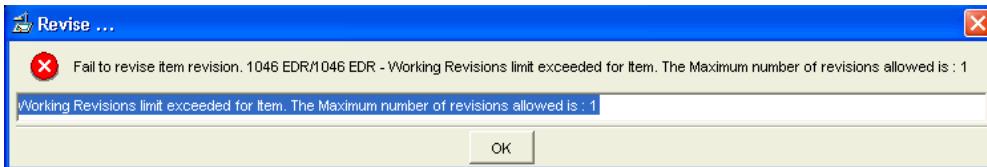

Figure 20 - Teamcenter Error of Having Maximum Revisions Reached

In contrast, when this error occurs in a programmatic import using the AAIT, the error is reported to the item report summary screen (shown in table 3 of section 4.3). Thus there is no error window which requires user interaction that halts the progress of the import. Rather the errors are simply collected for review, and code moves on to the next artifact or revision.

4.6 Error Handling

Efforts to perform error checking are essential to the success of the AAIT because it serves as a troubleshooting measure during the import validation stage. The lack of an error checking mechanism, in a program such as the AAIT, results in wasted time and resources during an artifact import.

As an example, we will use ITK function wrappers to demonstrate the concept of error checking to save time. Function wrappers are used to avoid crashes within Teamcenter. Additionally, they are used to catch and troubleshoot error codes native to Teamcenter. An example of an error code function wrapper is shown in the code fragment below.

IJ_ERROR(ITEM_list_rev_attachs_of_type (rev, ITEM_spec, &count,

&objects),"get attached item of type",0);

Code Fragment 5 - Error Code Function Wrapper

In this case, the error checking function wrapper is "IJ_Error." This code would be able to return a list of errors from a command prompt which indicates to the user any problems which occurred corresponding with the ITEM_list_rev_attachs_of_type function. Some potential errors of this function could be "invalid input type" or "revision attachments not found". For example, an error message from the IJ_Error function wrapper for "invalid input type" would look similar to what is shown in the figure below.

By producing errors which clearly indicate the status of a function to the user, debugging is greatly simplified. Because of this, error checking must be used extensively in creating an item, revision, and datasets to indicate errors such as item collisions. By doing this, error handling measures improve the security and reliability of an automated import. Thus, the viability of an automated import is reliant on the ability to catch potential sources of error.

```
**************************************************

Function Error "ITEM_list_rev_attachs_of_type" –Code 12474

Attempt to "get attached item of type" did not successfully complete

Cause – Invalid Input type into function

**************************************************
```

Figure 21 - Results of Error Code Function Wrapper

4.6.1   Status Codes

Status codes are delivered to Microsoft Excel showing which items have been imported

successfully and which items failed. A group of status codes indicating that the item has been

successfully imported are shown in the boxed column "F" in Figure 22.

Figure 22 - Status Codes

Each status code represents the degree to which the import was successful for a given

item. In the figure above, the status code 999 describes an item and revision which has been

uploaded successfully. The status codes are updated in real-time as the AAIT is working. To do this, the AAIT updates a column within Excel via COM automation which allows the user manipulate data within a spreadsheet programmatically. In this way, a user can visually check the spreadsheet for any non-successful-codes (codes that are not 999) to ensure that all artifacts have been imported correctly. Alternatively, if the user does not want to watch the import as it is happening, he can perform a search for any codes that are not 999 at a later time.

As described previously, this method of data validation is a combination of implementing programmed codes that are created in this research, and native codes that come from Teamcenter. By implementing both types of codes, validation data is delivered to a spreadsheet in real-time and potential issues are flagged that can be resolved in an efficient manner.

5   Results and Discussion of Results

Several test cases were performed to evaluate the reliability, efficiency, and security of the AAIT (Automated Artifact Import Tool) in a real-world environment. This chapter will describe three test cases to show what improvements the AAIT has over an interactive artifact import.

5.1   Test Case 1 - AAIT Reliability

To evaluate the reliability of the AAIT, the metric chosen was the percentage of items successfully imported. This metric was chosen because the selected successful import goal for this test case was 95%, meaning that 95% of the artifacts imported successfully. This metric also implies that the user must know precisely which artifacts failed and why. By finding out which items failed in the import, the user can manually correct the problems later.

