Enhancing the Software Improvement Processes through Object-Process Methodology

By

Christine Miyachi

BS, Electrical Engineering
University of Rochester, 1984
SM, Technology and Policy/Electrical Engineering,
MIT, 1986
MS, Electrical Engineering
UMASS Lowell, 1999

Submitted to the Sloan School of Management
in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Engineering and Management

At the

Massachusetts Institute of Technology

January 2001

Signature of Author____

MIT Sloan School of Management
January 2001

Certified by Dov Dori ____

Prof. Dov Dori
MIT School of Engineering
Thesis Supervisor

Accepted by____

Stephen C.Graves
LFM/SDM Co-Director
Abraham Siegel Professor of Management

Accepted by____

Paul A. Lagace
LFM/SDM Co-Director
Professor of Aeronautics & Astronautics and Engineering Systems

This thesis is dedicated to my husband Hiroshi and three children, Mari, Ken, and Tomi, whose love and support made it possible for me to complete.

This thesis is also dedicated to Dr. Dov Dori whose guidance and brilliance in creating OPM was an inspiration for this work.

Enhancing the Software Improvement Processes through Object-Process Methodology

By

Christine Miyachi

Submitted to the Sloan School of Management
in Partial Fulfillment of the Requirements for the Degree of Master of Science in Management

Abstract

Object Process Methodology is a systems engineering methodology that provides a way to express and communicate architecture of complex systems among stakeholders. Unlike other methodologies, it provides both a visual part through Object Process Diagrams (OPDs) and a natural language specification in the Object Process Language (OPL). The OPD and OPL have exact correspondence to each other.

The objective of the research was to model and examine two software development approaches using OPM. The research first developed an OPM tool over Microsoft Visio®. Then, it applied that tool to two software process improvement methodologies: the Capability Maturity Model (CMM) and the Unified Software Development Process (USDP), which is based upon the Unified Modeling Language (UML).

Using OPM as the analysis tool, improvements and misconceptions about these two methodologies were shown. In CMM, it was clear that many people were needed to carry out the process and that this excluded smaller companies with fewer resources from implementing it. In USDP, some goals of the process architecture were not met as shown in the analysis. Automating both methodologies, with an OPM tool provides would speed up the process and makes it more reliable.

Thesis Supervisor: ***Dov Dori***
Visiting Associate Professor, Engineering Systems Division

Enhancing the Software Improvement Processes through Object-Process Methodology

By

Christine Miyachi

Submitted to the System Design and Management Program
January 2001 in Partial Fulfillment of the
Requirements for the Degree of Master of Science in
Engineering and Management

Executive Summary

Problem Statement

Many enterprises have adopted the Capability Maturity Model (CMM) and the Unified Software Development Process (USDP) to increase their level of software development productivity. This commitment to these processes is expensive as it is managed largely without software support. This thesis employs Object Process Methodology (OPM) to model and monitor the software improvement process in order to make its adoption easier through automation and decrease the expenses associated with its implementation. OPM is used to model CMM and USDP. Methods for using OPM to automate these processes have been created.

B. Originality Requirement

The application of OPM to model and monitor the software improvement process is original. OPM is a systems engineering design paradigm for modeling systems in general, not just software systems. Therefore, we were able to model systems that involve organizations, hardware, and software, which are at the heart of the software improvement process. To be able to generate Object-Process Diagrams (OPDs) and translate them to Object-Process Language (OPL) and vice versa, we developed a software tool called OP-tool.

C. Content and Conclusions

We examined and modeled the architecture of both software improvement processes and found that the CMM process architecture does indeed meet its goals, but there is one coupling in the architecture that the designers do not seem to be aware of. USDP goals are stated with implementation embedded in them. We found that some goals were not capable of being met through this process. Our OPM model shows that CMM requires a high level of human involvement, making it laborious and expensive. The OP-tool has demonstrated that having both a diagram and textual parts of the analysis is indispensable. The OPL served as the specification and the OPD served as the visual stimulus for the analysis. Many times, we got the diagrams wrong only to look at the OPL and make it right.

D. System Design and Management Principles

Software development processes are a necessary component of managing and practicing software engineering in today's competitive world where quality and faster time-to-market is key. However, software development processes are not without cost. This thesis analyzed the architecture of two software development processes and determined whether they meet their goals. Complexity and number of people needed to implement the process were analyzed. These two factors are directly related to cost. We used OPM to do the modeling and analysis. Moreover, we envision that using OPM can yield an improved, less costly software development process through automation.

Our research also indicates that current software development processes are too expensive for small and medium size companies. We conclude the work by recommending several versions of the software development process, which depend on the size of the organization.

E. Engineering and Management System Content

This thesis is founded on the engineering systems principle of top-down system analysis and specification. This principle is applied in two domains. The first is analyzing the architecture of two widely practiced software development processes – Carnegie Mellon - Software Engineering Institute's Capability Maturity Model (CMM) and the Unified Software Development Process (USDP), which has been proposed as a process to be used in conjunction with the Object Management Group's standard Unified Modeling Language (UML). The second domain is

engineering a prototype of an OPM-based software system to support the implementation of the specification of these two software development processes. This included eliciting and analyzing requirements, designing, and finally implementing a preliminary, limited functionality, proof-of-concept prototype of an OPM tool to model and automate the various software engineering processes.

To analyze and design both the architecture of the processes and the prototype, we used the architectural framework developed in ESD.34 “Systems Architecture”. Any architecture has *goals* that have to be met, and *structure* and *behavior* (also referred to as “form” and “function”) to meet those goals. For each of the process architecture we determined the goals, as well as the combination of structure and behavior to meet these goals. All the modeling activities were done with OPM. To implement the design of the proof-of-concept prototype, an algorithm and code were developed for moving back and forth between Object Process Diagrams (OPD) and Object Process Language (OPL) using the PERL scripting language. Finally, a proof-of-concept prototype of a tool that generates one type of OPL structure sentence from OPD was developed using Visual Basic for Applications (VBA) and Visio®, a standard diagramming tool.

The work contained in the thesis is the author’s and original.

Table of Contents

List of Figures

C h a p t e r 1

INTRODUCTION

Chapter 1: Introduction

Many enterprises have adopted the Capability Maturity Model (CMM) and the Unified Software Development Process (USDP) to increase their level of software development productivity. Commitment to these processes is expensive as they are managed largely without software support. This thesis employs Object Process Methodology (OPM) to model and analyze the software improvement process in order to make its adoption easier through automation and decrease the expenses associated with its implementation. OPM is used to analyze the software process improvement methodology and then it will be shown how OPM can enhance and improve the methodologies. The methodologies chosen to analyze are the Capability Maturity Model (CMM) and the Unified Software Development Process (USDP), a process that uses the Unified Modeling Language as one of its core tools. These two processes are radically different and OPM shows their differences and ways for improvement. The OPM tool developed can be used to improve the software processes. This is demonstrated through a prototype developed with Visio®.

All code developed is shown in the Appendices.

The goal of OPM is to allow for communications among all parties in product development. It is one of the only systems engineering methodologies for engineering systems, not just software or hardware, and thus can be used to analyze, design, and communicate complex systems among all involved stakeholders.

C h a p t e r   2

USING OPM TO ANALYZE CMM AND UNIFIED SOFTWARE DEVELOPMENT PROCESS

Chapter 2: Using OPM to analyze CMM and Unified Software Development Process

Introduction

This chapter will now focus on the application of OPM to an architecture problem. This chapter will analyze the architecture of software development process.

Unlike other methodologies that just traditionally work with software or mechanical systems, OPM can be used to analyze many aspects of systems architecture and engineering. Using OPM as the tool for analyzing processes, this chapter examines the architecture of two software processes and compares their strengths and weaknesses. The two software processes we focus on are the Capability Maturity Model (CMM) process and the Unified Software Development Process will be compared.

Architectural Framework

To evaluate the architecture of these two processes, we need a framework for defining architecture. The framework [3] used is described in the Object-Process Diagram (OPD) in below.

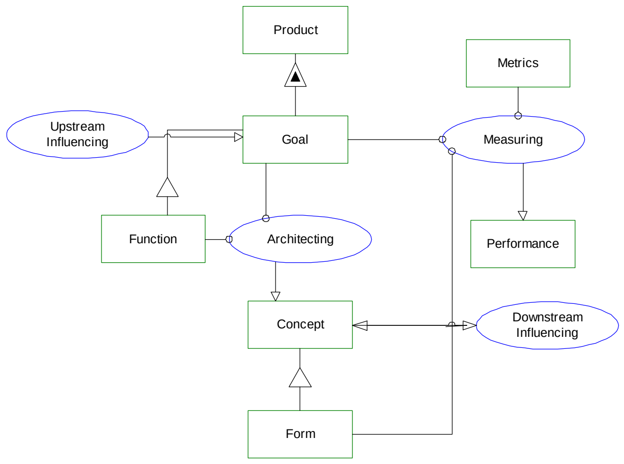


Figure 1 A framework for defining architecture

Upstream Influencing yields **Goal**.
Product exhibits **Goal**.
Function is a **Goal**.
Measuring requires **Goal, Metrics** and **Form**.
Measuring yields **Performance**.
Architecting requires **Function** and **Goal**.
Architecting yields **Concept**.
Form is a **Concept**.

The upstream influences considered in our analysis are corporate strategy, market data and competition, market strategy, technology, and operations strategy. These influences are translated into needs and then to goals. Goals lead to more refined functions of the architecture. The goals and functions are required in the architecting process, which yields a concept and then a more refined form. The form, goals, and a set of metrics are used to evaluate the architecture and produce a performance measure.

We use this framework, described in OPM, to analyze and evaluate the architecture of the two software processes. We define good architecture by the extent to which the architecture meets its goals. The goals, function and form of the architectures of the two processes will be defined and evaluated by measuring the goals with a set of metrics.

Capability Maturity Model

Watts Humphrey and the Software Engineering Institute (SEI) at Carnegie-Mellon University invented capability Maturity Model (CMM). It was originally used for military avionics applications to evaluate software vendors, but has now spread and is being used by major organizations in virtually every segment of the economy in the US and globally [4].

Goals of the CMM Architecture

Most products contain software today, whether ubiquitous, like in a washing machine or Internet applications, or mission-critical, like a space shuttle. However, software is perceived to be the weak link in developing high quality products. CMM was created not as a silver bullet to solve this problem, but as common-sense engineering process for software. The goal of the CMM architecture is to apply Total Quality Management (TQM) principles to software. The highest-level goal of TQM is to meet the needs and expectations of the customer, now and in the future. Any software improvement effort should operate inside the larger context of a business [9]. Although the CMM does not state explicitly that the customer should be satisfied, it does state that the software supplier should work with the customer to understand the customer's requirements and should build software products that satisfy the customer's needs as documented in the requirements allocated to the software component of the total system or product being supplied.

Meeting the needs of the customer can be decomposed into three goals: be on time, be on budget, and meet the requirements. For software, being on time and on budget means preventing the amount a rework that comes about by misunderstanding the requirements.

To measure how those goals are met, we will study the function and form of the process using OPM. We will then evaluate studies that measure the performance of the architecture.

Critics of the CMM state that it encourages too much bureaucracy. Back [1] argues in favor of "software heroism." Back claims that heroic software people find a way to solve problems and software projects are just a sequence of problems encountered and solved. The CMM authors specifically state that software heroes are too hard to find and burn out quickly.

A summary of CMM goals is listed in the table below.

Goal	*Metric*
Process should allow software to meet needs of customers	Measure % of customer needs met
Allow for easy adoption of the process	Measure time and cost of adoption
To reduce the amount of rework in a software project	Measure amount of rework

Table 1- Goals and Metrics for CMM Architecture

Overview of the CMM

The figure below is the OPD and OPL of the high-level architecture of CMM.

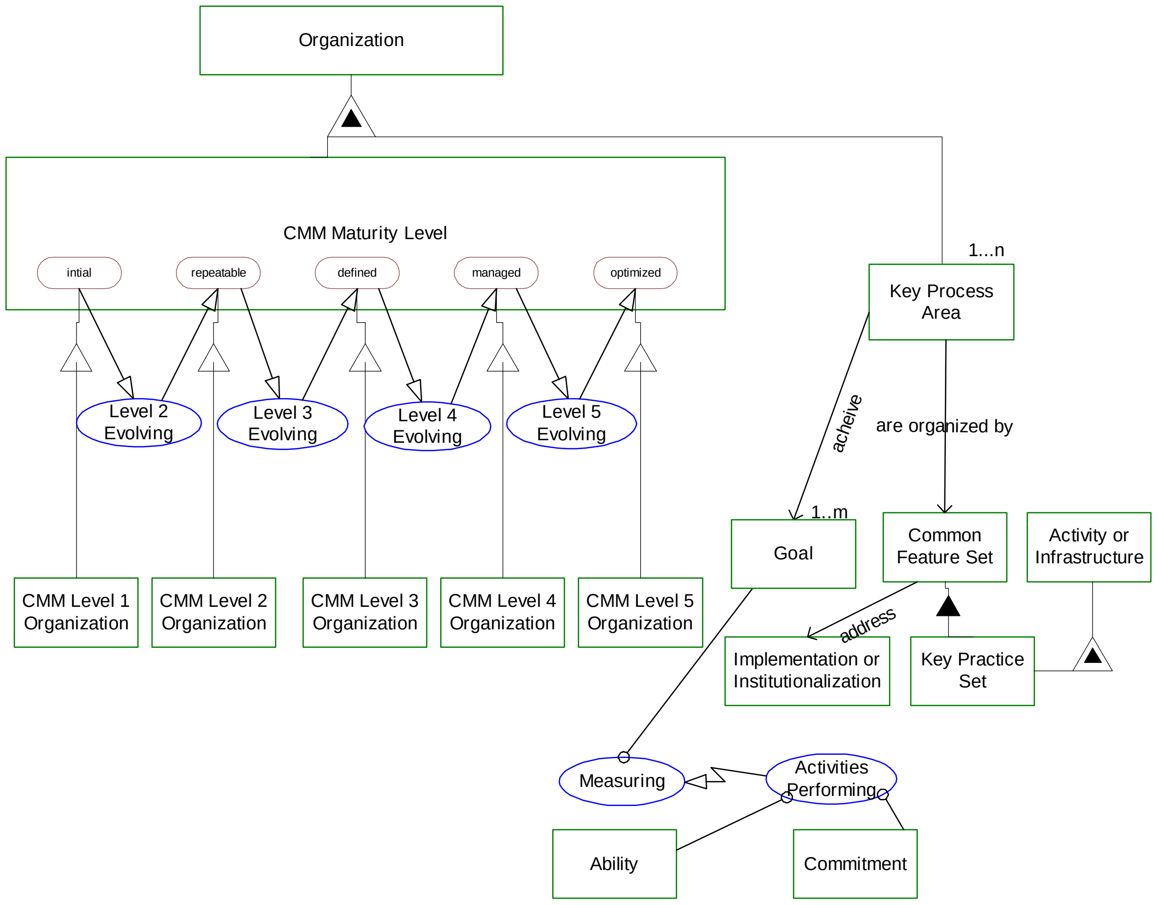


Figure 2: High-Level architecture of the CMM

Organization exhibits **CMM Maturity Level.**
CMM Level 1 Organization is an **Organization,** the **CMM Maturity Level** of which is **initial.**
CMM Level 2 Organization is an **Organization,** the **CMM Maturity Level** of which is **repeatable.**
CMM Level 3 Organization is an **Organization,** the **CMM Maturity Level** of which is **defined.**
CMM Level 4 Organization is an **Organization,** the **CMM Maturity Level** of which is **managed.**
CMM Level 5 Organization is an **Organization**, the **CMM Maturity Level** of which is **optimized**.

Key Process Areas are organized by Common Features and **achieve Goals**.
Common Features consist of **Key Practices**.
Key Practices features **Activities or Infrastructure**.
Common Features address **Implementation** or **Institutionalization**.
Level 2 Evolving changes **CMM Maturity Levels** from **initial** to **repeatable**.
Level 3 Evolving changes **CMM Maturity Levels** from **repeatable** to **defined**.
Level 4 Evolving changes **CMM Maturity Levels** from **defined** to **managed**.
Level 5 Evolving changes **CMM Maturity Levels** from **managed** to **optimized**.

