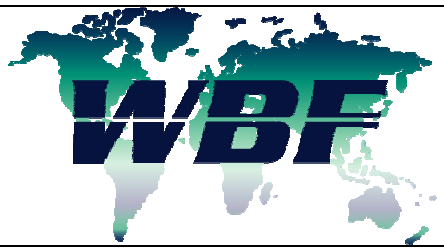
Presented at the
WBF
North American Conference
Atlanta, GA
March 5-8, 2006



195 Wekiva Springs Road, Suite 200
Longwood, FL 32779-2552
+1.407.774.5764
Fax: +1.407.774.6751
E-mail: info@wbf.org
www.wbf.org

Applying ISA-S88 to Software Migration Engineering

Allen D. Benton
Engineering Partner
PRIME Integration
3217 Brampton Street
Dublin, OH 43017
USA
Tel. (614) 761-3904
adbenton@prime-integration.com

KEY WORDS

Migration Engineering, Reverse Engineering, Design Recovery, Functional Design Specifications

ABSTRACT

Dow Chemical maintains thousands of batch control systems world-wide built on its proprietary MOD 5 computer system. Over decades, they have been meticulous to preserve and enhance these systems. Dow intends to preserve and reap the rewards of this diligence by developing reverse engineering best practices and tools to migrate this intellectual property to next-generation platforms. In the summer of 2005, a proof of concept project began with the following goals:

- Improve design recovery to ensure functionality is preserved with a high degree of confidence.
- Improve design documentation to accelerate the migration process from concept to implementation.

The process steps to be explored include:

1. Import software into relational and hierarchical (XML) databases.
2. Analyze and map imported data to an ISA-S88-based framework.
3. Refine data artifacts based on reuse potential and export to Detail Design Specifications (DDS).
4. Develop functional narratives based on software patterns and export to Functional Design Specifications (FDS).

The objectives of this paper are: 1) provide an interim report on the results of these efforts, 2) introduce a common terminology for understanding software migration engineering, and 3) show how these steps can be applied to other batch control systems.

INTRODUCTION

In the early days of process automation, Dow engineers, mathematicians, and scientists programmed small computers to solve complex control problems that they understood intimately. The resulting "super code" is Spartan-like, yet rich with process and safety engineering knowledge. Being pioneers, Dow developed and maintained its own proprietary systems and software. Thirty years later, they are faced with the necessity to mine this software on their own because most commercial PAS vendors, with their object-based application frameworks ("super-sized code"), are wholly focused on the needs of their installed base. The research looking to span this architectural divide with software translation is still basic and unproven [1]. The good news is that research directed at design recovery has demonstrated tangible ways to supplement what has been a labor-intensive and often fruitless activity. The starting point for Dow and others is not to translate old software into new, but to re-interpret it to derive the original intent that we know as requirements. Dow is committed to advancing this new engineering discipline by developing competent committed people, flexible proven processes and light-weight tools to reach their objectives.

System Background

In the early 1970's, Dow developed a fairly rapid succession of MOD 1, MOD 2 and MOD 3 computers under the direction of computer scientist, Wayne Depree. A single all-digital MOD 3 control system was installed and commissioned in Midland, Michigan using a mini computer with just 16 kB of memory. This system incorporated an early version the Dowtran process control language to replace earlier assembly language programming. It was able to execute 2 sequences per I/O subsystem (referred to as a "Can") with a maximum of 5 can's per system. In 1974, the MOD 4 was introduced which contained a 16 bit processor, commercially available computer with 80 kB of memory. It interfaced to a SY FA Process Information system that provided rudimentary history and reporting capabilities. A total of 70 systems were successfully built and installed over the next 4 years. The first MOD 5 computer prototype was completed in 1978. At its core was a military grade computer built by Plessey Semiconductor, originally conceived to program tank turrets to fire while moving. Depree customized the basic CPU architecture to improve processor throughput and hardware reliability. This system became the first fully redundant process control system with parallel CPUs, I/O and power supplies. It also allowed programs to be reloaded without having to shutdown operations. In 1980, the MOD 5 went into full scale production. In the decades since, the DEC line of mini-computers were incorporated into the system architecture, starting with the PDP-11/40, followed by the VAX and currently running on the Compaq Alpha. A typical MOD 5 installation consists of 5 cans. Each can has a 400 I/O capacity, graphical displays, and operator interface units. The current MOD 5 system is certified for SIL 3 applications by the German certifying agency TÜV.