To set up the test case, items were broken into several categories depending on how they responded to the import. The categories are shown in the list below.

- New Item and New Revision

- Item/Revision Already Exist

- Item Existed but New Revision

- Item # Unavailable

- Other Failure

Five folders were selected containing a suitable sample of artifacts. The results of the import test case are shown in table 4. The table shows five folders with corresponding number of files in each folder. The next three columns (New Item & New Revision, Item/Revision Already Exist, and Item Existed but New Revision) indicate artifacts which were imported successfully. The two red columns (Item # Unavailable / Other Failure) show which artifacts did not import correctly.

Table 5 - Test Case Results

Folder	Number of Artifacts in Folder	New Item & New Revision	Item/Revision Already Exist	Item Existed but New Revision	Item # Unavailable	Other Failure	Success Rate (Programmatic)
1	1371	474	884	13	0	0	100.00%
2	159	99	42	12	1	5	96.23%
3	363	363	0	0	0	0	100.00%
4	13	7	0	2	4	0	69.23%
5	244	135	2	29	32	46	68.03%
Total	2150	1078	928	56	37	51	95.91%

Table 5 shows that the total success rate of the programmatic import was 95.91%, which exceeded the goal to have 95% of the artifacts import successfully. The remaining 4.09% required manual interaction to complete the import. This was done by using the aforementioned

error checking methods in chapter 3 and 4, such as preparing the item names with prefixes. Once this was done the import success reached 100%.

The validation test also gave great insight as it showed which errors are commonly involved with an automated import. Most common errors which occurred were item number unavailable, invalid revision, and maximum revision reached.

These errors were identified with the aid of status codes which specifically listed which items needed to be manually corrected in order to achieve 100% import quality. The use of status codes proved instrumental in helping the user to better utilize the AAIT to achieve the research objective: to perform a reliable, efficient, and secure automated import.

5.2   Test Case 2 - AAIT Efficiency

The metric chosen for efficiency for the AAIT was the amount of time required to import a specific amount of artifacts. This metric allows the reader to compare the time required to import artifacts for both the automated method and a manual import.

As mentioned before, efficiency of the AAIT depends greatly on the extent to which the data has been filtered and organized prior to import. If established procedures are in place to document artifact attributes (cost, material, model parameters, etc.) in a log prior to PLM import, the AAIT efficiency increases greatly. As a result, correct metadata management reduces manual interaction required in the process. By reducing the amount of human interaction, the efficiency gains explained in this section are realized.

A timed test was taken to measure the performance of the programmatic method versus an interactive approach. Both methods were performed using the same operations of importing and releasing items into Teamcenter. A difference between the two methods was that the

interactive method required the utilization of a 3 Tier web-based remote connection whereas the

AAIT was able to run directly on the PLM server that was initiated from a remote terminal.

Thus the programmatic method resulted in a substantial improvement in time when compared to

a manual import.

The Interactive method required several student test subjects who had been trained prior

to the test. Each subject performed a run on 10 artifacts, and each run involved a different group

of data. The time to perform each process was recorded and shown in Graph 1.
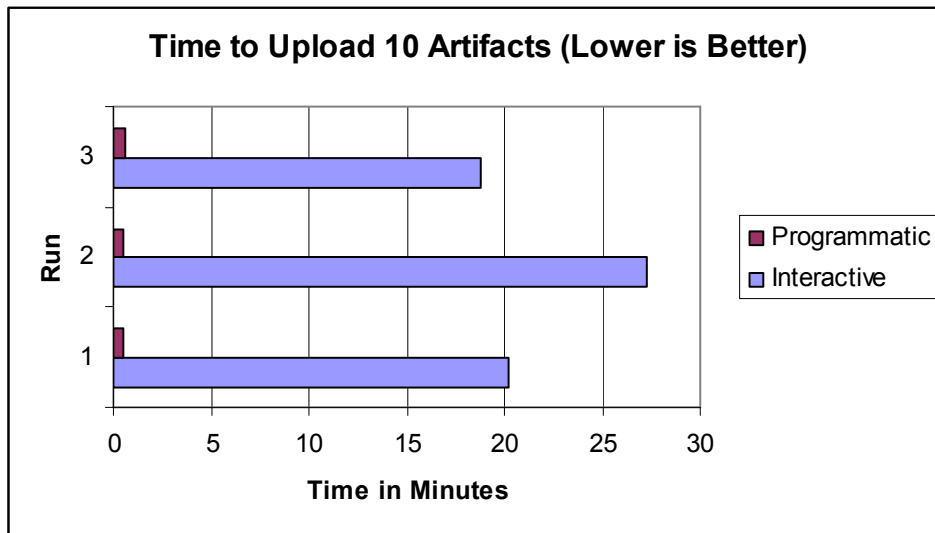
Graph 1- Timed Results (programmatic times averaged 30 seconds)