The specification of the CMM is generated via OPD and it OPL script equivalent. To allow companies to meet the demands of customers, the CMM designers created a process that is phased and allows companies to evolve. They did this in response to the observation (which is a customer need) that organizations cannot change overnight. Table 5 describes the levels of organization evolution according to CMM.

CMM Level	*Focus*	*Key Process Area*
1 Initial	Competent people and heroics	None
2 Repeatable	Project management processes	Requirements management Software project planning Software project tracking and oversight Software subcontract management Software quality assurances Software configuration management
3 Defined	Engineering processes and organization support	Organization process focus Organization process definition Training program Integrated software management Intergroup coordination Peer reviews
4 Managed	Product and process quality	Quantitative process management Software quality management
5 Optimizing	Continuous process improvement	Defect prevention Technology change management Process change management

Table 2: CMM Maturity Levels, and Focus

Each CMM level has a number of major characteristics. At the initial level, Level 1, the software process is characterized as ad hoc, and occasionally even chaotic. Few processes are defined, and success depends on individual effort and heroics. At Level 2, basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes and projects with similar applications. At Level 3, the software process for both management and engineering activities is documented, standardized, and integrated into a standard software process for the organization. Projects use an approved tailored version of the organization's standard software process(es) for developing and maintaining software. At Level 4, detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled. At Level 5, continuous process improvement is facilitated by quantitative feedback from the process and from piloting innovative ideas and technologies.

OPM provides a framework to distinguish between processes and objects, which is sometimes muddled in architecture design. Consequently, the language used in the OPL is slightly different than the language used above, which is the language in the CMM. The language of the OPL originated in the CMM, but was adapted to follow OPM rules.

Next Level of Decomposition

The next level of decomposition is shown in the figure below. Each organization, which achieves CMM certification at a specific level, will have the shown processes and objects in place. In the figure below, Level 2 is shown.

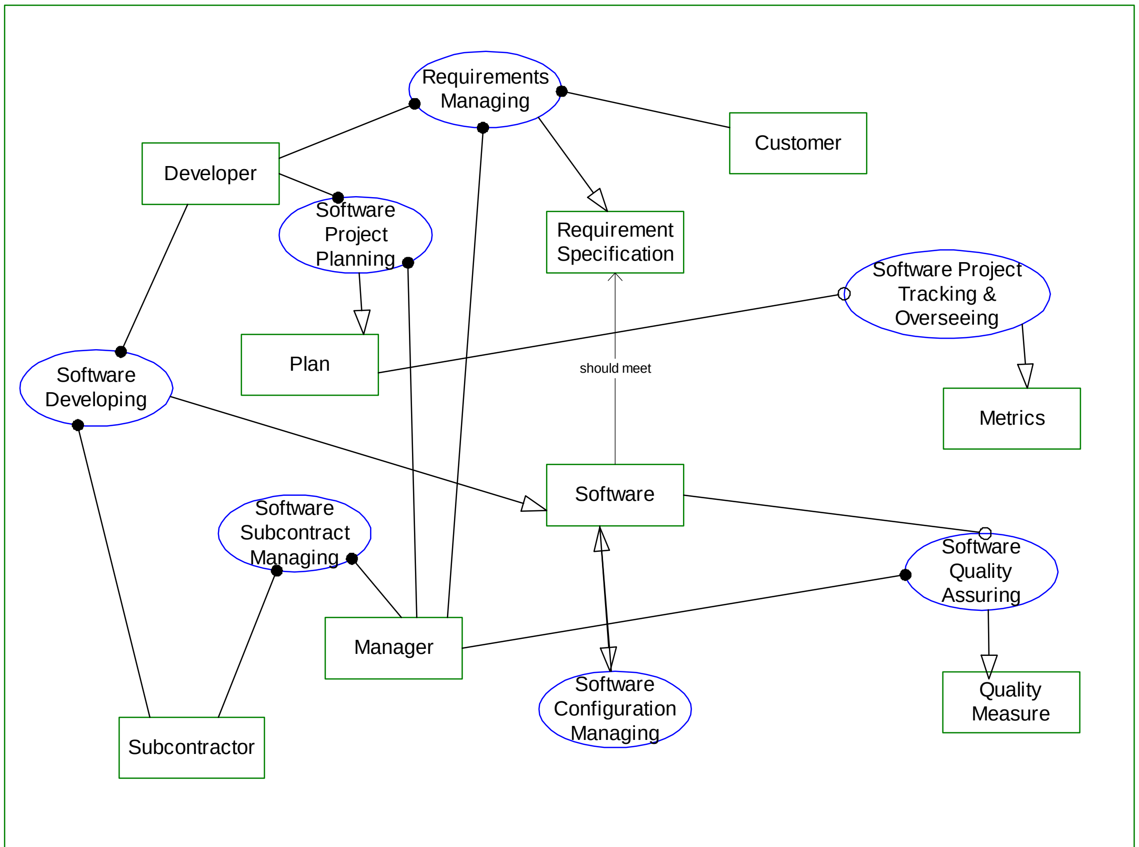

Requirements Managing
Customer
Developer
Software Project Planning
Requirement Specification
Software Project Tracking & Overseeing
Plan
should meet
Software Developing
Metrics
Software
Software Subcontract Managing
Software Quality Assuring
Manager
Software Configuration Managing
Quality Measure
Subcontractor

Figure 3-3: CMM Level 2 Organization

Figure 3: CMM Level 2 Organization

Software Developing yields **Software**.
Software Project Planning yields **Plan**.
Software Project Tracking & Overseeing yields **Metrics**.
Software Quality Assuring yields **Quality Measure**.
Software Configuration Managing affects **Software**.

Software should meet Requirement Specification.
Requirements Managing yields **Requirement Specification**.
Developer and **Subcontractor** handle **Software Developing**.
Developer and **Manager** handle **Software Project Planning**.
Developer, Manager, and **Customer** handle **Requirements Managing**.
Manager handles **Software Quality Assuring**.
Software Project Tracking & Overseeing requires **Plan**.
Manager and **Subcontractor** handle **Software Subcontract Managing**.

As stated earlier, the key process areas at Level 2, which are represented as the five processes in the OPD, focus on the software project's concerns related to establishing basic project management controls. Interestingly, the specification does not list explicitly the objects that the processes affect. Due to the fact that OPM requires that objects be involved in the occurrence of processes, we were able to derive these objects, and they are displayed via the OPD in the figure below. From the CMM specification [9], the purpose of each key process area is described.

Requirements Management: Establish a common understanding between the customer and the software project of the customer's requirements to be addressed by the software project. This agreement with the customer is the basis for planning and managing the software project.

Software Project Planning (SPP): Establish reasonable plans for performing the software engineering and for managing the software project.

Software Project Tracking and Oversight (SPTO): Establish adequate visibility into actual progress so that management can take effective actions when the software project's performance deviates significantly from the software plans.

Software Subcontract Management: Select qualified software subcontractors and manage them effectively.

Software Quality Assurance: to provide management with appropriate visibility into the process being used by the software project and of the products being built.

Software Configuration Management: Establish and maintain the integrity of the products of the software project throughout the project's software life cycle.

Level 2 is often the most difficult level to obtain. Almost 100% of the companies that do not attain Level 2 certification fail to get it because they cannot implement SPTO and SPP [5]. According to one study of 28 organizations, it took an average of 26.5 months to evolve from Level 1 to Level 2 [5].

The Level 3 organization is described in the figure below and the following OPL.

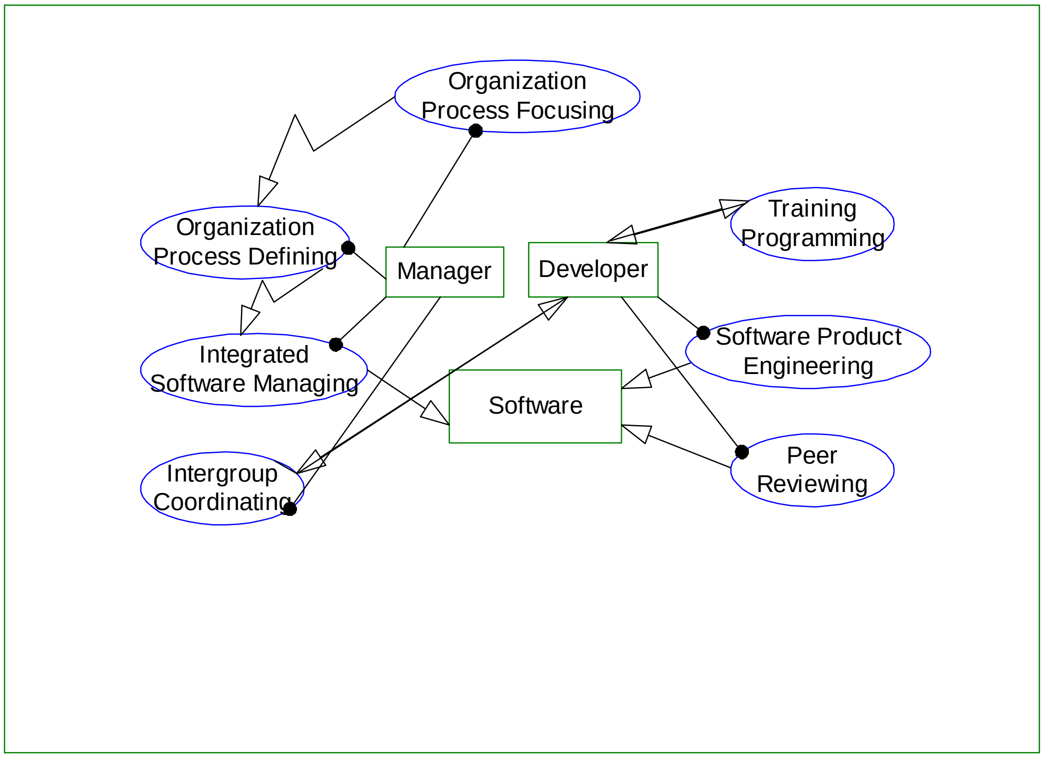

Organization Process Focusing
Organization Process Defining
Integrated Software Managing
Intergroup Coordinating
Manager
Developer
Training Programming
Software Product Engineering
Software
Peer Reviewing

Figure 4: CMM Level 3 Organization

Manager handles **Organization Process Focusing, Organization Process Defining**, **Integrated Software Managing,** and **Intergroup Coordinating**.
Intergroup Coordinating affects **Developer**.
Training Programming affects **Developer**.
Integrated Software Managing, Software Product Engineering, and **Peer Reviewing** yield **Software**.
Developer handles **Software Product Engineering** and **Peer Reviewing**.
Organization Process Focusing yields **Organization Process Defining**.
Organization Process Defining yields **Integrated Software Managing**.

The key process areas for Level 3 are focused on project and organizational areas. Its processes concentrate on creating an organization that can establish the infrastructure for institutionalizing effective software engineering and management processes across all projects. As done above, the purpose of each key process area is described below.

Organization Process Focus: Establish the organizational responsibility for software process activities that improves the organization's overall software process capability.

Organization Process Definition: Develop and maintain a usable set of software process assets that improves process performance across the projects and provides a basis for defining meaningful data for quantitative process management. These assets provide a stable foundation that can be institutionalized via mechanisms such as training.

Training Program: Develop the skills and knowledge of individuals so they can perform their roles effectively and efficiently.

Integrated Software Management: Integrate the software engineering and management activities into a coherent, defined software process that is tailored from the organization's standard software process and related process assets. This tailoring is based on the business environment and technical needs of the project.

Software Product Engineering: Perform consistently a well-defined engineering process that integrates all the software engineering activities to produce correct, consistent software products effectively and efficiently.

Intergroup Coordination: Establish a means for the software engineering group to participate actively with the other engineering groups so the project is better able to satisfy customer needs effectively and efficiently.

Peer Reviews: Remove defects from the software work products early and efficiently.

According the same study [5], it took 23 organizations an average of 24 months to evolve from Level 2 to Level 3.

Level 4 concentrates on establishing a quantitative understanding of both the software process and the software products being built. The processes in these areas largely parallel Six Sigma because they include statistical approaches to software development [2]. The figure below shows a Level 4 organization.

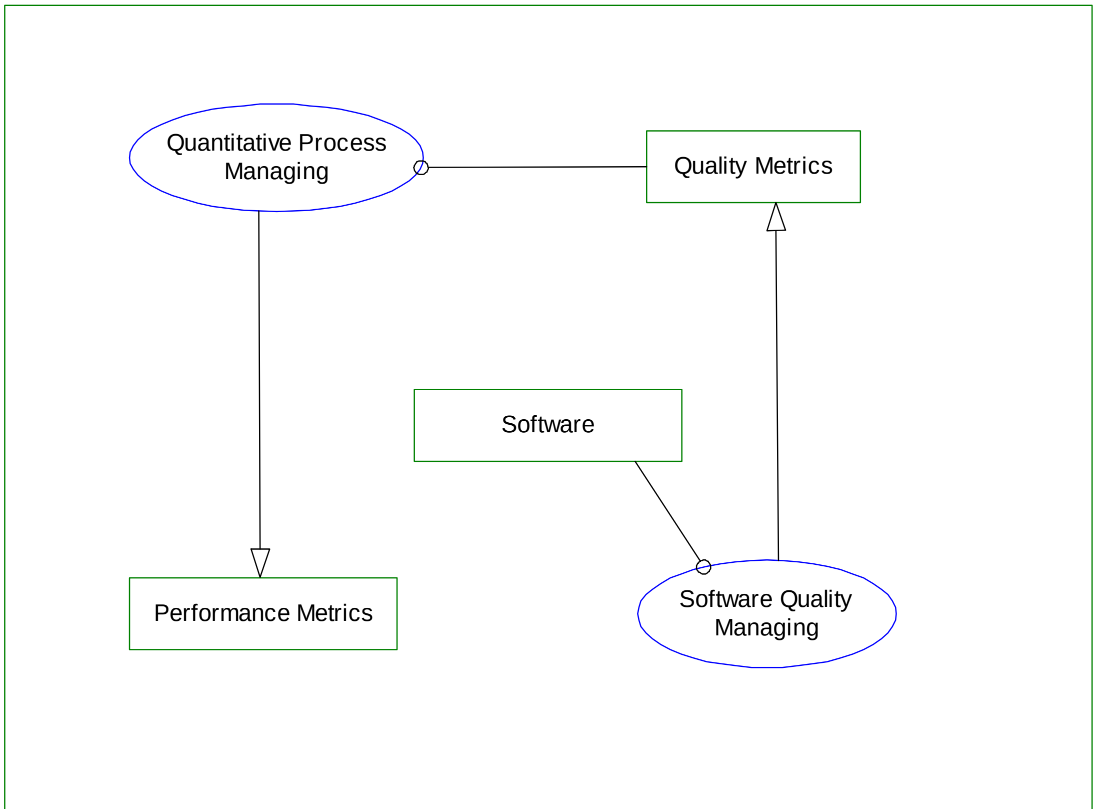


Figure 3-5: CMM Level 4 Organization

Figure 5: CMM Level 4 Organization

Quantitative Process Managing yields **Performance Metrics**.
Software Quality Managing yields **Quality Metrics**.
Quantitative Process Managing requires **Quality Metrics**.
Software Quality Managing requires **Software**.

Notice that this organization has a coupling between the processes. We are not sure if this was intentional but we can easily see the coupling using OPM.

What about Quality as an attribute of Software here too, with various values (levels) with Software Quality managing affecting it?

Level 4 concentrates on establishing a quantitative understanding of both the software process and the software products being built. The purposes of the key process areas are discussed below.

Quantitative Process Management: Control the process performance of the software projects quantitatively.

Software Quality Management: Develop a quantitative understanding of the quality of the project's software products and achieve specific quality goals.

A Level 5 organization is shown in below.