Software Background

An important feature of the MOD 5 control system is that every sequence is executed by the system's computers at a rate of once per second, with options to execute selected portions at rates of ten or one hundred times per second.

A key characteristic of Dowtran programming is that each statement of a sequence is independent (no nesting) and sequential (no branching) and is conditionally executed each time the program cycles. This creates a true deterministic programming environment similar to PLC logic.

The Dowtran language / database uses array elements for all real-time data (I/O and non-I/O). There are about 20 regulatory element types that support instrument I/O, alarms, display input/outputs, redundancy, etc. It also includes support for batch applications in the form of generic recipe element and step event arrays. All arrays are global in scope which complicates translation to the S88 model because of the opportune use of cross-unit coupling.

MIGRATION TERMINOLOGY

To begin to understand migration engineering, it helps to start with a model that most everyone can agree on. The Software Engineering Institute (SEI) has a model they call the "Horseshoe" [2] that attempts to divide the re-engineering process into phases (recovery, transformation and development). The horseshoe name comes from the arc shape that appears on a graph of abstraction level over the lifecycle of a project. One difficulty of this model is that the user must first adopt an architecture-centered approach to software design which is still in research. An important contribution is that it recognizes the pragmatic reuse of software artifacts across the reverse – forward engineering divide.

The following figure presents a new model that builds on this work, but focuses on the deliverables associated with classic software design. The model recognizes the adage that "Reverse Engineering is Forward Engineering in reverse" [3].

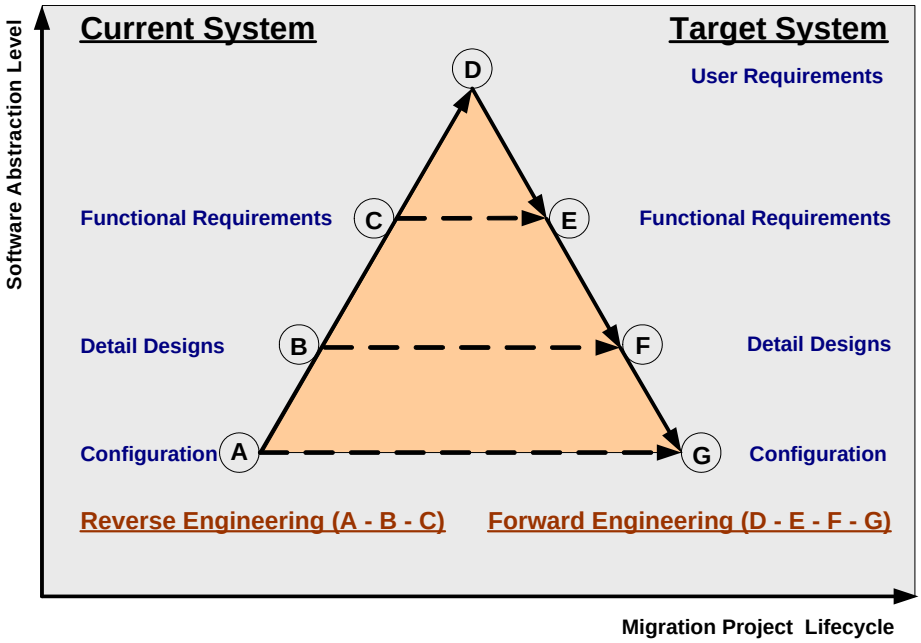


Figure 1 – Migration Engineering Model

Reverse Engineering

The first leg of the process (A → C) is referred to as Reverse Engineering (REV-ENG). Starting with computer-readable configuration files, engineers advance from point A to B using tools to create human-readable Detail Design Specifications (DDS) containing data elements, data flows and control flows. These documents are patterned after the existing modularity and data structure of the existing system (I/O, graphics, sequences, recipes, etc.). During this step, meaningful descriptions are applied to the data, including engineering units of measure. Once these documents have been reviewed and approved, the next phase (point B to C) starts with creation of an ISA-S88 representation of the process domain that the software currently operates in. Many assume this is not possible if the system was implemented without the aid of these models. In reality, the absence of modularity allows the existing configuration to be manipulated quite easily. The phase is completed with the creation of the Functional Design Specification (FDS) which is also reviewed and approved. This document is made up of brief narratives describing the role and logic of each useful software artifact at the Unit, Equipment Module and Control Module levels of the Physical model, as well as the Unit Procedure, Operation and Phase levels of the Procedural Control model.