As seen in Table 3, the automated method is on average 45 times faster than the manual

import. This data can be projected out to a larger scale import of 10,000 product artifacts. With

10,000 artifacts, the programmatic method would take a little over a day of computation time,

whereas the interactive method would take over 45 work days of computation time and user

interaction.

This test was performed over the internet, connecting to a remote server. The time

required to complete run two interactively is noticeably greater than the other runs. This is

because bandwidth became a limitation as the connections that made up the 3 Tier PLM structure slowed down. This validated the assumption that efficiency gains were possible by implementing an automation algorithm, as stated in the thesis objective.

Bandwidth limitations, when combined with user interactions that require a lot of server requests, contributed to the reduced efficiency when performing the import interactively. The quantification of these factors, in terms of computational cost, is explained by research done by Leong and Lee. [7] By reducing the amount of server requests in the automated method, a considerable amount of required bandwidth is reduced. The reduced bandwidth requirements, combined with the improved speed of the algorithm provided dramatic time savings over a manual artifact import.

5.3   Test Case 3 - AAIT Security

As a result of the automated import, many security issues were alleviated. The test case showed that issues caused by human error were reduced. These security issues, such as accidentally moving folders, deletion of product artifacts, and accidental renaming of artifacts, were eliminated. The metric for improved security of PLM was that the artifacts were able to be securely stored without the aforementioned issues.

The resulting structure of a secure import indicates that all revisions associated with an item have been successfully imported. Figure 23 shows this secure structure for an item containing multiple revisions with multiple datasets.
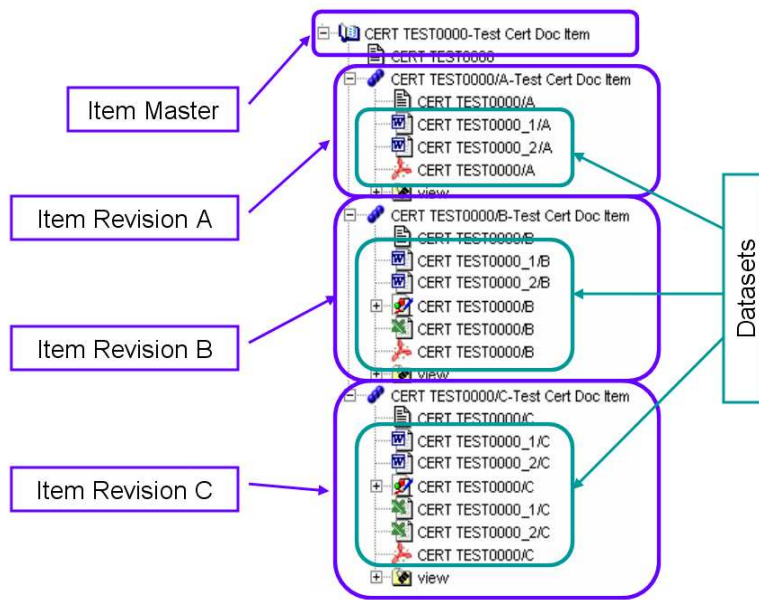
Figure 23 - Imported Structure of artifacts after Automated Import

The AAIT, however, did create some complications during the import. These complications involved the releasing of CAD data. When product artifacts contain associated CAD assemblies, the links to all parts within the assemblies must be maintained. When the AAIT imported and revised certain CAD parts, some of these links were broken. This resulted in revised assemblies that did not show all of the parts within the assembly. This problem presented itself after the AAIT was created. In order to fix the problem, custom workflows are needed to copy over the existing datasets to the new revision as the workflow is called. Thus, the links to all CAD parts within an assembly can be maintained and no security violations occur. This fix could be implemented in future versions of the AAIT.