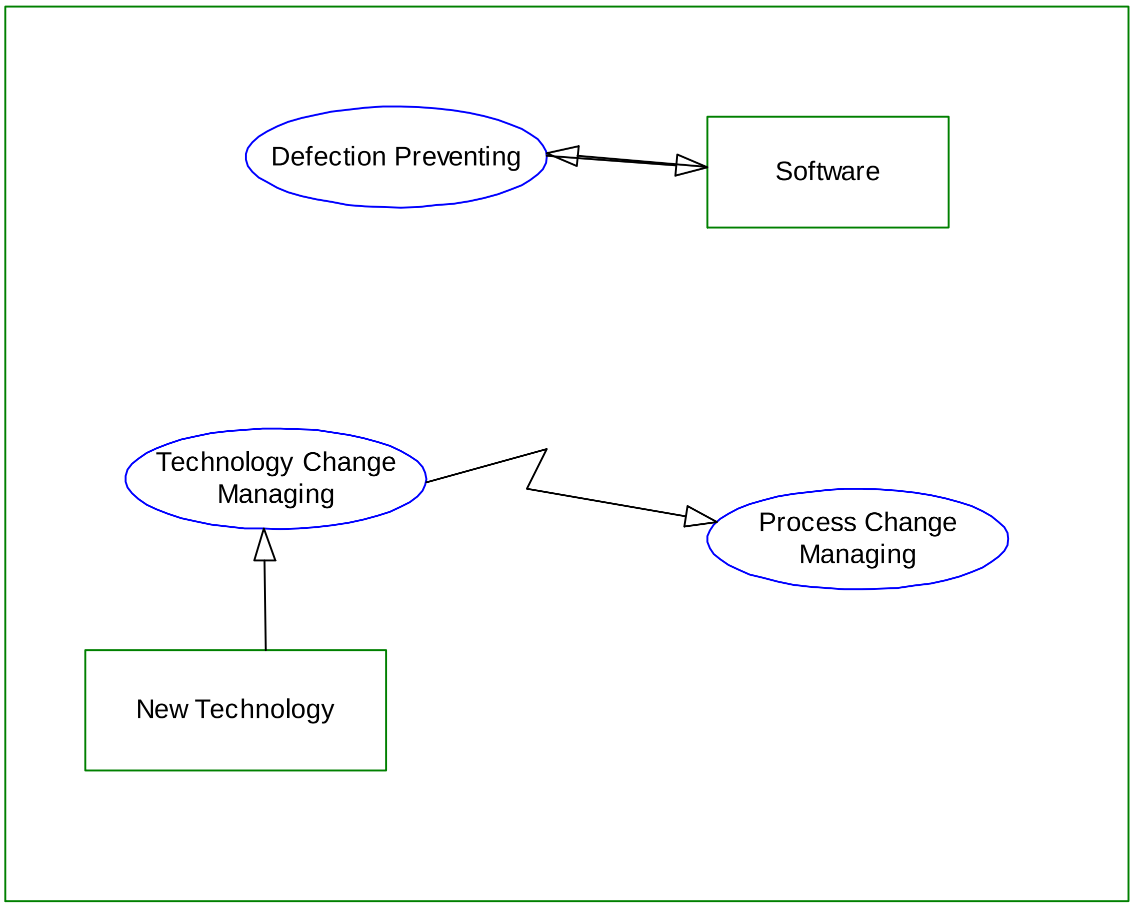


Figure 6: CMM Level 5 Organization

Technology Change Managing invokes **Process Change Managing**
Technology Change Managing requires **New Technology**.
Technology Change Managing yields.
Defect Preventing affects **Software**.

Level 5 concentrates on the issues that both the organization and the projects must address to implement continuous and measurable software process improvement. The purposes of the key process areas are listed below.

Defect Prevention: Identify the causes of defects and prevent them from recurring.

Technology Change Management: Identify beneficial new technologies (i.e. tools, methods, and processes) and transfer them into the organization in an orderly manner.

Process Change Management: Continually improve the software process used in the organization with the intent of improving software quality, increasing productivity, and decreasing the cycle time for product development.

The Architecture of the Evolution

The figure below describes the evolutionary nature of the architecture of the CMM process. Each maturity level builds on the previous one. Each Key Process Area (KPA) at level i must exist at level i+1 and higher. Some KPAs can also evolve, as shown below.

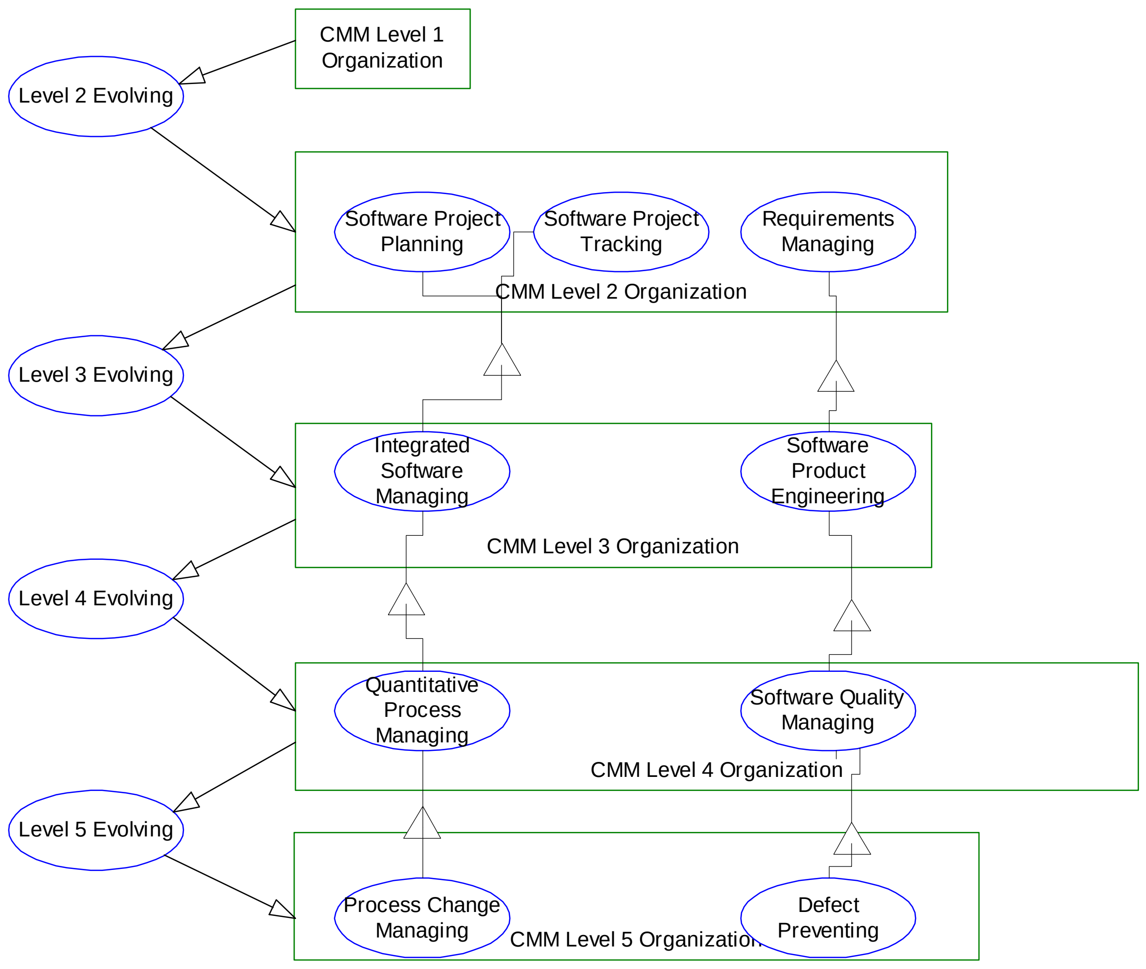


Figure 7: Architecture of CMM Evolution

Integrated Software Managing specializes **Software Project Planning** and **Software Project Tracking**.
Software Product Engineering specializes **Requirements Managing**.
Quantitative Process Managing specializes **Integrated Software Managing**.
Process Change Managing specializes **Quantitative Process Managing**.

Software Quality Managing specializes **Software Product Engineering**.
Defect Preventing specializes **Software Quality Managing.**
Integrated Software Managing specializes **Software Project Planning** and **Software Project Tracking**.
Software Product Engineering specializes **Requirements Managing**.
Quantitative Process Managing specializes **Integrated Software Managing**.
Process Change Managing specializes **Quantitative Process Managing**.
Software Quality Managing specializes **Software Product Engineering**.
Defect Preventing specializes **Software Quality Managing.**
Integrated Software Managing is **Software Project Planning** and **Software Project Tracking**.
Software Product Engineering is **Requirements Managing**.
Quantitative Process Managing is **Integrated Software Managing**.
Process Change Managing is **Quantitative Process Managing**.
Software Quality Managing is **Software Product Engineering**.
Defect Preventing is **Software Quality Managing.**

As you can see from the OPM, SPTO and SPP evolve into continues process improvement in Level 5. Requirements management evolves into defect preventing in Level 5. Many software defects are caused by not understanding requirements.

Does CMM Meet the Goals of the Architecture?

Now that we have studied the architecture, we can evaluate whether it meets the needs of the customer. The needs of a customer using CMM are categorized by the following. Does implementing the CMM make us more productive, have a quicker Time To Market (TTM), reduce our post-release defects, and is cost effective to implement? According to a study done by the SEI [4], the answer is yes. The table below summarizes the results.

Category	Range	Median	Data Points
Productivity gain/year	9%-67%	35%	4
Time to Market (reduction/year)	15%-23%	-	2
Post-release defects (reduction/year)	10%-94%	39%	5
Business value ration	4.0-8.8:1	5.0:1	5

Table 3: Customer Goals – Survey of Results

Another study also measured how expensive it was to implement CMM [5]. It cost on average $1,375 per engineer. One organization where the author works spends 20% of their departmental budget on CMM implementation.

The criticism of CMM has focused on the fact that an organization may become concerned only about the process and not about the product. In the same study [5], 96% of respondents disagreed that CMM was counterproductive, 90% disagreed that they neglected non-CMM issues, and 85% disagreed that they became more rigid and bureaucratic. Another criticism is that CMM organizations become risk-averse. According to the same respondents, only 42% of Level 1 organizations said they were risk-takes, versus 74% in Level 2, and 79% in Level 3 organizations. Thus, the more use of CMM, the higher the risk taking the company is.

For the second goal - making it easy to adapt to the CMM - the architects created the evolutionary levels where each level could build upon the previous one. This satisfies the goals but for most medium to small companies, the cost of $1,375 per engineer per year and the time to move to Level 2, which is 25.6 months on average, is too much of an investment for most small companies under 200 people. In fact, the smaller the company, the less relevant they found the CMM course material [5].

For the third goal – to reduce the amount of rework in software projects – we can measure how well budgets, schedules, and customer requirements are met. The percentage of respondents [5] found an over 20% increase in product quality, productivity, ability to meet schedules, ability to meet budgets, and staff morale. Customer satisfaction increased overall but went down when a company went from Level 1 to Level 2.

By using OPM, we observe many agent links in the CMM organizations. This is supported by the cost and time numbers reflected in the survey.

In summary, the CMM architecture does meet the goals. However, the architecture is too heavy for small companies to adopt. Perhaps CMM should only be meant for larger companies that need to control quality in a more sophisticated manner.

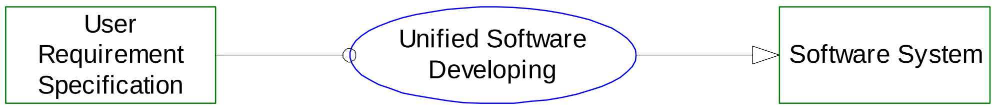
Unified Software Development Process

Another popular development process is based on the Unified Modeling Language and is called the Unified Software Development Process (USDP) [7]. The architectural goals of the process are as follows:

- To provide guidance to the order of a team's activities.
- To direct tasks of individual developers and the team as a whole.
- To specify what artifacts should be developed.
- To offer criteria for monitoring and measuring a projects' products and activities.

The top-level architecture of the process is shown in the figure below. This OPD could be generalized to all software development activities, whether using the Unified Process or not.

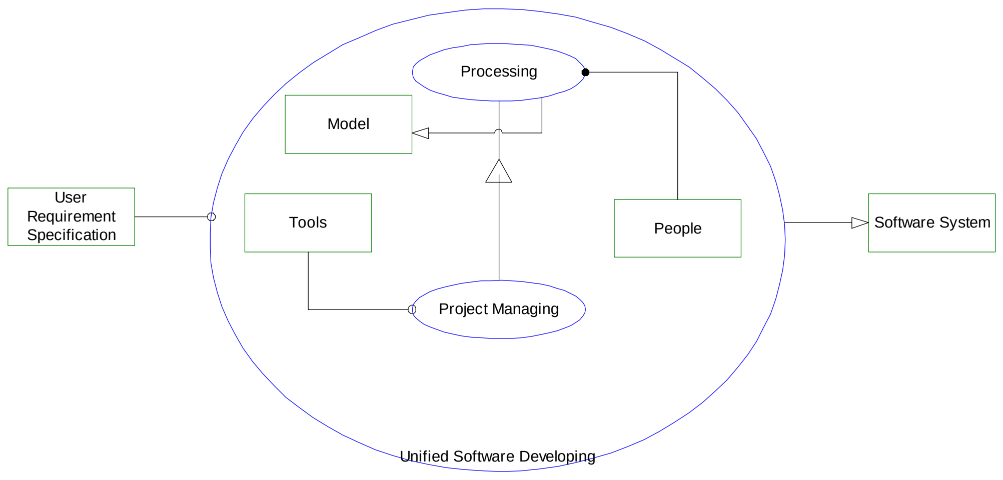
Figure 3-8: USDP Top Level Architecture



User Requirement Specification is required for **Unified Software Developing**.
Unified Software Developing produces **Software System**.

Requirements come in and the software system is produced. The figure below shows the process at the next level of decomposition.

Figure 8: The Unified Software Development Process Inside



Tools are required for **Project Managing**.
People handle **Processing**.
Processing yields **Model**.
Project Managing specializes **Processing**.

The important piece to note about this diagram is that the process is a template for the project. Each project will specialize the process for it's own purpose. Below shows the details of this template.

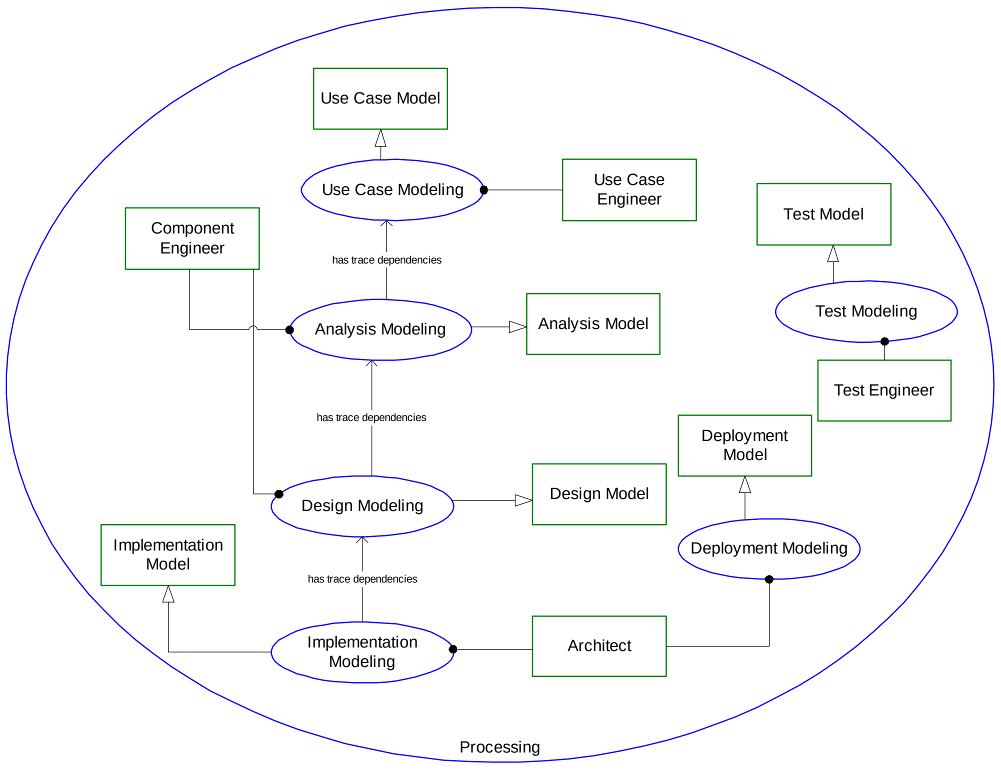


Figure 9: The Process Template

Test Engineer handles **Test Modeling**.
Component Engineer handles **Analysis Modeling** and **Design Modeling**.
Architect handles **Deployment Modeling** and **Implementation Modeling**.
Use Case Engineer handles **Use Case Modeling**.
Use Case Modeling yields **Use Case Model**.
Analysis Modeling yields **Analysis Model**.
Design Modeling yields **Design Model**.
Implementation Modeling yields **Implementation Model**.
Deployment Modeling yields **Deployment Model**.
Test Modeling yields **Test Model**.