Forward Engineering

The second leg of the process (D → G) is referred to as Forward Engineering (FWD-ENG) to distinguish it from traditional software engineering. This is a useful distinction since classic software engineering tends to be iterative and labor-intensive, while FWD-ENG emphasizes document automation and software reuse. The converging arrows (D → E, C → E), (E → F, B → F), and (F → G, A → G): show where reuse typically occurs. Data from REV-ENG documents is combined with abstractions from the FWD-ENG deliverable to create a useful starting point for the next deliverable. Tools and a data repository can be developed to automate each phase of this process. Forward Engineering starts with the user spelling out in general terms what existing functions are to be removed and what functions are to be added or changed. This document is referred to as the User Requirements specification (URS) and is also reviewed and approved. The remaining documents and the processes used to create them are mostly self-explanatory.

Why bother with multiple steps? In a word, the answer is “semantics”. Each step of the process introduces generalization or abstraction into the system description. Abstraction is what allows the essence of the current system to be captured and carried over to the target system. The process is optimized when software artifacts are abstracted only to a level where the semantics that describe them apply also to the target system. Semantic distance describes the degree to which an abstraction is different from its implementation.

Static Analysis Techniques

With full access to a running system, design recovery could be accomplished by observing all behaviors under all possible conditions (Dynamic Analysis). The reality is that most systems are continuously on-line and many scenarios have safety implications. The techniques known as Static Analysis rely on source code and configuration data to answer the questions: “What is the system capable of doing?” and “How does it accomplish it?” In the same way Google was able to capitalize on the insight that web pages contain valuable structural and linkage information, static analysis has demonstrated that source

code is equally rich in this type of content and can be leveraged to make reverse engineering projects become cost-effective. The Proof-of-Concept work started with the gathering of industry-proven approaches to understanding of a software system from a functional perspective [4]. The existing MOD 5 software migration toolkit supports many of these analyses. The following techniques represent current state of the art:

Configuration Disintegration – This technique starts with a language-specific parser to break source code into tokens which are saved in a relational database as data elements, data flows and control flows. Certain data elements of interest (ex. Analog Outputs, Digital Outputs, Step transitions) are then re-combined into "Clusters" [5] using Dependency graph techniques which trace flow relationships between program statements. This step effectively removes language syntax variations so general purpose tools and database structures can be shared for further analyses.

Pattern Matching – This technique compares clusters using distance algorithms to look for like repetitions of data flows and control flows [6]. A comparison algorithm that exploits knowledge of the software grammar is able to point out differences most accurately. Patterns are useful for recovering design solutions within a process domain which need only to be described, re-implemented, and tested once at the functional level.

Program Slicing – This technique allows the user to view clusters in the context of the original source code [7]. An XML representation of the source code is generated and XSLT is used to create an HTML-based source code listing. Hyperlinks are inserted in the source code to allow the engineer to traverse the tree structure of the cluster.

Dependency Evaluation – This technique takes a macro view of the system by showing the interdependence of data elements with different sub-systems such as operator displays, recipes, I/O, reports, etc. Traditional implementations of this technique include cross reference reporting.

Metrics Evaluation – Once clusters have been assigned to control modules (Physical Model) and phases (Procedural Control Model), various characteristics such as size, complexity, quality, maintainability, etc. (ex. Halstead Complexity factor and Design Structure Quality Index) can be measured [8] and assigned at the cluster and module level. These techniques can be used to access re-implementation and testing risks or to highlight overly complex control strategies.

View Exploration – No single view of software is best when attempting to analyze functionality. This technique makes at least four views available. These include: 1) Physical Hierarchy view, 2) Procedural Hierarchy view, 3) Statement Type Cluster view, and 4) Source Code view. In addition, each view should provide easy navigation within the view and to other views. Each of these views also needs to be available in paper report formats that can be printed and marked up during review meetings.

Flow Visualization – This technique differs from Exploration in that it uses a graphical representation of data flows (Data Flow diagrams) and control flows (State Transition diagrams) to represent portions of the system. A useful chart type for batch applications is the Step chart which shows each step and all possible transitions to and from the step.

MIGRATION PROCESS

The figure on the next page shows the stages of analysis, the techniques applied and the deliverable products. Four sequences, selected from different processes, were used to exercise the proof of concept.