6 Conclusions

The results shown in the test cases give a clear picture that the AAIT was able to perform a reliable, efficient, and secure automated import. It has also been established that the methods described in this thesis can be applied to other PLM systems due to the similarities which exist in PLM of a RDBMS (Relational Database Management System), File Server, and API. In this conclusion, a summary of the AAIT will be evaluated in terms of each category mentioned (reliability, efficiency, and security). Recommendations will also be given for future work in this area.

6.1 Improved Reliability

User interaction reduces the level of consistency while importing product artifacts into PLM. During the validation test, user interaction was attributed to errors in artifact attributes such as description, revision, and occasionally the filename. Because of this, the interactive method had lower reliability than the programmatic method. Unlike an interactive import, the programmatic method contained error checking, which ensured that the artifact was handled in a consistent fashion.

Automating release workflows aid in the security of process modeling associated with a product artifact. This ensures the artifact goes through the appropriate gates before it is released.

The method of calling workflows programmatically improves upon PLM reliability as it establishes standards for process control.

6.2   Improved Efficiency

The efficiency of the automated artifact import process outlined in this research depends greatly on gathering and organizing the appropriate artifact data. Once the data is organized, the programmatic method is used to import artifacts into PLM. Import time greatly decreased, making it more cost effective than previous methods. The results of the test case showed that an automated method for importing legacy data is 30 to 60 times more efficient, depending on the interactive steps required. The results of this method have been shown to local companies near Brigham Young University. Some of these engineering companies have expressed interest in the application of this program as an effective method to quickly store their legacy project data.

It is important to note that lack of prepared data hinders the efficiency of this method. If associated metadata is not correctly prepared or has not been logged, efficiency of the AAIT decreases. It is only when this data has been well managed prior to importing that time-saving benefits of the AAIT are realized. Companies with well established procedures concerning the management of metadata will benefit greatly by the increased efficiency of the AAIT algorithm.

6.3   Improved Security

Importing artifacts into a database component provided improved security over managing artifacts outside PLM. The improvement in security is not quantified as easily as reliability and efficiency. However, artifacts managed outside of PLM are subject to incorrect user rights administration, which allows any users to accidentally drag, misplace, or delete a folder. Once

the artifacts are misplaced, it can take a considerable amount of time to recover the archived data. In this way, PLM remedies the security issues involved with artifact storage and management. The AAIT takes PLM security to the next level by reducing human interaction and therefore reducing error.

There were some shortcomings to the AAIT with regards to CAD assembly integrity. Even when considering these shortcomings, the overall security of the process was increased over a manual import method.

6.4   Future Work

Additional work performed would be considered an extension or improvement upon the research proposed in this thesis. The areas of future work which have been considered are listed below.

- Utilizing Knowledge Based Attributes to Direct Workflows

- Managing Multi-Document Engineering packages within PLM

- Utilizing AAIT for CAD assemblies

- The Big Picture

6.4.1   Utilizing Knowledge Based Attributes to Direct Workflows

In order to take full advantage of knowledge based engineering, attributes passed into PLM can be used to direct workflows. Attributes such as artifact type, project type, and description could be used to determine where the dataset is routed. This would require that the associations of each custom process be programmed into the AAIT. This would not be a universal solution for every PLM deployment, rather templates could be developed which give

the capability to create programmatic routings. These templates could be called from a user interface which is created to manage attribute-workflow associations.

The results of associating knowledge based attributes with workflows would save time as the design nears completion. Benefits of this research could also be linked to the ability to direct workflows associated with ERP and supply management tasks.