Note that the process template is based on six different models. Each of these models makes use of specific Unified Modeling Language (UML) diagrams, which is why this process is tied to the UML. The output of the process template is a model. Below shows the various models that make up this model.

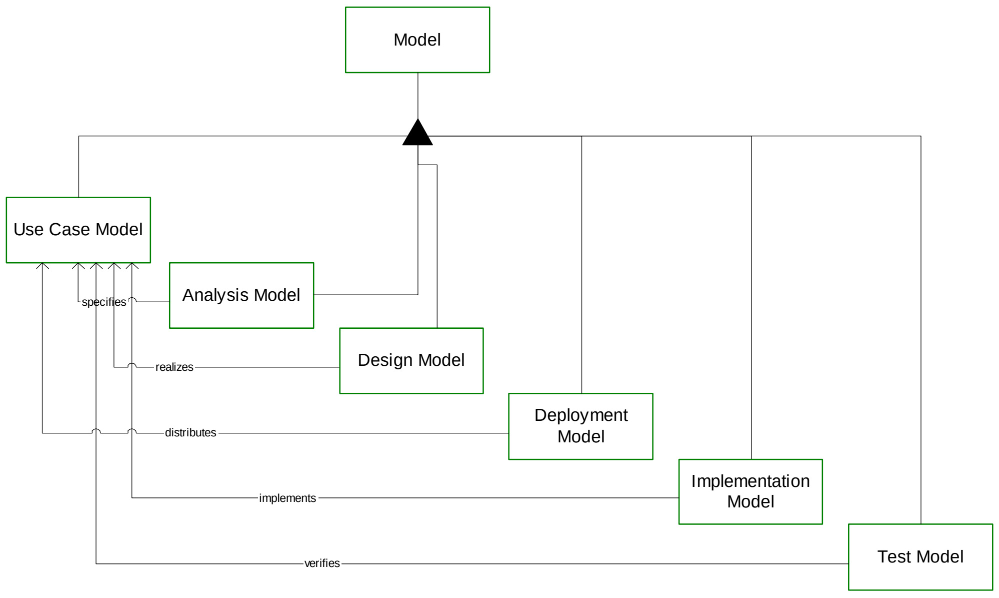


Figure 10: The Model Set

Model consists of **Use Case Model**, **Analysis Model**, **Design Model**, **Deployment Model, Implementation Model**, and **Test Model**.
Analysis Model specifies **Use Case Model**.
Design Model realizes **Use Case Model**.
Deployment Model distributes **Use Case Model**.
Implementation Model implements **Use Case Model**.
Test Model verifies **Use Case Model**.

All the models relate back to the Use Case Model, which defines the requirements of the system. The figure below describes the project management part of the process. This is a specialization of the process and also describes the life cycle of the process. Each phase produces iterations of the work products.

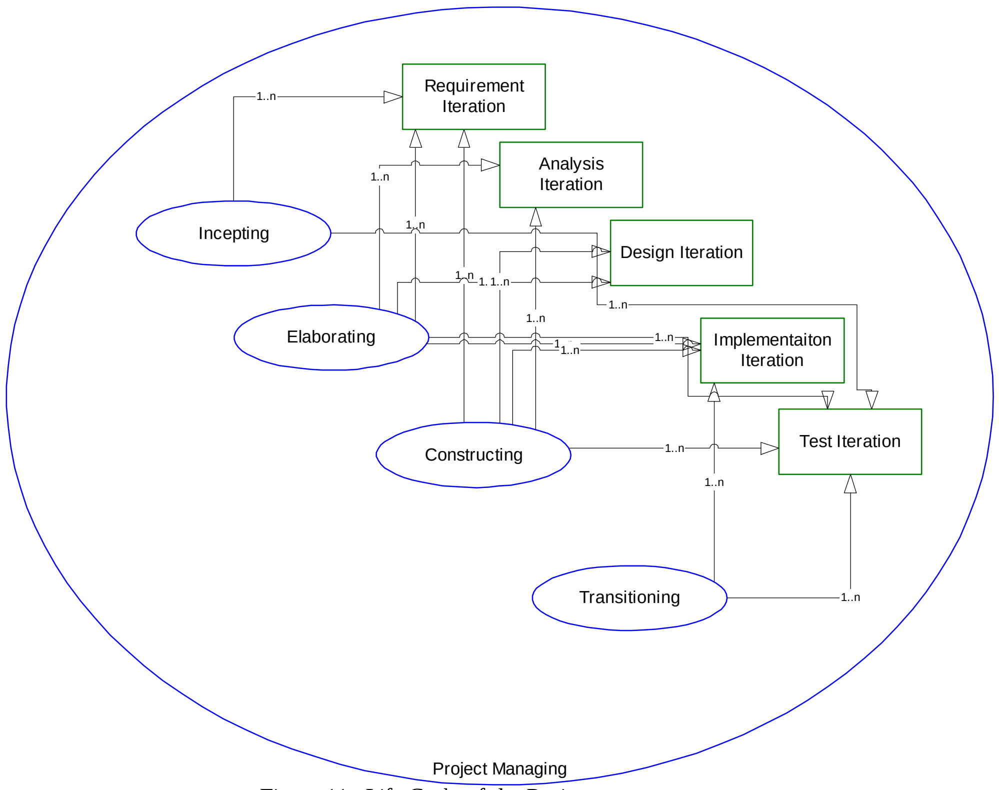


Figure 11: Life Cycle of the Project

Incepting yields 1 to n **Requirement Iteration** and 1 to n **Test Iteration**.
Elaborating yields 1 to n **Requirement Iteration**, 1 to n **Analysis Iteration**, 1 to n **Design Iteration**, 1 to n **Implementation Iteration** and 1 to n **Test Iteration**.

Constructing yields 1 to n **Requirement Iteration**, 1 to n **Analysis Iteration**, 1 to n **Design Iteration**, 1 to n **Implementation Iteration** and 1 to n **Test Iteration**.
Transitioning yields 1 to n **Implementation Iteration** and 1 to n **Test Iteration**.

The USDP is an iterative process, which produces various sets of work products throughout the life of the product.

Does USDP Meet the Goals of the Architecture?

The table below reiterates the goals of the architecture.

Goal	*Metric*
To provide guidance to the order of a team's activities	Order of activities – was it done that way on a real project and was it effective?
To direct tasks of individual developers and the team as a whole	On real project – measure the tasks specified – were any tasks left out - % of tasks specified.
To specify what artifacts should be developed	List and completeness of artifacts specified.
To offer criteria for monitoring and measuring a projects' products and activities	Measure criteria against a real project – and if it contributed to the success of project

Table 4: Goals of the USDP Architecture

Although there is no research on the effectiveness of this particular process, the first goal is common to many software processes, such as the one described in [6]. The successes of an iterative phased approach to development are described in [10]. For the second goal, although we have not measured this on a real project, the OPM describes the work processes and products of the key engineers, so it does meet the second goal in theory. Similarly, for the third goal, the process clearly specifies what work products to be produced. The one goal that the process fails to meet is the fourth one. Specific criteria are never mentioned in the book.

Therefore, the architecture of the USDP meets three out of the four goals.

Comparison of CMM and USPD

To compare the processes, we first compare the goals of the two processes, which are distinctly different.

Goal of USPD	*Goal of CMM*
To provide guidance to the order of a team's activities	Process should allow software to meet needs of customers
To direct tasks of individual developers and the team as a whole	Allow for easy adoption of the process
To specify what artifacts should be developed	To reduce the amount of rework in a software project
To offer criteria for monitoring and measuring a projects' products and activities	

Table 5: Architectures Comparison

The goals of USPD, as stated in the specification [7], are not as high level as they should be. For example, why would you want to provide guidance and direct tasks? CMM better states these goals as any software process has the goal of better meeting customer needs. The CMM goals are stated at a much higher level without any implementation. The USPD goals have some implementation mixed in with the mention of artifacts and tasks.

Another method of comparison is to measure the complexity of a process. OPM provides a way to measure this. Let each process or object or "thing" have 2 points and each link have 1 point. The higher the number of the diagram, the higher the complexity is displayed. If we measure this for a Level 2 organization, we get a complexity measure of 50. If we measure the template of the USPD, we get a complexity measure of 46.

Overall, the USPD diagrams are less complex, and use fewer agents then the CMM diagrams.

Another key difference is the CMM does not specify how to implement the CMM. It gives a description for each process area, but leaves it up the organization to design an implementation. The USPD describes the process in more decomposition shown here, and brings in the UML at later levels.

Recommendations

- CMM has mostly agents (humans) involved. This process is very costly and may benefit by having more automation or instrument links in the architecture.

- The CMM process specification did not describe clearly the output of the processes. We were able to surmise them by the description but OPM forces the issue, giving a framework for clearly specifying the architecture. The CMM could benefit from clearly specifying the input and output of the processes.

- Smaller companies might better use CMM if it were less expensive to implement. The biggest hurdle seems to be in Level 2. The level of complexity of this diagram is relatively high. Moreover, according to [5], the effectiveness of CMM as surveyed is shown below.

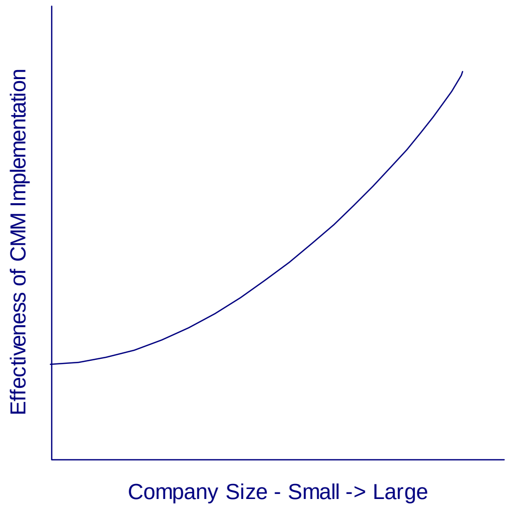


Figure 12: CMM Effectiveness vs. Company Size

- The USDP, although specified more completely, also has a relatively high level of complexity. Nevertheless, the iterative nature of the process is more in tune with modern successful software practices [10].

- Although we don t have the data, we believe small companies would be unlikely to implement all the models specified in the USPD. We recommend reducing the number of models in some implementations.

An important factor with any process is how it is implemented. The merits of a process are many [7]:

- Everyone on the development team can understand what he or she has to do to develop the product.
- Developers can better understand what other developers are doing.
- Supervisors and managers can understand what developers are doing.
- Developers, supervisors, and managers can transfer between projects without having to relearn a new process.
- Training can be standardized.
- The course of the software development is repeatable and thus can have predictable schedules and costs.

The problem with any process when implemented is that if it is too rigid, it becomes ineffective as shown below.

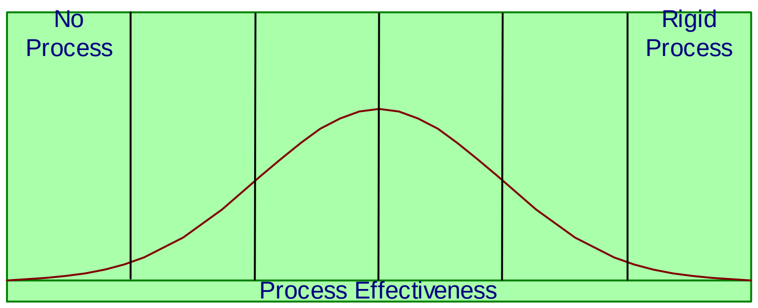


Figure 13: Process Effectiveness

Conclusion

OPM has successfully been used to analyze the architecture of two processes. Because of the precise nature of OPM, the architecture could be examined in ways even the architects could not. The combination of OPD and OPL provides a exact way to produce a specification. It is easy to use natural language imprecisely, but OPL, although natural in language, is precise in its 'use. Thus, specifications are clear, exact, and naturally stated

References

1. Back, J., *Software Herosism.* IEEE Software, 1995.

2. Card, D., *Sorting out Six Sigma and the CMM.* IEEE Software, May/June 2000.

3. Crawley, E., *Form and Function, Systems Architecture Class, ESD.34, MIT*, . 2000.

4. Herbsleb, J., et.al, *Benefits of CMM-based software process improvement: Initial Results.* SEI, CMU, 1994.

5. Herbsleb, J., et.al., *Software Quality and the Capability Maturity Model.* Communications of the ACM, June 1997. **40**(6): p. 30-40.

6. Highsmith, J., *Adaptive Software Development*. 2000: Dorset House Publishing.

7. Jacobson, B., Rumbaugh, *The Unified Software Development Process.* 1999: Addison Wesley.

8. Krishnan, M.S.a.K., Marc, *Measuring Process Consistency: Implications for Reducing Software Defects.* IEEE Transactions on Software Engineering, November/December 1999. **25**(6): p. 800-815.

9. Paulk, M., et.al., *The Capability Maturity Model: Guidelines for Improving the Software Process*. 1994: Addison Welsey.

10. Preston G. Smith, e.a., *Developing Products in Half the Time*. 1998: John Wiley & Sons.

C h a p t e r   3

REQUIREMENTS AND DESIGN OF AN OPM TOOL

Chapter 3: Requirements and Design of an OPM Tool

Introduction

The OPM tool has requirements and a design to satisfy those requirements. This chapter first will detail requirements.

Two specific aspects of the design will be further broken down. First, the user interface will be discussed in detail. Second, the algorithm, which handles the translation between OPD and OPL, will be shown along with a prototype.

Model Driven Architecture

Model-Driven Architecture (MDA) is a comprehensive standard framework
for understanding, development and life cycle support of systems that comprise software, hardware, humans, and business practices.

MDA specifies standard approaches to describing, developing and maintaining these systems and provides standardized methods that serve as building blocks
to support the various system-life cycle phases.

OPM is a MDA methodology. The OPD diagrams and natural English OPL can understand by interdisciplinary teams.

Requirements

Over the past six months of work on this thesis, we have gathered the following top-level requirements for an OPM tool.

High Level Requirements

1. OPD - a graphics program is required to represent a complete set of OPD elements. A list of all the graphical sentences (collections of one or more OPD symbols linked in a particular arrangement) will be created along with their corresponding OPL sentences. This is called "the OPD-OPL tool".

2. The OPD and OPL need to persist after creation. A database will be needed for storing OPD elements and language.

3. There must be bi-directional translation between OPD and OPL.

4. The OPD-OPL tool needs work with distributed groups. The database must be able to handle multiple accesses from different locations so various people can work on different sub-systems of the same system simultaneously by agreement on predefined interfaces

5. The OPD-OPL tool should have knowledge management system to be able to manage the database. Search tools, report generation, simulation, and real-time constraint specification for process duration, state duration and state-transition duration. Constraint modeling would be part of this system.

6. The OPD-OPL tool must be extendable by users in terms of domain-specific symbols and links and the corresponding OPL sentences. This would require a user interface for users to extend OPD / OPL.