The following steps have been found to provide the most efficient path through the REV-ENG portion of the migration process:

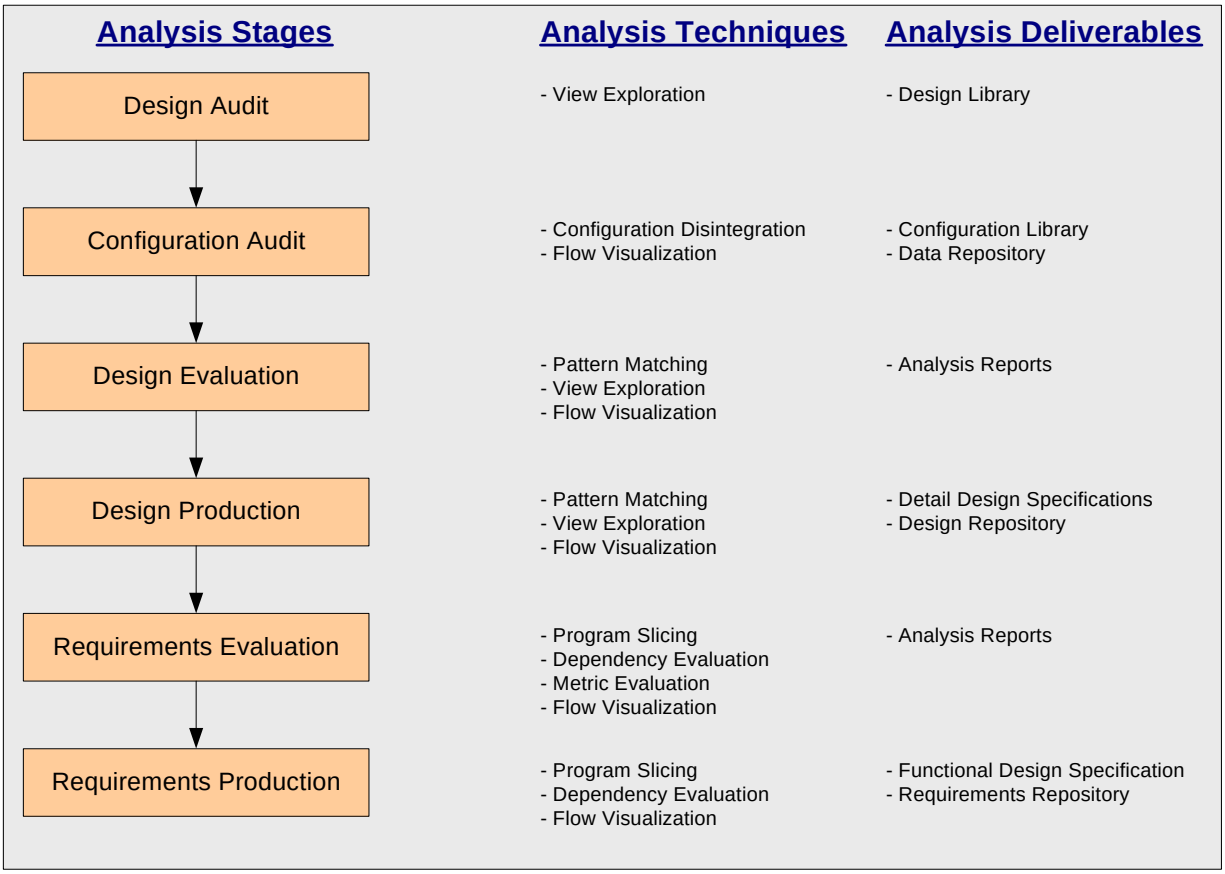


Figure 2 – Reverse Engineering Analysis Stages

Stage 1 - Design Audit

Step1 - Take inventory of existing design documents

Existing design documentation may be terribly out of date but it often offers insights into the developer's intents. The revision history of these documents (if available) often suggests opportunities for design improvement. It can also reveal the implicit modularity of the process domain that monolithic software was written to control. Only certain documents (P &IDs, Instrumentation Lists) need to be updated and verified to become useful reference documentation for the physical instrumentation hardware.

Stage 2 - Configuration Audit

Step 2- Take inventory of existing software configuration

All application software and data needs to be identified and cataloged with names that describe the

functionality within. These files should be in ASCII format and accessible by the computer where the analysis work is being done. If on-going software maintenance is taking place on the original system during the migration project, detailed change control procedures should be in place to insure changes appear on the target system as well.