6.4.2   Managing Multi-Document Engineering Artifacts Within PLM

PLM multi-document management requires complex action handlers and rules in order to be successful. At the moment, it is common for CAD assemblies to be managed inside PLM. However, management of a multi-document engineering package, containing several Microsoft Word documents, images, and Microsoft Excel files inside PLM, is not common in industry. Tools, such as the mail-merge function in Microsoft Word, may be used to create new template documents as a workflow is instantiated. Associations between documents in these packages are maintained similarly to the associations which are found in CAD assemblies. As the sections in a given document are revised, the PDF document must also be updated.

These types of large, multi-section documents are commonly found within the aerospace industry, as designs are sent to the FAA for approval. Initial designs must be approved by the FAA, as well as designs which are reverse engineered. Section 4.3 discusses how the security of revisions, with multiple documents, should be handled within the scope of this thesis. Workflows have been introduced which allow for selective dataset maintenance. These are steps in the right direction, but do not address all the issues involved with the management, release, and the electronic signing of multi-document PLM documents. No research has been found which addresses all the aforementioned issues associated with the management of these document artifacts.

6.4.3 Utilizing AAIT for CAD Assemblies

Third party tools are available for PLM systems, such as Teamcenter, to import assemblies from various CAD systems. These tools require each assembly to be manually imported from CAD. A company may have thousands of assemblies, each with multiple parts. The manual process of importing Solidworks assemblies into Teamcenter was tested as an example. Using a tool, provided by Siemens Teamcenter experts, each assembly required at least twelve steps to be successfully vaulted into the PLM RDBMS. Additionally, all the metadata did not transfer over into the PLM system. Attributes, such as revision, stored in the CAD data did not transfer over automatically. An automated import of assemblies, which carries over associated attributes, would be extremely helpful according to a company surveyed in this research. In this case, the alternative to importing all legacy assembly CAD data into PDM/PLM, was to import only the released PDF versions of each part or assembly belonging to a project. Research for an automated import of CAD assembly data is currently lacking and does not address all of the present issues.

6.4.4 The Big Picture

The tool described in this thesis accomplishes the reality of a "Singular Data Source" which can be extended to other areas in product development. The data that the AAIT uses may be scattered throughout a company in the form of CAD data, Analysis Results, Test Data, or Documentation. The objective of the research described in this thesis is to gather this scattered data into one source, PLM. This source can then be utilized by other divisions within a company. For example, Figure 24 shows how the scattered data can be passed through PLM and then utilized in an ERP system such as SAP or Oracle.

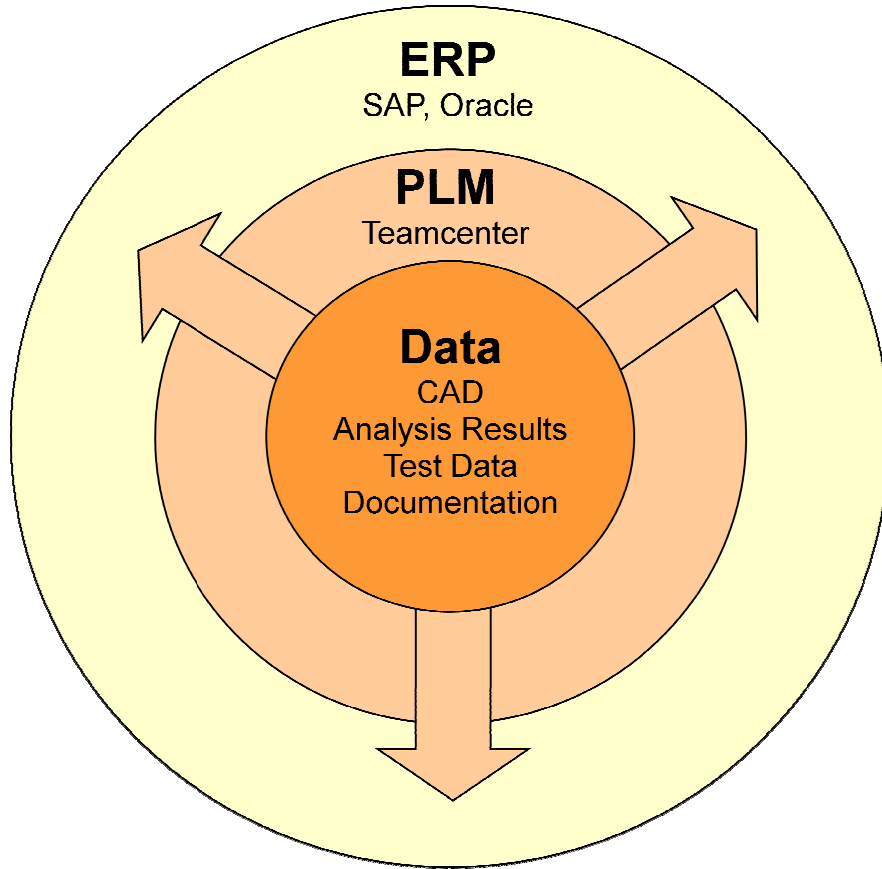**Figure 24 - Knowledge Flow, From Source Data to PLM and ERP**