7. The OPD-OPL tool must be extendable by programmers. This would allow them to build translators to other languages such as C++, JAVA, XML or OCL.

· The OPD-OPL tool must be able to translate from OPL to UML (using OCL?)

· The OPD-OPL tool must be able to translate from OPL to C++, JAVA, and XML

8. The OPD-OPL tool must be portable, i.e. OS independent.

9. A development environment must be established for the work done on the tool by a group of software engineers.

10. The OPD-OPL tool must have the ability to simulate the design. The simulation needs to be graphic as well as textual in accordance with the OPM philosophy.

11. The OPD-OPL tool must be able to export diagrams and sentences into Word and Excel documents for automatic document generation. It also must have the ability to export text only sentences.

User Interface

- If there is any ambiguity on which sentence to generate or graphical item to choose, the user will be asked to choice between the possibilities using a drop-down dialog box listing the choices.

- The recognizer consists of three basic steps: input, check, and translate.

- Each thing in the OPD and OPL should have the option to be linked to any multimedia item: image, video, spreadsheet table, text file, engineering drawing; a right click will open a drop down menu shown what exist for that thing so the user can open it and get a clear idea what the thing is, how it looks, behaves, what it does, how defined, etc.

OPL

- The OPL string will have the following rules:
    - Objects, processes and states will be bold.
    - Objects and process will begin with capital letters, states will not.
    - Processes will always end in a word that ends in "ing".
    - Relation text will be in bold.

    - All OPL keywords will not be in bold

OPD

- Structural relations should be more easy to denote: click on a triangle symbol then get a dialog box telling you to highlight the root (whole or general or characterized or class) thing then telling you to highlight the set of one or more descendants (parts or specializations or features or instances) by shift-click or mouse drag over area that includes these things, then automatically place the triangle and all the (ortho-normal) links to the father and sons.

- The tool must be able to do square lines for all the relationships (maybe switch between square and straight or curved).

- Ability to query graphically - generate new OPD around a set of one or more specified things, denoting as a parameter the Hamiltonian distance ("radius") from each thing should be included in the query

- Text should along links as in OPCAT (the original OPM prototype) but should not disappear if space is too small.

- The participation constraints should be attached as small boxes to the edges of the lines in the fundamental links that touch the things. They can be not only to the structural links but also to procedural as in "Mixing requires 3 Mixers" or "Packing yields between 6 to 10 Packages."

- Physical and environmental objects are represented by 3-D and dotted lines and can be mixed together and with the solid line and non-3-D internal and informatical objects. An algorithm is needed to align things and links in an OPD. Manual help optional to realign/rearrange/move/resize- will be needed always. Various link routing modes (curves, splines, many knees, manually changing knee locations to avoid crossing things). Automatic line crossing disambiguating, ability to switch line modes. Stay with ortho-normal modes to structural and all the rest for procedural (gives the feeling of flowing)

- Faithful graphical zooming in/out with care for fonts is needed. Easy change of fonts, size color style superscript etc. (word proc. options).

- Simulation graphics - flashing, flowing in lines, time control, etc will be required for future development.

Navigation

- Filtering of the language and the diagrams would be useful. This is form of encapsulation that would allow a user to easily filter specific items or types of items both in OPD and OPL for particular audiences.

- Ability to abstract, not just detail - by pointing at a set of things and require a new OPD that has a new thing which is their father but without the sons. (Both zooming out and folding for both objects and processes, and state suppressing for states). There should be an option to do this but remain in the same OPD if there is already at lease one OPD that has the more detailed picture (that is, that we warn before deleting captured details)

- Automatically detect and denote partial aggregation, and the rest of the fundamental structural relations, when there are subsets of the sets of fundamental descendants (parts, specs, features or instances).

- Ability to query and see a complete list of fundamental descendants (parts specs, features or instances) or any combination thereof, both graphically and textually.

1. Continuous updating of the SysteMap with links from the source thing to the destination thing - always in the direction of detailing (even if we do abstracting as mentioned above). Ability to move SysteMap nodes manually to enhance clarity. Algorithm to make it clear.

2. One of the items on this menu will be a list of all the children - parts, specs, features and instances of it.

Data Storage

- OPD is non-linear and OPL is linear and we will in some instances combine OPL sentences to reflect more concise graphic patterns in OPD. To start with, we will just output the simple sentences but we should keep in the back of our mind this requirement

- Continuous Consistency checking with respect to the evolving system OPDBMS database. The Object Process database management system (OPDBMS) is a database created specifically for the type of data we need to store.

Patterns

- An error-handling pattern (both for standard return codes and exception handling) would be useful for the pattern library.

Queries

- Query all the info related to a certain thing in various crosses or sorting.

Portability / Translation

- The tool must be integrated with MS Word for a complete documentation set which will consist of OPL and any commentary added by user.

- Filter and/or convert to generate the various UML diagrams as well as other types of diagrams.

- Code generation - Java + XML first, then C++.

Design

We have used OPM to design the OPM tool. This design is called the "Meta-Language" of the OPM. We need to start with a "kernel" of OPM to do this and we chose objects, processes, and relations.

OPM was a natural way to discuss the design between all parties – not much background was needed. Requirements often came to us at different levels of the hierarchy. OPM was able to capture this.

The OPD in the OPM figure below and the associated OPL is a description of the OPM tool, cited from a patent pending by D. Dori.

The core of the Learner is the OPM kernel. The OPM kernel is the basics needed for OPM. Using the Learner, a user can extend the OPM kernel to create symbols specific to their architecture. For example, a chemical processing architecture added a new link to specify a manufacturing transformation between objects.

The OPM Kernel consists of Objects, Processes, States, the transforming connections, the enabling connections, and the structural relations. This is shown in following three tables, reproduced from [1].

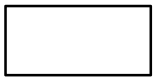
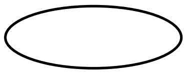
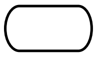
Things		
Object	*Process*	*State*
		
A thing that exists at some state	*A thing that changes object state*	*Where the object is at a point in time*

Table 6 OPM Entities

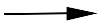
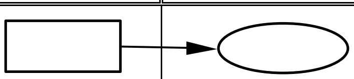
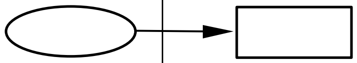
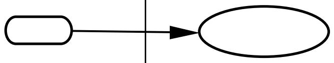
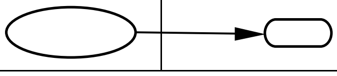
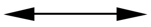
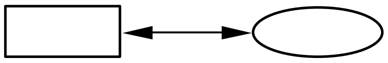
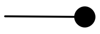
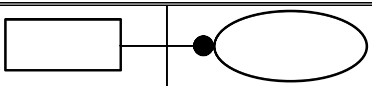
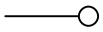
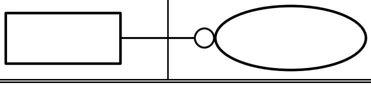
Type	Name	Symbol	Source	Destination
transforming	*Consumption*			
	Result			
	Input			
	Output			
	Effect			
enabling	*Agent*			
	Instrument			

Table 7 – Procedural Links

Type	Name	Symbol
general	*Unidirectional*	$\xrightarrow{\text{forward name}}$
	Bi-directional	$\underset{\text{backward name}}{\overset{\text{forward name}}{\rightleftharpoons}}$
fundamental	*Aggregation-Participation*	▲
	Exhibition-Characterization	◬
	Generalization-Specialization	△
	Classification-Instantiation	⧊

Table 8 – Structural Relations

The top-level model is described in the following figure. The corresponding OPL is listed after the model.

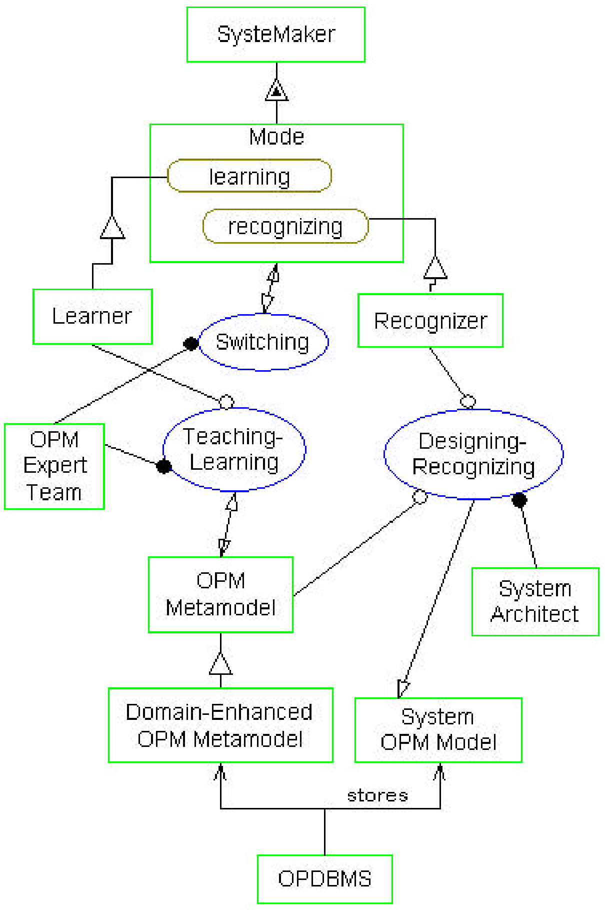


Figure 14 – The Top Level Diagram for an OPM tool

SysteMaker exhibits **Mode.** *(Exhibition sentence)*
Mode can be **learning** or **recognizing.** *(State enumeration sentence)*
Learner is **SysteMaker,** the **Mode** of which is **learning.** *(Exhibition sentence)*
Recognizer is **SysteMaker,** the **Mode** of which is **recognizing.** *(Exhibition sentence)*
OPM Expert Team handles **Switching** and **Teaching-Learning.** *(Compound agent sentence)*
Teaching-Learning requires **Learner.** *(Instrument sentence)*
Teaching-Learning affects **OPM Metamodel.** *(Effect sentence)*
Domain Enhanced OPM Metamodel is an **OPM Metamodel.** *(Specialization sentence)*
System Architecting Team handles **Designing-Recognizing.** *(Agent sentence)*
Designing-Recognizing requires **Recognizer.** *(Instrument sentence)*
Designing-Recognizing yields **System OPM Model.** *(Result sentence)*
Domain Enhanced OPM Metamodel is an **OPM Metamodel.** *(Specialization sentence)*
OPDBMS stores OPM Metamodel and **OPM System Model.** *(Compound structure sentence)*
Graphics Learner is a **Graphics Window.** *(Specialization sentence)*
Text Learner is a **Text Editing Window.** *(Specialization sentence)*
Graphic Sentence Defining is a process**.** *(Simple Process sentence)*
Formal English Sentence Defining is a process**.** *(Simple Process sentence)*

Notice that the specification of the tool is created in the OPL, which exactly matches the OPD. The method allows designers too look at designs via natural language specification and diagrammatically.

The meta-model in the OPD and OPL describes OPM. No extra text was needed to describe the system. The OPL and OPD described the design completely using consistent, unambiguous language.

Design of a Parser

The OPM tool allows for users to either start using OPD or OPL. A tool was created in PERL for parsing OPDs and generating OPLs and for taking OPLs and producing OPDs. The code and output from this work are in Appendix A.

Design Chapter Summary

In this chapter, we have laid out high level and second level requirements. From there, we designed an OPM tool using an OPM Meta-Model. The design was fully specified using OPL and no extra text was needed. This OPM (the set of OPD and OPL) can be used to communicate the design to a whole array of people – managers, engineers, testers – that will be needed to develop the tool.

References

1. Dori, D., *Object Process Methodology: A Holistic Systems Development Paradigm.* 2001: Sringer Verlag.

Chapter 4

TOOL IMPLEMENTATION

Chapter 3: Tool Implementation

Introduction

After completing requirements analysis for the OPM tool, we went on to determine a design for the tool. There are three main aspects we felt were important in the design:

1. Dori [3] has details for each graphical element that need to be followed. For example, the agent and instrument link should be just touching the object or process. The correct figure is shown below on the left, while the incorrect is shown on the right. The OPD specification has hundreds of details for the graphics that are important to serve the vision of the look and feel of the tool.

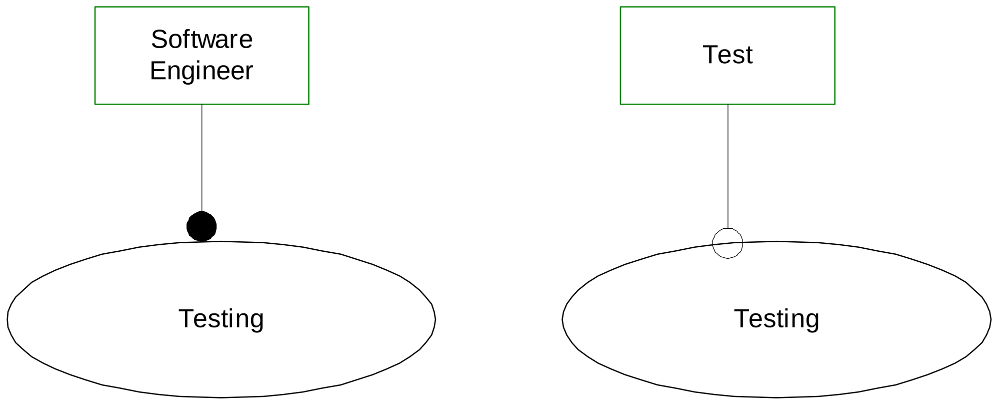


Figure 15 – The agent link on the left is correctly attached to the process while the instrument link on the right is incorrect; it should be touch the edge of the process ellipse with the edge of its 'circle.

2. The tool must allow for collaborative development. The vision of OPM is that it will be used between the different groups in an organization. To do this, it must be able to allow multiple people work the diagrams simultaneously. This is a similar requirement for a distributed database.

3. The learner is a key design piece to allow organizations to extend the architectural language and diagrams for their specific needs. This also allows the OPM kernel to be extended easily.

This chapter will address these three main design issues and detail a proof-of-concept that was developed for the OPM tool. Issue 3 was mostly detailed in the previous chapter but will be outlined here on how it could be done given our chosen solution. The emphasis will be on issues 1 and 2.

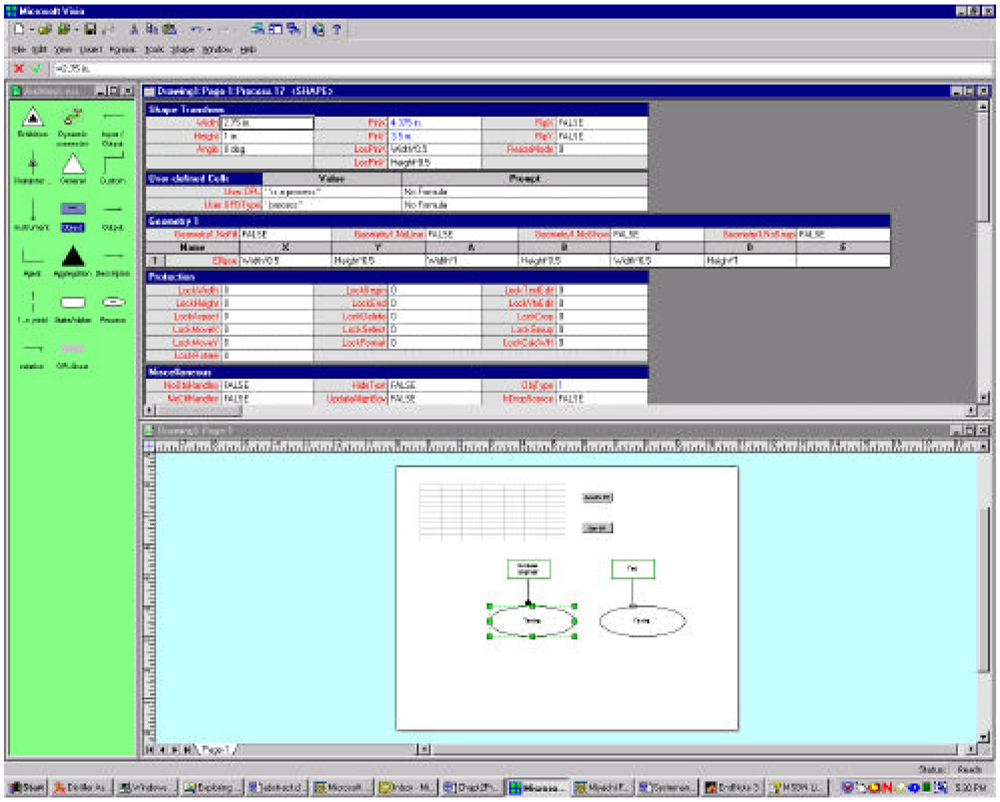
Graphical Details of OPD

The graphical details are an extremely important feature for the look and feel of the OPM tool. Rather than program them ourselves, we decided to build upon an existing graphics tool. The tool we chose was Microsoft VISIO® 2000. Visio® 2000 is entirely programmable. We used Visual Basic for Applications (VBA) to program our tool. VISIO® allows for shape geometry and behavior to be finely controlled by a designer.