Step 3- Import software into a data repository

Depending on the structure of the source files, various techniques can be used to parse code into tokens and write it to various tables in a relational database. Ideally, this database would be assessable from a central server that supports concurrent users. The only requirement for the source files is that the grammar be context-free and that the source code must compile without errors. With Dow's older installations, source code comments are the richest source of design documentation so one of the requirements for tool development was to preserve comments and make them accessible in the correct context.

Stage 3 - Design Evaluation Stage

Step 4 - Modify (complete/correct) data definitions

The importance of this step is tempting to overlook since the existing system works. The goal is to clean up / standardize all data element descriptions, instrument ranges and instrument units of measure to create a "data dictionary" that can be verified and referenced later. Equations, scaling and constants need to be understood and checked for invalid/outdated assumptions. Alarm types and limits should be re-assessed for present day relevance. If this stage is bypassed, the result will be unnecessary test, perpetuated quality problems and nuisance alarms.

Stage 4 - Design Production Stage

Step 5 - Categorize data by reuse potential and ISA-S88 hierarchy

Ideally, assessing reuse potential is a task reserved for the Forward Engineering and creation of the User Requirements Specification (URS). In practice, decisions need to be made at the design level as to which software artifacts to further analyze at the functional level. The decision to assign or not-assign clusters and parameter data to the ISA-S88 hierarchy effectively accomplishes this task since only linked elements are visible at the functional level. The result is an extension to the ISA-S88 hierarchy as shown in figure 3. This drawing is not intended to suggest an extension to S88 specification, simply provide a frame of reference.

Step 6 - Export to Detail Design Specification (DDS)

Dow recognized that getting buy-in from future stakeholders for the new system needs to begin early. One way to accomplish this is to bring them into the review and approval process of the design documents. In this context, a wealth of information about how the existing system operates in the real world will begin to surface. This feedback must be captured in an issues database and analyzed in order to establish credibility with the team. One obstacle that was addressed was the need to present detailed documents in the format Dow engineers had grown accustomed to through their existing work processes. A Microsoft® VBA conversion utility was developed to translate the database-optimized detail design document to a reviewer-optimized version.