By passing valuable information to ERP that is relevant to a given product, post-design

tasks such as managing a Bill of Materials and supply chain management are greatly simplified.

The value of this research extends throughout the product development cycle adding value to

business and engineering tasks that will save companies time and money.

References

[1]   Lund, J., Fife, N., Jensen, C., (2005). "PLM-Based Parametrics for Design Automation and Optimization", *Computer-Aided Design and Applications*, Vol. 2, Nos. 1-4, CAD'05.

[2]   Xu, X. William, Liu, Tony (2003), "A Web-Enabled PDM System in a Collaborative Design Environment", *Robotics and Computer-Integrated Manufacturing*, vol. 19, pp. 315-328.

[3]   Leong K.K., Yu K.M., Lee W.B. (2002). "Product Data Allocation for Distributed Product Data Management System", *Computers in Industry*, vol. 47, pp. 289-298.

[4]   Chu, Xingjun, Fan, Yuqing (1999). "Product Data Management Based on Web Technology", *Integrated Manufacturing Systems*, vol. 10, pp. 84-88.

[5]   Philpotts, Mike (1996), "Introduction to the concepts, benefits and terminology of product data management", *Industrial Management and Data Systems*, vol. 96, pp. 11-17.

[6]   Wang, Ying Daisy; Shen, Weiming; Ghenniwa, Hamada (2003). " WebBlow: a Web/agent-based multidisciplinary design optimization environment ", *Computers in Industry*, vol. 52, pp. 17-28.

[7]   Liu, D. Tony, Xu, X. William (2001). "A Review of Web-Based Product Data Management Systems", *Computers in Industry*, vol. 44, pp. 251-262.

[8]   Xu, X., William, Liu, Tony (2003). "A Web-Enabled PDM System in a Collaborative Design Environment", *Robotics and Computer-Integrated Manufacturing*, vol. 19, pp. 315-328.

[9]   Koonce, David, Department of Industrial Engineering, Ohio (2003), "A Hierarchical Cost Estimation Tool", *Computers in Industry*, vol. 50, pp. 293-302.

[10]   Lund, J., Fife, N., Jensen, C. (2004), "PLM-Based Parametrics for Design Automation and Optimization", *Computer Aided Design*, vol. 2, Nos 1-4.

[11]   Sudarsan R., Fenves S.J., Sriram R.D. (2005). "A product information modeling framework for product lifecycle management", *Computer-Aided Design*, vol.37, pp. 1399-1411.

[12]   Roach, Gregory M. (2003), *The Product Design Generator--A Next Generation Approach to Detailed Design*, Brigham Young University

[13]   Cochran, Jeffery K., Hong, Suck-Chul (2004). "An Automated Relational Database Structure Building System for the Life Cycle of Manufacturing Simulation Models", *International Journal of Computer Applications in Technology*, vol. 19, pp. 53-65.

[14] Barbosa, C.A.M., et. al. (2002). "An Object Model for Collaborative CAD Environments", *Proceedings of the International Conference on Computer Supported Cooperative Work in Design*, vol. 7, pp. 179-184.