VISIO® has 2-D shapes and 1-D shapes. 1-D shapes are typically connectors. In OPM, there are only three 2-D shapes: Objects, Processes, and States. States are only allowed to be grouped inside Objects. The links and structural relates are all connectors in VISIO® [1].

Moreover, the OPL can be automatically generated using VBA. VBA allows programmers to write code that will examine connectors and shapes. From this, logic can be added that will determine what connectors connect to which shapes.

We originally thought that we needed a database to associate the OPL text with the OPD. But the structure is more like a table. In VISIO®, we are able to associate data with shapes and connectors. This turned out to be enough.



Drawing1:Page 1:Process.17 <SHAPE>

Shape Transform					
Width	2.75 in.	PinX	4.375 in.	FlipX	FALSE
Height	1 in.	PinY	3.5 in.	FlipY	FALSE
Angle	0 deg.	LocPinX	Width*0.5	ResizeMode	0
		LocPinY	Height*0.5		

User-defined Cells	Value	Prompt
User.OPL	"is a process"	No Formula
User.OPDType	"process"	No Formula

Geometry 1								
Geometry1.NoFill	FALSE	Geometry1.NoLine	FALSE	Geometry1.NoShow	FALSE	Geometry1.NoSnap	FALSE	
Name	X	Y	A	B	C	D	E	
1 Ellipse	Width*0.5	Height*0.5	Width*1	Height*0.5	Width*0.5	Height*1		

Protection					
LockWidth	0	LockBegin	0	LockTextEdit	0
LockHeight	0	LockEnd	0	LockVtxEdit	0
LockAspect	0	LockDelete	0	LockCrop	0
LockMoveX	0	LockSelect	0	LockGroup	0
LockMoveY	0	LockFormat	0	LockCalcWH	0
LockRotate	0				

Miscellaneous					
NoObjHandles	FALSE	HideText	FALSE	ObjType	1
NoCtlHandles	FALSE	UpdateAlignBox	FALSE	IsDropSource	FALSE

Figure 16 – A VISIO® Shape Sheet associated with a process

The figure above shows part of a shape sheet associated with the processes selected in the diagram. The User-Defined Cells show two types that were added by us to define a process. The first is the OPL associated with the process. The second is the OPD type. These cells are required for all OPD shapes – both 2-D and 1-D "things" and "links". "Things" are called "shapes" in VISO and "links" and "relations" are called connectors.

VISIO® Proof-of-Concept

This section will outline the proof-of-concept using the VISIO® tool. VBA code was written to produce this tool.

The figure below shows the OPM template. The figure below shows the new type of diagram called an OPM.

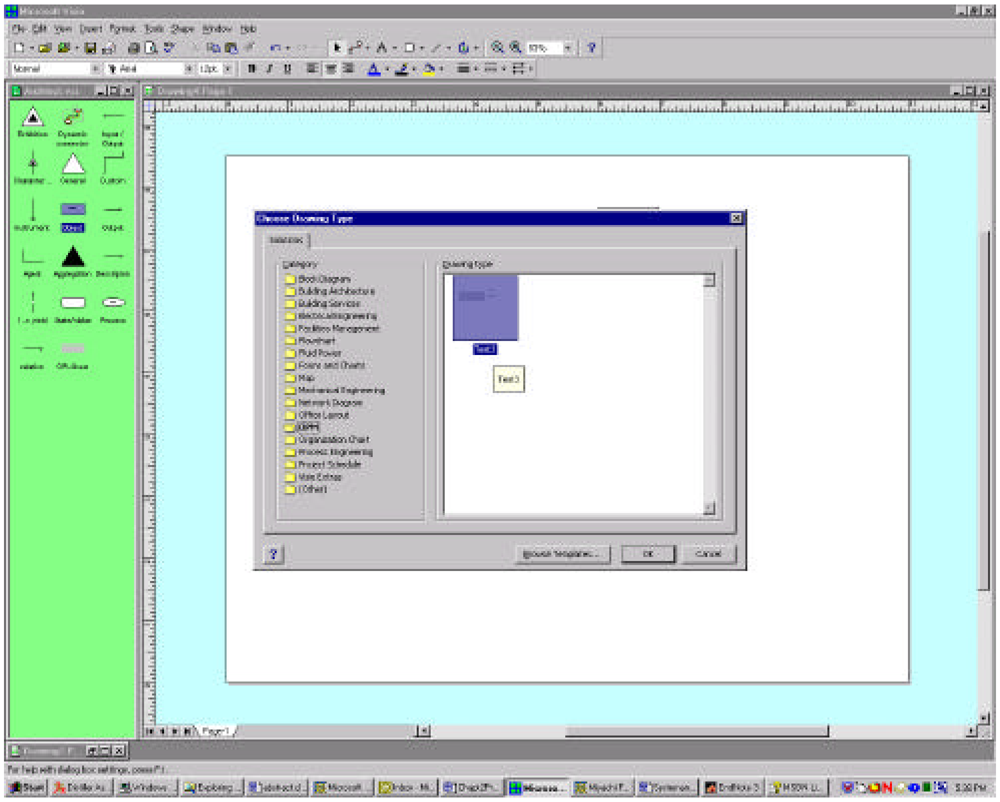


Figure 17 – When the user wants to create a new diagram, OPM is a new diagram type they can select.

Once this type is chosen, an empty screen comes up.

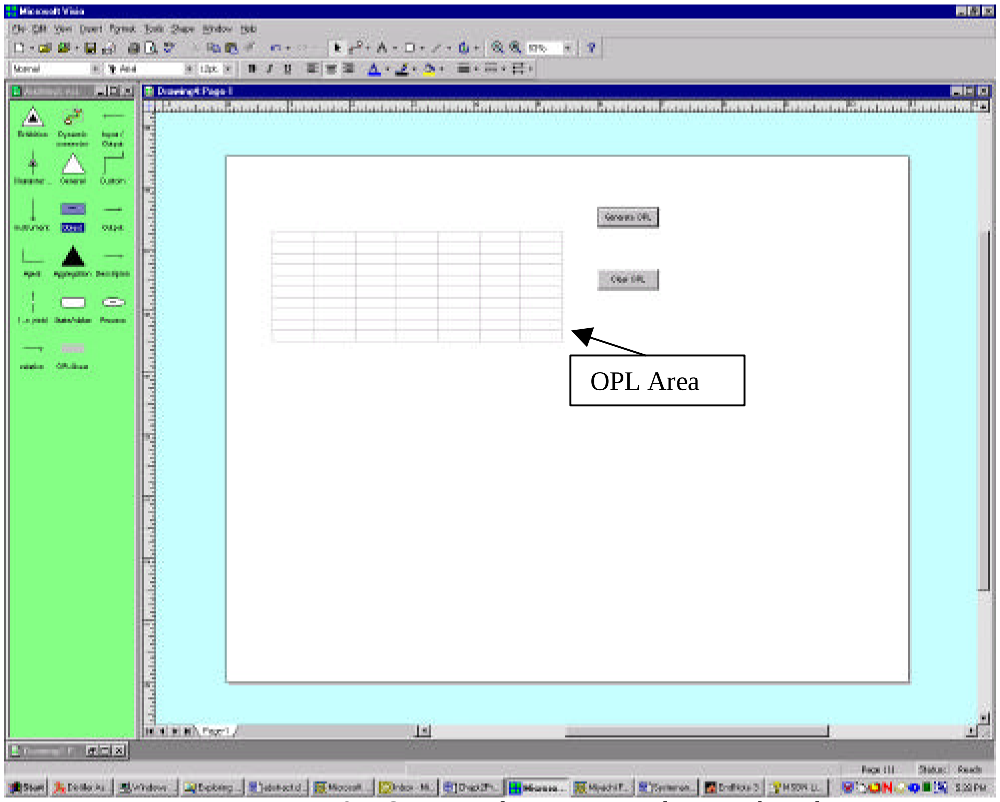


Figure 18 – OPM new drawing. Notice the area where the OPL will be generated.

The green area on the left shows the OPD stencils. This area is not complete.

As a demonstration, we will add a process. When we do this a dialog box appears.

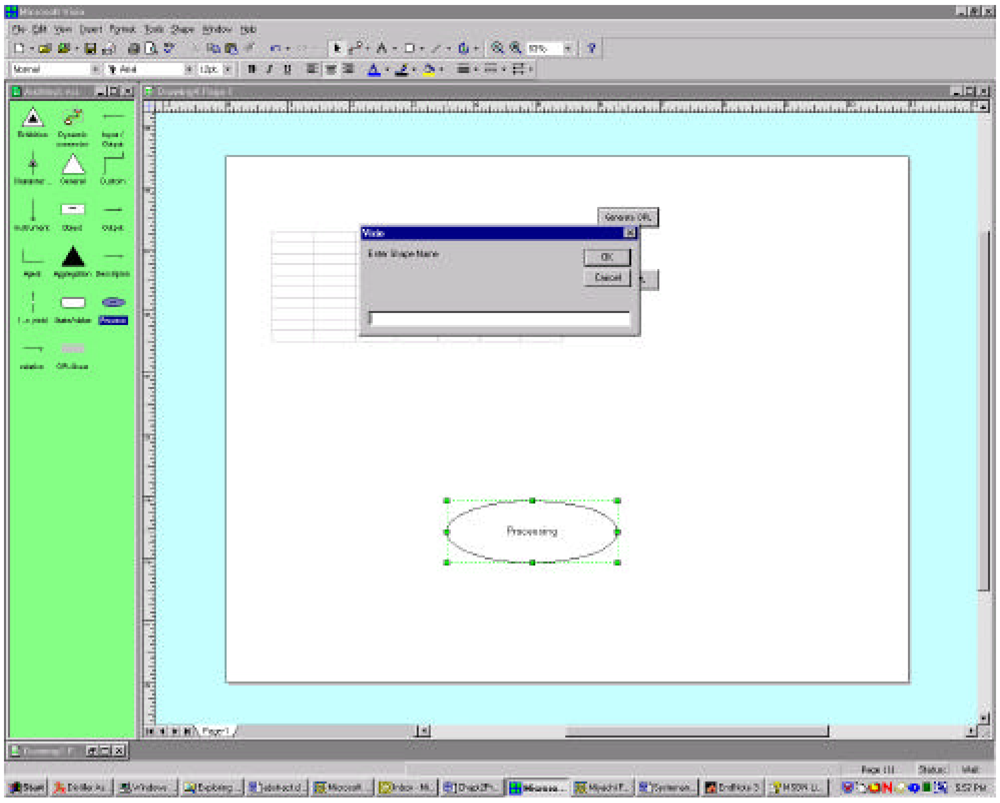


Figure 19 – Dialog box querying the user for the name of the process.

Each process must begin with a capital letter and end with "ing" although there can be more than one word.

Two objects are also added.

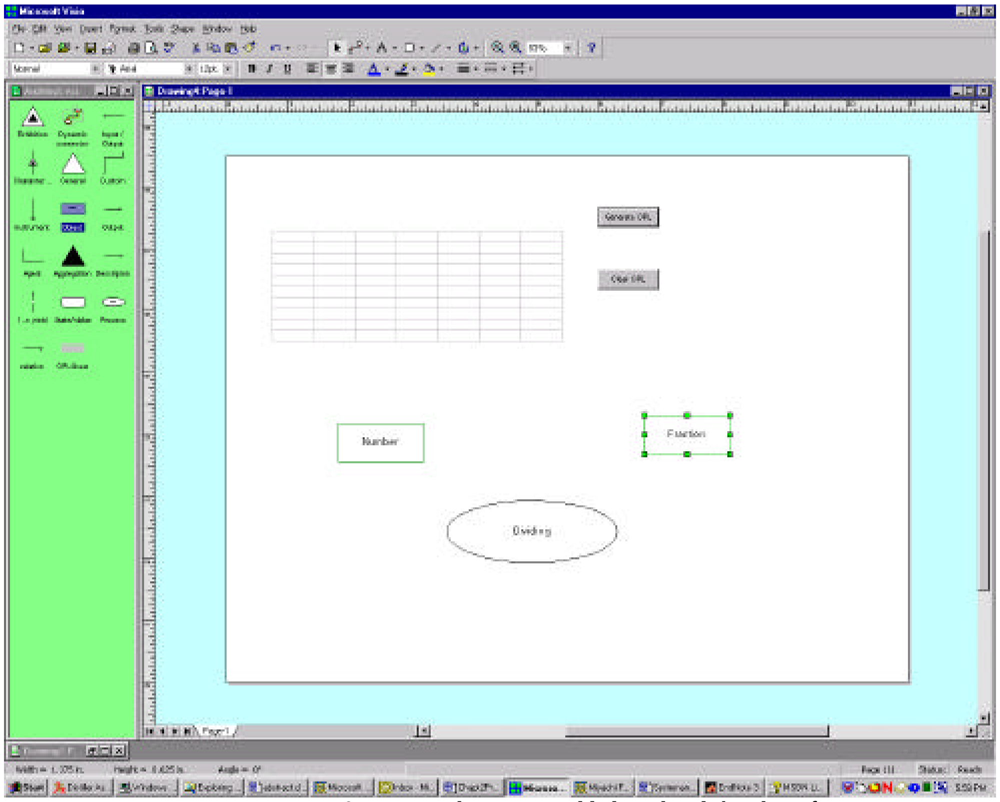
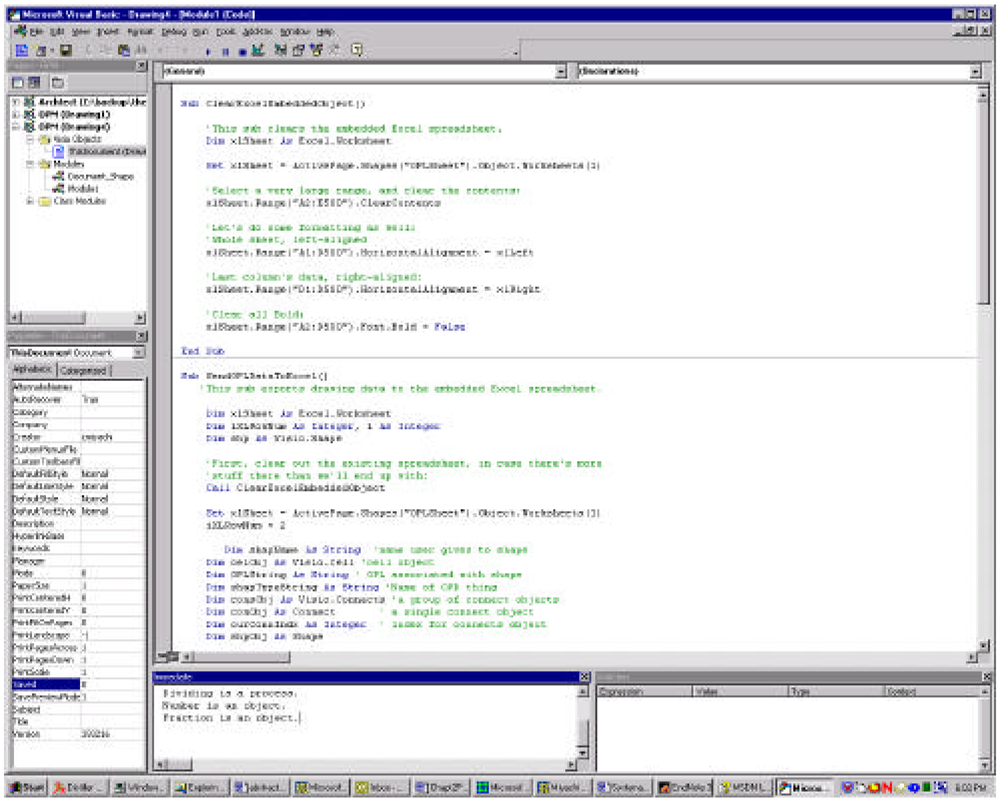


Figure 20 – Two objects are added. The dialog box for naming the object is not shown.