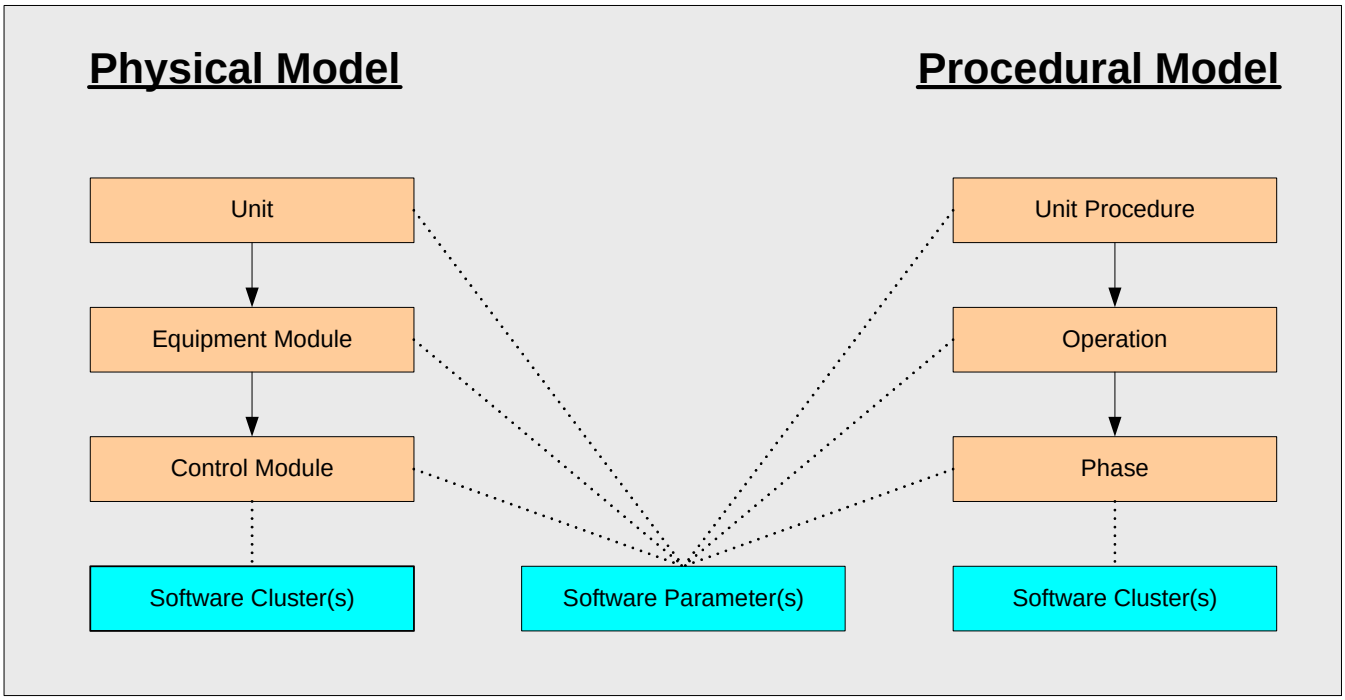


Figure 3 – Control Module / Phase Linkages

Stage 5 - Requirements Evaluation

Step 7 - Develop upper-level narratives (Units, Equip. Modules, Unit Procedures and Operations)
This is a Top-down approach to requirements development that defines the scaffolding (context) on which the detailed system requirements will be attached. The task is especially important if the existing system was created without an ISA-S88 perspective. Numerous educational resources are available from WBF and ISA to guide this effort.

Stage 6 - Requirements Production

Step 8 - Develop functional narratives for Control Modules and Phases
This is a Bottom-up approach to requirements development. The task is to synthesize functional narratives from cluster-level source code. A useful narrative melds a process description (purpose) with a software description (automation scope) that communicates the original intent to multiple reviewers. For complex requirements with nested decision branches, a pseudo code vocabulary needs to be established and enforced until it becomes natural.

Step 9 - Export to a Functional Design Specification (FDS)
This is the second opportunity to promote stakeholder buy-in. If requirement narratives have been captured in a relational database, tools can be used to auto-generate the document in a MS Word

format using a customer-defined template. As with the previous review, new information about the existing system will surface and this feedback must also be captured and analyzed.

MIGRATION TOOLS

The figure below shows the organization of views used to implement a proof of concept Software Migration Environment (SME).