[15] Peng, Ting-Kuo, Trappey, Amy J. C. (1996). "Cad-Integrated Engineering-Data-Management System for Spring Design", *Robotics and Computer-Integrated Manufacturing*, vol. 12, pp. 271-281.

[16] Baker, Tyson J. (2004). *Attribution Standardization for Integrated Concurrent Engineering*, Brigham Young University.

[17] Bowland, N.W., Gao, J.X., Sharma, R. (2003). "A PDM- and CAD-integrated Assembly Modelling Environment", *Journal of Materials Processing Technology*, vol. 138, pp. 82-88.

[18] Carroll, Michael P., Hawkins, Christopher M. (2002), *Web Based Analysis*, ACM Digital Library.

[19] Oh, Youchon, Han, Soon-hung, Suh, Hyowon (2001). "Mapping Product Structures Between CAD and PDM Systems Using UML", *Computer-Aided Design*, vol. 33, pp. 521-529.

[20] Gao, J.X., Aziz, Hayder, et al (2003). "Application of Product Data Management Technologies for Enterprise Integration", *International Journal of Computer Integrated Manufacturing*, vol. 16, pp. 491-500.

[21] Lund, Jonathon (2006). *The storage of Parametric Data in Product Lifecycle Management Systems*, Brigham Young University.

[22] Fife, Nathaniel L. (2005). "Embedding Process Integration and Design Optimization within Product Lifecycle Management", *Brigham Young University Master's Thesis*.

[23] Lihong Qiao, Yizhu Zhang (2007), "Engineering Change Process Modeling and Control for Product Lifecycle Management", *International Conference on Comprehensive Product Realization 2007*.

[24] Giullian, N.C., Berglund, C. and Jensen, C.G. (2007). "Applying PLM to Global Collaboration Projects", *Product Lifecycle Management book*, Milan, Italy.

[25] Day, Martyn (2002). *What is PLM*. Cad Digest.

[26] Hill, Sidney (2006). *A winning strategy*. Manufacturing Business Technology.

[27] Hill Sidney, Jr., (2008) *How to be a Trendsetter: Dassault and IBM PLM Customers Swap Tales from the PLM Front*, MSI.

[28] CIMDATA INC (2003). *PLM to PDM: Growth of an Industry*, CIMDATA INC.

[29] Johnson R. H., (1986). "Product Data Management - with Solid Modeling". *Computer-Aided Engineering Journal*.

[30] Kaiser, G.E., Barghouti, N.S., Feiler, P.H., Schwanke, R.W. (1988). "Database support for knowledge-based engineering environments", *IEEE Expert,* vol.3, pp. 18-23, 26-32.

[31] Grieves, Michael, (2006) *Product Lifecycle Management*, McGraw Hill

[32] Kantardzic, Mehmed (2003). *Data Mining: Concepts, Models, Methods, and Algorithms*. John Wiley & Sons.

[33] Fayyad, Usama, Gregory Piatetsky-Shapiro, and Padhraic Smyth (1996). "From Data Mining to Knowledge Discovery in Databases" *AI Magazine*.

[34] Harper, Gavin; Stephen D. Pickett (2006). "Methods for mining HTS data". *Drug Discovery Today* 11 (15-16): 694-699.

[35] Clark, Nicola (2006). "Airbus replaces chief of jumbo jet project*". International Herald Tribune*. Retrieved 2006-09-16.