Objects must begin with a capital letter and be nouns. But they also can be multiple words.

In the Visual Basic window, internally, the OPL for the things are generated internally the moment the thing is dropped onto the page.



```
Sub ClearExcelEmbeddedObject()

    'This sub clears the embedded Excel spreadsheet.
    Dim xlSheet As Excel.Worksheet

    Set xlSheet = ActivePage.Shapes("OPLSheet").Object.Worksheets(1)

    'Select a very large range, and clear the contents:
    xlSheet.Range("A2:E500").ClearContents

    'Let's do some formatting as well:
    'Whole sheet, left-aligned
    xlSheet.Range("A1:D500").HorizontalAlignment = xlLeft

    'Last column's data, right-aligned:
    xlSheet.Range("D1:D500").HorizontalAlignment = xlRight

    'Clear all Bold:
    xlSheet.Range("A2:D500").Font.Bold = False

End Sub

Sub SendOPLDataToExcel()
   'This sub exports drawing data to the embedded Excel spreadsheet.

    Dim xlSheet As Excel.Worksheet
    Dim iXLRowNum As Integer, i As Integer
    Dim shp As Visio.Shape

    'First, clear out the existing spreadsheet, in case there's more
    'stuff there than we'll end up with:
    Call ClearExcelEmbeddedObject

    Set xlSheet = ActivePage.Shapes("OPLSheet").Object.Worksheets(1)
    iXLRowNum = 2

       Dim shapName As String  'name user gives to shape
    Dim cellObj As Visio.Cell 'cell object
    Dim OPLString As String ' OPL associated with shape
    Dim shapTypeString As String 'Name of OPM thing
    Dim consObj As Visio.Connects 'a group of connect objects
    Dim conObj As Connect         ' a single connect object
    Dim curConnIndx As Integer  ' index for connects object
    Dim shpObj As Shape
```

```
Dividing is a process.
Number is an object.
Fraction is an object.
```

Figure 21 – OPL sentences that are not connected are generated internally as shown in the "Immediate" window.

A unidirectional relation tag is added between objects. This text must be a verb phrase that relates two nouns. The VBA code can have logic that checks for this constraint.

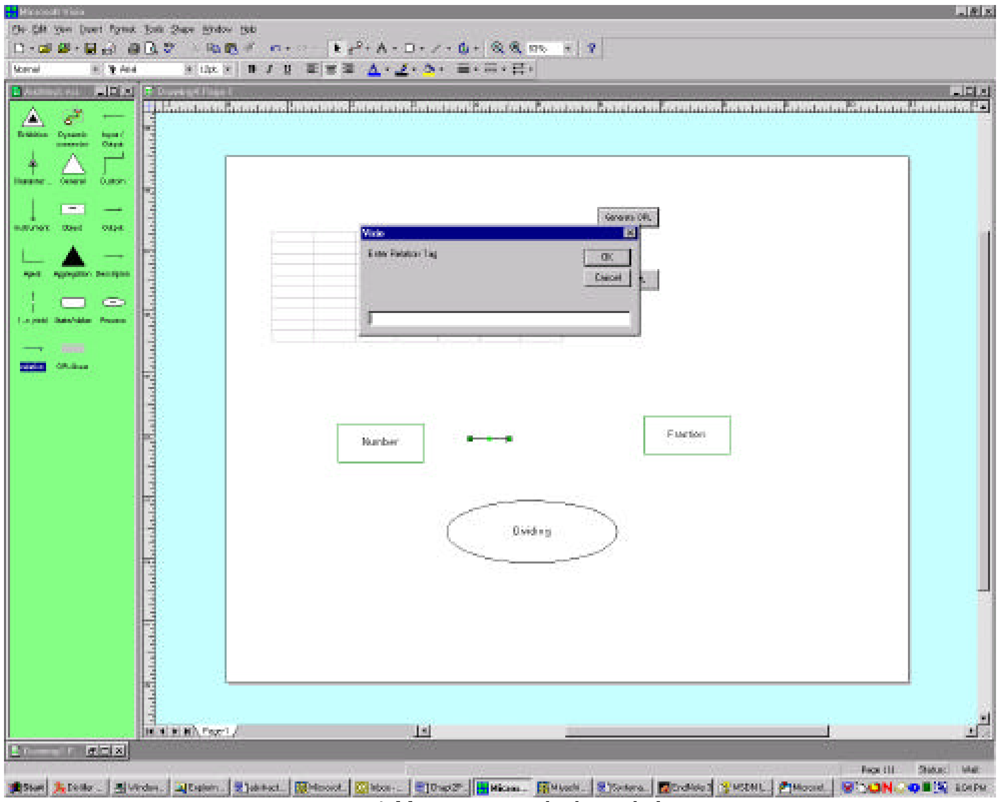


Figure 22 – Adding a structural relation link or connector as it is called in VISIO®.

The next figure shows several structural relations with the corresponding OPL that was generated by pushing the generate OPL button. The vision for the final product is to have the OPL produced as the links are made. VISIO® and VBA are capable of this since all mouse clicks can be captured as events. Once a click is made, the shape it is related to can be examined to see if OPL can be generated.

Another interesting note in this figure is the routing of the links or connectors. VISIO® automatically creates right-angle connectors but these connectors can be made into arcs or straight-diagonal lines, depending on how they are programmed. Note also how the routing of the "works at" structural relation shows jumps where is crosses over the other connectors. This behavior again can be programmed in VISIO®.

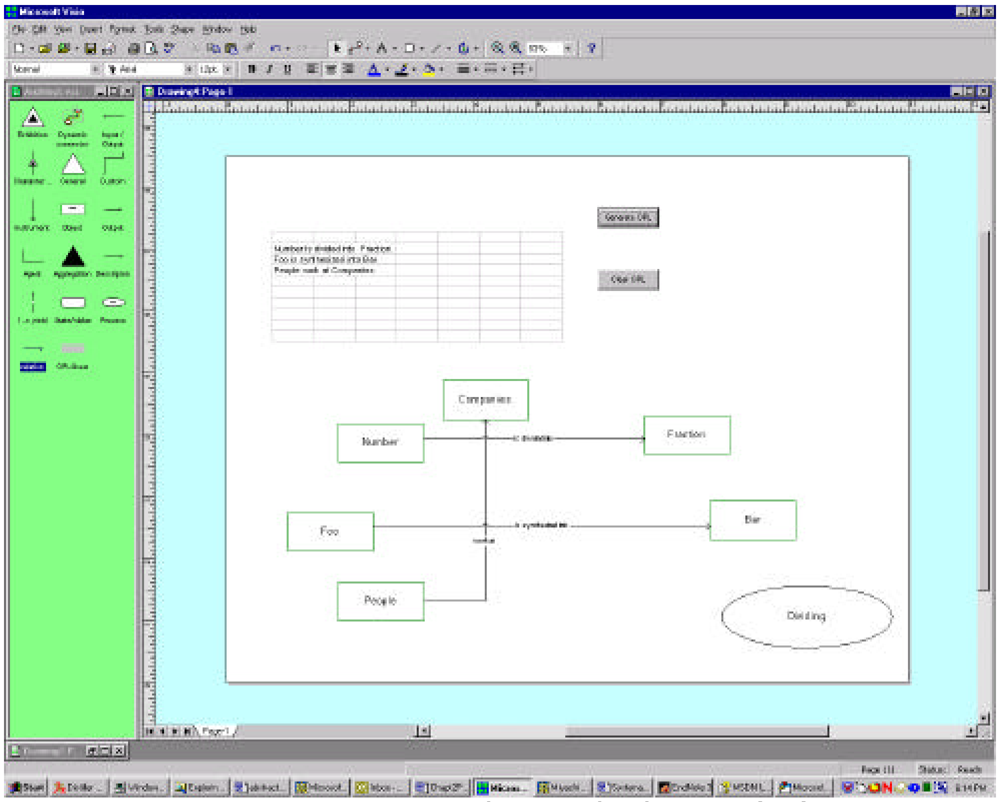

Number is divided into Fraction
Foo is symbolized into Bar
People work at Companies

Figure 23 – Several structural relations with their corresponding OPL

There are rules for the OPL to be highlighted. All keywords should not be in bold and everything else should be in bold. Although this is not shown, VISIO® is capable of doing this. The OPL section of the diagram is an EXCEL spreadsheet. The text in the spreadsheet can be selected and stylized programmatically.

Another thing that was programmed into this tool was that the relation connector was not allowed to connect anything other then objects. For example, the relation connector could not connect a process to an object.

The VBA code is shown in Appendix B.

Collaborative Development

Using VISIO® on a client computer is great for an individual architect but the diagrams must be able to be shared and modified by larges groups of people for product design and development.

Microsoft provides this solution with BizTalk Server. BizTalk Server 2000 will make it fundamentally easier to orchestrate the next generation of Internet-based business solutions. BizTalk Server 2000 will unite, in a single product, enterprise application integration (EAI), business-to-business integration and the much-anticipated BizTalk Orchestration technology to allow developers, IT professionals and business analysts to easily build dynamic business processes that span applications, platforms and businesses over the Internet. [2]

"The technology challenge of the next five years is to realize the value of the Internet to connect and orchestrate companies and applications that today are lone islands," said Chris Atkinson, vice president of the .NET Solutions Group at Microsoft. The OPM tool we describe here is not a lone island and will not be from the design.

BizTalk Server 2000 offers a broad set of tools and an infrastructure to simplify and speed the orchestration of applications and businesses together into next-generation solutions [2]:

- Rapid development of the OPM tool. BizTalk Orchestration technology builds on the Visio® diagramming platform to provide a familiar graphical environment for quickly building dynamic, distributed business processes, and an advanced orchestration engine for executing and managing those processes.

- Easy application and business integration. BizTalk Server 2000 provides tools to easily integrate applications and businesses using industry standard XML.

- Interoperability with industry standards. BizTalk Server 2000 supports other transports and protocols in addition to XML, including EDI (X12 and UN EDIFACT), HTTP, HTTPS, Microsoft Message Queue Server (MSMQ), SMTP (e-mail), and flat file transfer.

- Reliable document delivery over the Internet. Support for the BizTalk Framework ensures reliable, "guaranteed once only" delivery of business documents, such as purchase orders or insurance claims, over the Internet.

- Secure document exchange. BizTalk Server 2000 supports industry-standard security technologies such as public key encryption and digital signatures to ensure secure document exchange with trading partners. This is important for OPM users who want to secure company secrets and only allow for certain people to modify the architecture.

- XLANG support. XLANG is an XML-based language for describing business processes. XML is the industry standard for web development and the OPM tool wants to build upon standards for leverage and to be accepted.

The Learner

The design of the learner was detailed in the previous chapter. However, VISIO® can easily build learner applications. The data can be saved and shapes can be created into templates and stencils for users.

Moreover, OPL can be parsed and diagrams can be created. VISIO® has powerful automatic layout tools, and preferences can be set for specific applications. Again, this could take years of programming but we can build upon the solution VISIO® already has.

The OPL can be parsed using standard compiler technology. The sentences, although natural English language, have rules that could be used for a compiler. This is similar to what we did with the PERL script in the previous chapter.

Summary

This chapter detailed the prototype for an OPM tool built on top of VISIO® . VISIO® was shown to be a flexible tool that we can gain a tremendous advantage using – we can build upon years of work programming in flexible and programmable details, which we can customize for our own use. Moreover, Microsoft is committed to VISIO® as part of their overall Internet development strategy, which will allow our tool to work over the network.

Bibliography

1. *Office Developer Documentation* - VISIO, , MSDN Library.

2. *Microsoft Announces the Availability of BizTalk Serer 2000*, . 2000, MSDN.

3. Dori, D., *Object Process Methodology: A Holistic Systems Development Paradigm*. 2001: Sringer Verlag.

Chapter 5

SUMMARY AND CONCLUSIONS

<u>Chapter 5</u>: Summary and Conclusions

This work has applied OPM to analyze two software development processes. The software development processes examined, in particular CMM, was shown to be expensive and resource heavy, especially for managers. The data presented in Chapter 1 confirmed what the OPM modeling exposed.

The OPM modeling also confirmed the difference in complexity among the CMM Maturity levels. This is confirmed in Fayad et. al. [2]. The levels are uneven in terms of cost to achieve and value.

According to Fayad [1], the data supports the claim that software development processes are necessary, but there are a number of problems in implementation. Processes are often viewed as bureaucracy that decreases the productivity of the development effort and this is often true. But why is this so, when the data for large organizations especially shows that the opposite is true?

One of the problems lies in the cost and value of assessment and certification [2]. Most small and medium size organizations cannot justify the cost. We believe a way to improve this situation is to automate both the implementation and assessment part of a software development process. We also believe that OPM can be used to automate both the implementation and assessment.

According to Wang et. al. [3], problems in software engineering processes can be traced to three root causes:

1. Lack of a formal description: there are many other software engineering processes other than the two we described in chapter 1. The way these processes are described varies considerably and causes confusion among practitioners and process designers.

2. Chaotic interrelationships: the differences between the various processes show the immaturity of the process discipline. The difference in orientation between USDP and CMM show this. USDP shows the process at a lower level of implementation while leaving the higher level more open, and CMM shows the higher level process while leaving the lower level implementation open.

3. Deficiency in validation: validation varies between processes. Some focus on capability (CMM), others lifecycle (USDP) and others on different topic areas. Perhaps all three – organization, development and management subsystems should be looked at in the highest level. On the flip side, benchmarks are needed for the lower levels of the process. These are currently lacking in most if not all software engineering processes [3]

We believe OPM would substantially improve the software engineering process and make it practical for small and medium sized organizations. In the case of lack of a formal description, OPM is the tool to use to describe a variety of systems needed for the architecture of a software engineering process – organization, development, and management systems. OPM could also solve the second problem, because it can be used to describe all processes and the differences can become clearer using the same system tool. This is what we did in Chapter 1.

Finally the validation processes could be described and automated in our future vision of OPM. OPM will not just be a graphical tool, but a complete development and simulation environment that can run and store benchmarks for a system. With this type of automation, the validation cost should be substantially reduced.

References

1. Fayad, M.E., *Software Development Process: A Necessary Evil.* Communications of the ACM, 1997. **40**(9): p. 101-103.