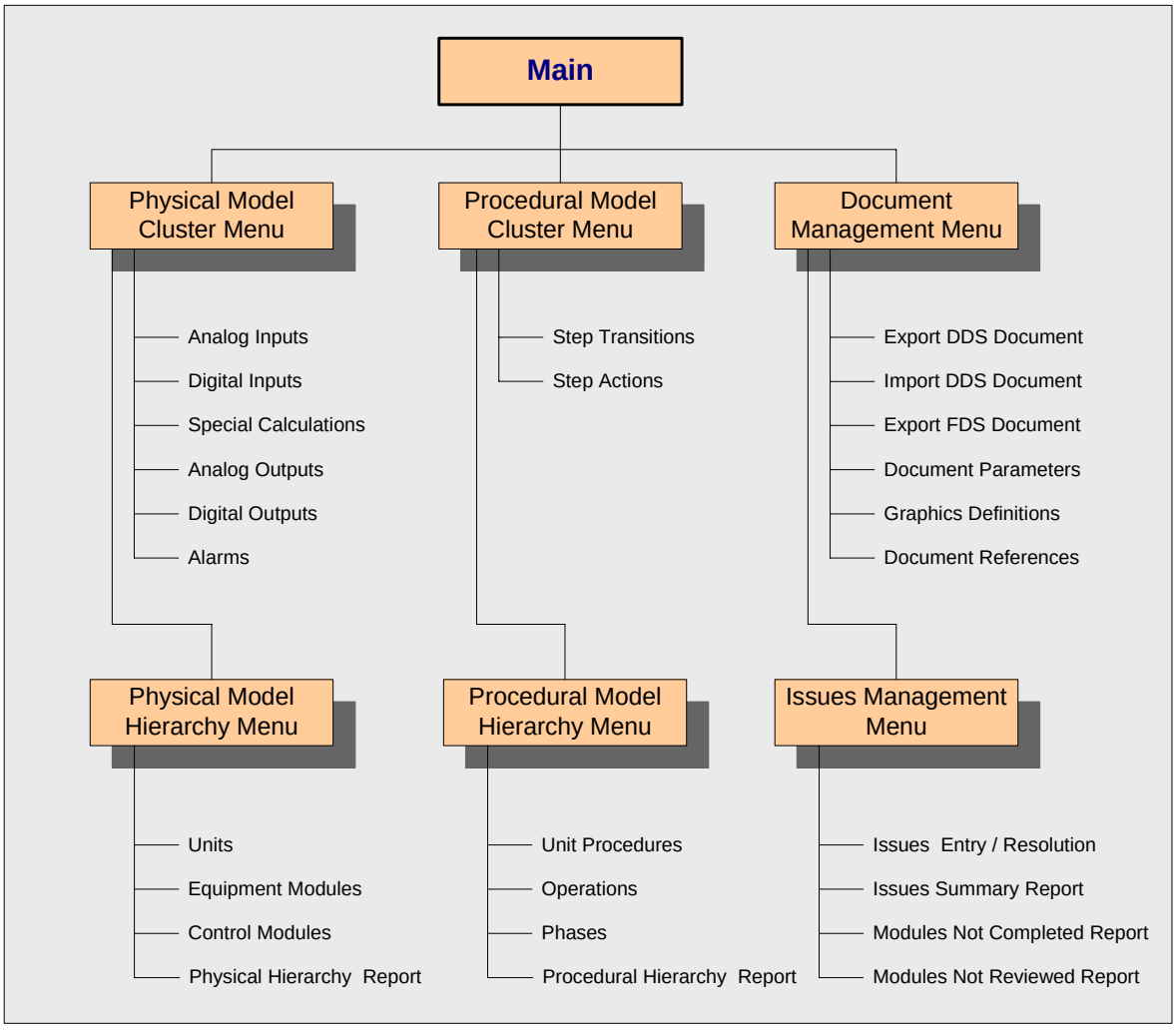


Figure 4 – Software Migration Environment

Microsoft® Access was used as the database and application environment for speed of deployment. A single Dowtran sequence may vary in size from 1 to 6 KLOC (excluding comments) which can result in a database in the order of 20 Mb. These file sizes create a practical limit of one unit design per database for the most complex applications. Potential workarounds include using a more efficient data schema or upgrading the database engine to MSDE. Also, database partitioning is not considered a handicap because the normal division of engineering labor occurs along unit boundaries. Database linking could later be used in the future to generate cross-unit reports for project-wide analysis.

SUMMARY

Dow has taken a robust first step towards mining its MOD 5 applications using these proof-of-concept reverse engineering tools and processes. They have demonstrated that design recovery can be systematically achieved using industry-proven software migration techniques. By applying the ISA-S88 hierarchy they have imposed modularity and demonstrated conformity to common object-oriented batch frameworks with the intent to accelerate software design and development. An equally important next step will be to approach commercial PAS vendors with electronic design documents and design data repositories to determine what forward engineering tools will be made available to automate significant portions of the configuration process.

REFERENCES

1. Maarit Harsu*; Identifying Object-Oriented Features from Procedural Software,* Department of Computer and Information Sciences, FIN-33014 University of Tampere, Finland. (2000)

2. *Reengineering: The Horseshoe Model*, Carnegie Mellon University, (2005). http://www.sei.cmu.edu/reengineering/horseshoe_model.html

3. Ira D. Baxter and Michael Mehlich; *Reverse Engineering is Reverse Forward Engineering,* Proceedings of Fourth Working Conference on Reverse Engineering (1997).

4. Dean Jin and James Cordy; *A Service Sharing Approach to Integrating Program Comprehension Tools* Queen's University, Kingston, Canada (2004).

5. Anquetil N., Lethbridge T. C; *Experiments with Hierarchical Clustering Algorithms as Software Remodularization Methods,* Proceedings of Sixth Working Conference on Reverse Engineering (October 1999).

6. Ira D. Baxter, et al, *Clone Detection Using Abstract Syntax Trees,* Proceedings of the International Conference on Software Maintenance (1998).

7. GrammaTech, Inc., *CodeSurfer Technology Overview: Dependence Graphs and Program Slicing* (2000) GrammaTech, Inc

8. Rudolph E. Seviora, *Lecture Notes Related to Metrics (Source code),* University of Waterloo http://www.swen.uwaterloo.ca/~kostas/ECE750-3/metrics-1-notes.pdf