Appendix A.   Automation Source Code

```
//---------------------------------------
// Function to provide ITK login
//---------------------------------------

int login(void) {

        //IJ_ERROR (ITK_initialize_text_services (ITK_BATCH_TEXT_MODE) ,"",0);
        IJ_ERROR (ITK_auto_login () ,"log in",0);
        IJ_ERROR (POM_set_env_info(POM_log_sql_switch, true, 0, 0, NULLTAG, (char*)NULL) ,"",0);
        return 0;
}
/*
*/
//---------------------------------------
// Single import function
//---------------------------------------

int import_to_dataset (
        char* ds_type,
        char* ds_name,
        char* ds_desc,
        char* refname,
        char* os_paths,
        int nfiles,
        int binary,
        char* tool_name,
        tag_t *new_dataset)
{
        int retVal = import_to_dataset (ds_type, ds_name, ds_desc, refname,
                &os_paths, 1, binary, tool_name, new_dataset);
        return retVal;
}


int jimport_named_ref(tag_t dataset, char* refname, char* os_path, int binary) {

        tag_t                           file_tag;
        IMF_file_t              file_descriptor;
        char buf[1024];
        printf("Checking if file \"%s\" exists...", os_path);
        FILE* f=fopen(os_path,"r");
        if(f==NULL) {
                printf("Nope.\n");
                return -1;
        } else {
                printf("Yes.\n");
                fclose(f);
        }
        char* token = strstr(os_path,".");
        if(token!=NULL && strstr(token+1,".")!=NULL) {
// handle problems arising from "."
                char cmd[1024];
```

77

```
                    sprintf(cmd,"copy \"%s\" h:\\tmp.pdf",os_path);
                    system(cmd);
                    strcpy(os_path,"H:\\tmp.pdf");
            }
            IJ_ERROR( IMF_import_file(os_path, NULL, binary,&file_tag,&file_descriptor),
"import the file",0);
            IJ_ERROR( AOM_save(file_tag) ,"save",0);
            AE_reference_type_t mine;
            IJ_ERROR( AE_add_dataset_named_ref(dataset,refname,AE_ASSOCIATION,file_tag),
"add to dataset",0);

            return 0;
}
//-------------------------------------------------------------
// Multiple import function (this actually uploads the file)
//-------------------------------------------------------------

int import_to_dataset (
            char* ds_type,
            char* ds_name,
            char* ds_desc,
            char* refname,
            char** os_paths,
            int nfiles,
            int binary /*SS_BINARY*/,
            char* tool_name,
            tag_t *new_dataset)
{
            tag_t       dataset_type        = NULL_TAG;
            tag_t tool = NULL_TAG;
            char        dname[WSO_name_size_c+1];
            char        ddesc[WSO_name_size_c+1];
            char        dtype[AE_datasettype_name_size_c+1];
            const char*         dataset_id;
            const char*         dataset_rev;
            strcpy(dname,ds_name);
            strcpy(ddesc,ds_desc);
            strcpy(dtype,ds_type);

            IJ_ERROR ( AE_find_datasettype ( dtype, &dataset_type ),"",0);

            //Deprecated function as of TC 7.0
            //IJ_BERROR ( AE_create_dataset(dataset_type, dname, ddesc, new_dataset),
"create new dataset",0 );

            IJ_ERROR ( AE_create_dataset_with_id(dataset_type, dname, ddesc, dataset_id, dataset_rev, new_dataset),
"create new dataset", 0);
            for(int i=0;i<nfiles;i++) {
                    char cwd[1024];
                    memset(cwd,0,sizeof(cwd));
                    getcwd(cwd,sizeof(cwd));
                    uprintf("Uploading file \"%s\" (curdir is: %s)\n", os_paths[i]+2, cwd);
                    IJ_ERROR ( jimport_named_ref(*new_dataset, refname, os_paths[i]+2, binary),
"import named ref",0 );
                    //IJ_ERROR ( jimport_named_ref(*new_dataset, refname, os_paths[i], binary),
"import named ref",0 );
            }
            if(binary==SS_BINARY) {
                    IJ_ERROR ( AE_set_dataset_format( *new_dataset, "BINARY_REF" ),"",0 );
            } else {
                    IJ_ERROR ( AE_set_dataset_format( *new_dataset, "TEXT_REF" ),"",0 );
            }

            IJ_ERROR ( AE_find_tool (tool_name, &tool), "attempting to AE_find_tool",0 );
            IJ_ERROR ( AE_set_dataset_tool (*new_dataset, tool)," attempting to AE_set_dataset_tool",0 );
            IJ_ERROR ( AOM_save(*new_dataset),"attempting to AOM_save",0 );
            printf("reached end\n");
            return 0;
}
```