2. Fayad, M.E., M.L., *Process Assessment Considered Wasteful.* Communications of the ACM, 1997. **40**(11): p. 125-128.

3. Yingxu Wang, G.K., *Software Engineering Processes.* 2000: CRC Press.

A p p e n d i x   A

PERL CODE

Appendix A: PERL Code

```
#
#       Perl program to develop OPM
#
#       Author          Date            Comment
#       Chris Miyachi   8/6/00          Created
#
#
# process options
#
if ((scalar @ARGV) != 2) {
        $opd_file = "opd.txt";
        $opl_file = "opl.txt";
}
else {
        $opd_file = $ARGV[0];
        $opl_file = $ARGV[1];
}

#print "Opening $opd_file $opl_file\n";


###################
#  Learner
#####################
#
#  create hash table which links OPL to OPD
#
%opm = ReadOPDFile($opd_file);

foreach $opd_item (keys %opm) {
        #print "$opd_item\n";
        foreach $keyword ($opm{$opd_item}->{keyword}) {
                #print $keyword,"\n";
        }
}

#####################
# Recognizer
#####################
```

```
#
# read in the user's OPL
#
@opl = ReadOPLFile($opl_file);

#
# process opl file
#
@opd = ProcessOPLFile(@opl);

#
# print results
#

#+++++++++++++++++++++++++++++++
# Subroutine ReadOPDFile
#+++++++++++++++++++++++++++++
#
#  read an OPD defintion file and
# put results into a hash table.
#
sub ProcessOPLFile {
        my @lopl = @_;
        my @lopd;

        # go through the OPL paragraph
        foreach $opl_sentence  (@lopl) {
                # print "$opl_sentence\n";
                $opl_sentence =~ s/\.//g;
                # remove white space
                @words = split /\s/,$opl_sentence;
                #
                # look for objects and processes
                #
                @processed = ();
                @keyword_items = ();
                foreach $item (@words) {
                        # look for capital letters
                        if ($item =~ /^[A-Z]/) {
                                if ($item =~ /ing$/) {
                                        $opd_result = FindKeyWord('()');
                                        #print "(): $item $opd_result.\n";
                                        push(@processed,$item);
                                }
                                else {
                                        $opd_result = FindKeyWord('[]');
                                        #print "[]: $item $opd_result.\n";
                                        push(@processed,$item);
                                } # if
                        } # if
                        # we've got possible keywords
                        else {
                                push(@keyword_items, $item);
                        }
                } # foreach
                # make the keywords a phrase
                $keyword_phrase = join(" ",@keyword_items);
                #
                # search the opm for the keywords
                #
                $original_phrase = $keyword_phrase;

                $opd_result = FindKeyWordOPD($keyword_phrase);
                #
                # print OPD
                #
                if ($opd_result eq " ") {
                        print "$processed[0] $keyword_phrase $processed[1]\n";
                }
                else {
                        print "$processed[0] $opd_result $processed[1]\n";
                        $keyword_phrase = "";
                }
        } # foreach
        #
        # otherwise item may be a keyword
        #





        return @lopd;
}

#+++++++++++++++++++++++++++++++
# Subroutine FindKeyWord
#+++++++++++++++++++++++++++++
#
# returns the keyword for a particular OPD from the OPM
#

sub FindKeyWord {

        my $opd_search = $_[0];
        foreach $opd_item (keys %opm) {
                # find a match with the opd
                if ($opd_item eq $opd_search) {
                        # return the keyword
                        foreach $keyword ($opm{$opd_item}->{keyword}) {
                                return $keyword;
                        }
                }
        }
}

#+++++++++++++++++++++++++++++++
# Subroutine FindKeyWordOPD
#+++++++++++++++++++++++++++++
#
```

```
# returns the OPD for a particular keyword/keyphrase
#

sub FindKeyWordOPD {

        my $key_search = $_[0];
        #print "+++++ $key_search\n";
        foreach $opd_item (keys %opm) {
                # find a match with the opd
                #print "+ $opm{$opd_item}->{keyword}\n";
                if ($opm{$opd_item}->{keyword} eq $key_search) {
                        return $opd_item;
                }
        }
        return " ";
}
#++++++++++++++++++++++++++++++
# Subroutine ReadOPDFile
#++++++++++++++++++++++++++++
#
#  read an OPD defintion file and
# put results into a hash table.
#
sub ReadOPDFile {

        %opm_hoh = (
        # simple obect sentence
        '[]'=>
        {
                right => undef,
                keyword => 'is an object',
                left => undef,
                associated => 'simple',
        },
        # simple process sentence
        '()'=>
        {
                right => undef,
                keyword => 'is a process',
                left => undef,
                associated => 'simple',
        },
        # value sentence - must be inside an object
        '{}'=>
        {
                right => '[]',
                keyword => 'can be ',
                left => undef,
                associated => '[]',
        },
        # structure sentence
        '->'=>
        {
                right => ['[]','{}','()'],
                keyword => ' ',
                left => ['[]','{}','()'],
                associated => undef,
        },
        # condition sentence
        '=>'=>
        {
                right => '()',
                keyword => 'occurs if',
                left => '{}',
                associated => undef,
        },
        # resolution sentence
        '+>'=>
        {
                right => '()',
                keyword => 'determines if',
                left => '{}',
                associated => undef,
        },
        # object generation sentence
        '__>'=>
        {
                right => '()',
                keyword => 'yields',
                left => '[]',
                associated => undef,
        },
        # agent sentence
        '__o'=>
        {
                right => '[]',
                keyword => 'handles',
                left => '()',
                associated => undef,
        },
        # Featuring sentence
        '-<+>'=>
        {
                right => '()',
                keyword => 'exhibits',
                left => '[]',
                associated => undef,
        },
        # featuring sentence
        '-<>'=>
        {
                right => '()',
                keyword => 'exhibits',
                left => '[]',
                associated => undef,
        },
        # specialization sentence
        '-<>'=>
        {
                right => '()',
                keyword => 'is a',
                left => '[]',
                associated => undef,
        },
        );

        return %opm_hoh;
}
#+++++++++++++++++++++++++++++++
# Subroutine ReadOPLFile
#+++++++++++++++++++++++++++++
#
#  read an OPL defintion file and
# put results into a hash table.
#
sub ReadOPLFile {

        @opl_list = (
        "Chris is an object.",
        "Hiroshi is an object.",
        "Chris is married to Hiroshi.",
        "Childbearing is a process.",
        "Chris can be pregnant.",
        "Chris is a Woman.",
        "Hiroshi is a Man.",
        "Marrying yields Children.",
        );

        return @opl_list;
}
```

A p p e n d i x   B

VISUAL BASIC FOR APPLICATIONS CODE

Appendix B: Visual Basic For Applications Code

```
Private Sub CommandButton1_Click()
    Call ClearExcelEmbeddedObject
End Sub

Private Sub GenerateOPL_Click()
    Call SendOPLDataToExcel
End Sub
Private Sub ClearOPL_Click()
    Call ClearExcelEmbeddedObject
End Sub


Private Sub Document_ShapeAdded(ByVal Shape As IVShape)
    Dim shapName As String  'name user gives to shape
    Dim mastObj As Master   'master object
    Dim celObj As Visio.Cell 'cell object
    Dim OPLString As String ' OPL associated with shape
    Dim shapTypeString As String 'Name of OPD thing
    Dim consObj As Visio.Connects 'a group of connect objects
    Dim conObj As Connect       ' a single connect object
    Dim curConnIndx As Integer  ' index for connects object
    Dim shpObj As Shape         ' Visio Shape

    curConnIndx = 1
    ' get the type of OPD object this is being place
    If Shape.CellExists("User.OPDType", 0) Then
        Set celObj = Shape.Cells("User.OPDType")
        shapTypeString = celObj.ResultStr("")
    End If
    'Determine if it is a shape or a connector
    If ((shapTypeString = "process") Or (shapTypeString = "object") Or
(shapTypeString = "value")) Then
        shapName = InputBox("Enter Shape Name")
        Shape.Text = shapName
        If Shape.CellExists("User.OPL", 0) Then
            Set celObj = Shape.Cells("User.OPL")
            OPLString = celObj.ResultStr("")
            Debug.Print shapName & OPLString
        End If
    ' the relation connector
    ElseIf (shapTypeString = "relation") Then
        shapName = InputBox("Enter Relation Tag")
        Shape.Text = shapName
        Set consObj = Shape.Connects
        ' check to see the the connector only connects objects
        For Each conObj In consObj
            ' get the current connect object from the collection
            Set conObj = consObj(curConnIndx)
            shpObj = conObj.ToSheet
            If shpObj.CellExists("User.OPDType", 0) Then
                Set celObj = shpObj.Cells("User.OPDType")
                shapTypeString = celObj.ResultStr("")
                If Not (shapTypeString = "object") Then
                    MsgBox "Can only connect to objects!", vbOKOnly, "Invalid
Connection"
                    Exit For
                End If
            End If
            curConnIndx = curConnIndx + 1
        Next
    End If
End Sub

Sub Document_Shape()
    Dim pagObj As Visio.Page    'Visio Page Object
    Dim shpsObj As Visio.Shapes 'Visio Shapes collection
    Dim shpObj As Visio.Shape   'Visio Shape object
    Dim celObj As Visio.Cell    'Visio Cell object
    Dim ShapeName As String     'Array to hold Names
    Dim OPLString As String
    Dim iShapeCount As Integer  'Counter
    Dim i As Integer            'Counter
    Dim mastObj As Master       'Master of shape

    Set pagObj = ActivePage
    Set shpsObj = pagObj.Shapes
    iShapeCount = shpsObj.Count


    For i = 1 To iShapeCount
        Set shpObj = shpsObj(i)
        Set mastObj = shpObj.Master
        ShapeName = shpObj.Text
        'Debug.Print ShapeName
        Debug.Print "Master " & mastObj.Name
        If shpObj.CellExists("Prop.OPL", visExistsLocally) Then
            Set celObj = shpObj.Cells("Prop.OPL")
            OPLString = celObj.ResultStr("")
            'Debug.Print ShapeName & " " & OPLString & "."
        End If
    Next
End Sub

Sub Relation_OPL()
```

```
Dim shapName As String  'name user gives to shape
Dim celObj As Visio.Cell 'cell object
Dim OPLString As String ' OPL associated with shape
Dim shapTypeString As String 'Name of OPD thing
Dim consObj As Visio.Connects 'a group of connect objects
Dim conObj As Connect       ' a single connect object
Dim curConnIndx As Integer  ' index for connects object
Dim shpObj As Shape
Dim shpObjFrom As Shape         ' Visio ShapeSet consObj = Shape.Connects
Dim pagObj As Visio.Page    'Visio Page Object
Dim shpsObj As Visio.Shapes 'Visio Shapes collection
Dim ShapeName As String     'Array to hold Names
Dim iShapeCount As Integer  'Counter
Dim i As Integer            'Counter

Set pagObj = ActivePage
Set shpsObj = pagObj.Shapes
iShapeCount = shpsObj.Count


Dim IsRelation As Boolean
' go through all the shapes on the page
For i = 1 To iShapeCount
    Set shpObj = shpsObj(i)
    Set consObj = shpObj.Connects
    ' get the type of OPD object this is being place
    If shpObj.CellExists("User.OPDType", 0) Then
        Set celObj = shpObj.Cells("User.OPDType")
        shapTypeString = celObj.ResultStr("")
    End If
    If shapTypeString = "relation" Then
        Dim RelString As String
        RelString = shpObj.Text
        Dim shpFromName As String
        Dim shpToName As String
        curConnIndx = 1
        ' check to see the the connector only connects objects
        For Each conObj In consObj
            ' get the current connect object from the collection
            Set conObj = consObj(curConnIndx)
            Set shpObj = conObj.ToSheet
            ' if relation is not connected to the rigth objects...
            GetShapeType shpObj, IsRelation, shapTypeString
            If IsRelation = False Then
                Exit For
            End If
            ' each connection only has a to and a from
            If (curConnIndx = 1) Then
                shpFromName = shpObj.Text
            Else
                shpToName = shpObj.Text
            End If
            curConnIndx = curConnIndx + 1
        Next ' For each...
        ' print out the OPL sentence
```

```
            If (IsRelation = True) Then
                Debug.Print shpFromName & " " & RelString & " " & shpToName & "."
            End If
        End If
    Next ' For i = ...
End Sub

' Determine the Type and if it can be connected to a relation.
Private Sub GetShapeType(shpObj As Visio.Shape, IsRelation As Boolean, shapTypeString
As String)
    IsRelation = True
    If shpObj.CellExists("User.OPDType", 0) Then
        Set celObj = shpObj.Cells("User.OPDType")
        shapTypeString = celObj.ResultStr("")
        If Not (shapTypeString = "object") Then
            MsgBox "Can only connect to objects!", vbOKOnly, "Invalid Connection"
            IsRelation = False
        End If
    End If
End Sub


Sub ClearExcelEmbeddedObject()

    'This sub clears the embedded Excel spreadsheet.
    Dim xlSheet As Excel.Worksheet

    Set xlSheet = ActivePage.Shapes("OPLSheet").Object.Worksheets(1)

    'Select a very large range, and clear the contents:
    xlSheet.Range("A2:E500").ClearContents

    'Let's do some formatting as well:
    'Whole sheet, left-aligned
    xlSheet.Range("A1:D500").HorizontalAlignment = xlLeft

    'Last column's data, right-aligned:
    xlSheet.Range("D1:D500").HorizontalAlignment = xlRight

    'Clear all Bold:
    xlSheet.Range("A2:D500").Font.Bold = False

End Sub

Sub SendOPLDataToExcel()
   'This sub exports drawing data to the embedded Excel spreadsheet.

    Dim xlSheet As Excel.Worksheet
    Dim iXLRowNum As Integer, i As Integer
    Dim shp As Visio.Shape

    'First, clear out the existing spreadsheet, in case there's more
    'stuff there than we'll end up with:
    Call ClearExcelEmbeddedObject
```

```
Set xlSheet = ActivePage.Shapes("OPLSheet").Object.Worksheets(1)
iXLRowNum = 2

   Dim shapName As String  'name user gives to shape
Dim celObj As Visio.Cell 'cell object
Dim OPLString As String ' OPL associated with shape
Dim shapTypeString As String 'Name of OPD thing
Dim consObj As Visio.Connects 'a group of connect objects
Dim conObj As Connect       ' a single connect object
Dim curConnIndx As Integer  ' index for connects object
Dim shpObj As Shape
Dim shpObjFrom As Shape         ' Visio ShapeSet consObj = Shape.Connects
Dim pagObj As Visio.Page    'Visio Page Object
Dim shpsObj As Visio.Shapes 'Visio Shapes collection
Dim ShapeName As String     'Array to hold Names
Dim iShapeCount As Integer  'Counter

Set pagObj = ActivePage
Set shpsObj = pagObj.Shapes
iShapeCount = shpsObj.Count


Dim IsRelation As Boolean
' go through all the shapes on the page
For i = 1 To iShapeCount
    Set shpObj = shpsObj(i)
    Set consObj = shpObj.Connects
    ' get the type of OPD object this is being place
    If shpObj.CellExists("User.OPDType", 0) Then
        Set celObj = shpObj.Cells("User.OPDType")
        shapTypeString = celObj.ResultStr("")
    End If
    If shapTypeString = "relation" Then
        Dim RelString As String
        RelString = shpObj.Text
        Dim shpFromName As String
        Dim shpToName As String
        curConnIndx = 1
        ' check to see the the connector only connects objects
        For Each conObj In consObj
            ' get the current connect object from the collection
            Set conObj = consObj(curConnIndx)
            Set shpObj = conObj.ToSheet
            ' if relation is not connected to the rigth objects...
            GetShapeType shpObj, IsRelation, shapTypeString
            If IsRelation = False Then
                Exit For
            End If
            ' each connection only has a to and a from
            If (curConnIndx = 1) Then
                shpFromName = shpObj.Text
            Else
                shpToName = shpObj.Text
            End If
            curConnIndx = curConnIndx + 1
            Next ' For each...
            ' print out the OPL sentence
            If (IsRelation = True) Then
                Debug.Print shpFromName & " " & RelString & " " & shpToName & "."
                'Add data to spreadsheet.
                xlSheet.Range("A" & LTrim(Str(iXLRowNum))).Formula = shpFromName & "
" & RelString & " " & shpToName & "."
                'Increment Excel row-tracking variable:
                iXLRowNum = iXLRowNum + 1
            End If
        End If
    Next ' For i = ...
End Sub



   ' Determine the Type and if it can be connected to a relation.
Private Sub GetShapeType(shpObj As Visio.Shape, IsRelation As Boolean, shapTypeString
As String)
    IsRelation = True
    If shpObj.CellExists("User.OPDType", 0) Then
        Set celObj = shpObj.Cells("User.OPDType")
        shapTypeString = celObj.ResultStr("")
        If Not (shapTypeString = "object") Then
            MsgBox "Can only connect to objects!", vbOKOnly, "Invalid Connection"
            IsRelation = False
        End If
    End If
End Sub
```