MIT/LCS/TR-334

AN APPROACH TO FUNCTIONAL OFFICE AUTOMATION

Craig L. Zarmer

This blank page was inserted to preserve pagination.

An Approach to Functional Office Automation

by

Craig Zarmer

June 1984

Massachusetts Institute of Technology
Laboratory for Computer Science
Cambridge, Massachusetts 02139

An Approach to Functional Office Automation

by

Craig L. Zarmer

Submitted to the
Department of Electrical Engineering and Computer Science
on April 30, 1984, in partial fulfillment of the requirements
for the Degree of Master of Science

Abstract

Current efforts in office automation emphasise developing tools for supporting common, low-level tasks such as word processing and electronic mail. While they have a wide market, they are not very sophisticated. At the other end of the spectrum are office-specific systems, designed with complete knowledge of the office's operations. Unfortunately, such systems have a market size of one, and so are not very practical.

Earlier work has suggested that all offices are not completely different. The premise of this thesis is that there are many offices that, while on the surface appear dissimilar, actually perform the same *function*. This suggests that a compromise between simple tools and office-specific systems may be possible by supporting these functions. The purpose of this thesis is to explore the practicality of building software systems that support office functions.

A study of project management at a software firm was completed, and it was used as the basis for evaluating the functional approach. A description of the project management offices, and how a project management support system could work is presented. While such a limited study is not sufficient to verify the premise of commonality, it showed that much could be done in support of project management beyond the currently available tools, and that individual differences in style and preferences do not alter the core functionality needed.

A methodology for implementing such systems was also developed. It shows how the abstract objects discovered through office analysis can be related directly to objects in an object-oriented programming language. An electronic desktop architecture is described that manipulates such software objects, and a toolkit is presented that makes it easier to build or modify a support system.

Key Words: Office Automation, Project Management

Thesis Supervisor: Irene Greif
Title: Principal Research Scientist

Acknowledgments

I'll begin by acknowledging my thesis advisor, Irene Greif. She has been supportive and helpful from the beginning, and her many insightful comments have greatly added to this thesis. It is not always easy to do research across the country from your advisor, but she has made it as easy as being down the hall.

I am deeply indebted to many, many people at Hewlett-Packard. Ray McCarthy gave me the freedom to pursue this research, and was supportive from the beginning. He and the other managers who participated in the case study each gave up many hours of their time. Their patience and understanding went well beyond the call of duty.

At HP Laboratories, Ira Goldstein gave me the time and equipment that enabled me to finish the research. I sought help from nearly every member of my lab at least once, and quite a bit more often from some. John Wynbeek, my supervisor at HP Labs, was particularly helpful. He contributed several ideas to this thesis, and in reviewing a draft proved himself a master editor.

My family and friends encouraged me to pursue and finish this work, and kept me sane while I did. I'd particularly like to thank the west coast division of New House II for their support and friendship. I can never thank my parents enough for the love and guidance they have given me. Most of all, I'd like to thank Kim Anderson for, well, everything. This thesis is dedicated to her.

Table of Contents

Chapter 1
Introduction

1.1 The Problem

Man has been inventing tools to do his job more efficiently since the beginning of time. At least in the office, each time a newer technology becomes available, we tend to use it to make our previous tools more powerful. Seldom do we start with a clean slate and see how a new technology can be put to undiscovered uses. The manual typewriter was first introduced in the early 1900's. It was followed by the electric typewriter and the correcting electric typewriter. Today, we use computers in offices to do word processing.

While there is nothing wrong with applying the latest technology to enhance existing tools, the tool building approach does not tap the full potential that modern computers have to offer. Indeed, several office automation (OA) vendors have taken the next step: joining several tools together under a single umbrella and allowing them to exchange information. These combinations do not add any new functionality, they are simply a bit easier to use.

What could be better? The yardstick used by Hammer and Sirbu seems reasonable: an office specific system [Hamm80.] That is, an office system designed from the ground up to assist an office worker in whatever way it could. By this yardstick, a collection of tools, even if packaged together so they can share data, falls considerably short.

This thesis is not about building office-specific systems. Elaboration in chapter two will show that an office-specific system cannot be a mass-market product. Office-

specific systems are not the answer, neither for the customer who cannot afford one or the OA vendor who cannot profitably produce one. They are, however, a goal worth striving for. The problem is, then, to find a compromise between generic task-oriented tools (*generic tools* hereafter) and an office-specific system that can be mass-marketed.

1.2 A Possible Solution

What lies between tools and office-specific systems? After all, no two offices are alike. Even a sampling of, say, personnel offices, will be different in *some* way. If nothing else, the people working in those offices will be different, and an office specific system has to account for personal preferences.

One possible compromise is to design OA systems that support a particular *business function.* Previous research has suggested that despite all of the surface-level differences between offices, there are relatively few functions that singly or in combination form the core of most office work. For example, consider two hypothetical personnel offices. Office A is in a young firm with a total employment of 100, a personnel staff of two, and not a computer in sight. Office B is in a large car manufacturer, thousands of employees, a personnel office staff of one hundred, and everything computerized. At first glance, these offices may seem miles apart. When the implementation details are cleared away, though, and basic functions of the offices are compared, they could be very similar. Business functions are the subject of chapter three.

Consider what it would mean to OA vendors if there really were a rather small number of business functions in the world. They could devote resources towards finding them and understanding how best to support them. They would have a product that would be much more sophisticated than the generic tools currently available. At the same time, they would have something that could be mass-marketed.

1.3 Purpose of this thesis

Despite the amount of work done thus far in studying offices, orders of magnitude more would have to be done to show that all offices could be described in terms a relatively small number of business functions. Just eliciting the taxonomy of business functions is a major undertaking; perhaps a definitive list does not even exist. However, it is not necessary for *every* office to be describable as a business function for the approach to be commercially feasible, just *most* of them. Even with this relaxation, determining what the business functions are and how they should be supported is a substantial job.

Given the size of the entire task, it is desirable to get some early test results from a very small subset of all possible offices. Suppose that we do have the definitive taxonomy of office functions. Can we pick one and implement an OA system that supports it better than a simple collection of tools? Is there even one real office in the world that fits that function well enough to make use of it's associated support system? How do you go about building such a system?

The purpose of this thesis is twofold. First, without considering implementation, it develops and tests the idea of functional support systems in a particular case. Second, it tests the feasibility of implementing such a system by developing a methodology and implementation scheme. The particular business function chosen for this thesis was *project management.*

Five different managerial offices in the research and development labs of a computer software division of Hewlett-Packard Company were studied. The five offices were at different levels in the organization, and the managers of those offices varied considerably in background and style. The study methodology is explained briefly in chapter three. Chapter four presents the highlights of the study and how a project management system could support the offices. A detailed description of the offices is in an appendix.

Because the offices studied were complex, some of the ways suggested in chapter four for supporting project management could be very difficult to implement.

For the functional office automation approach to succeed, a methodology is needed to guide designers from office study to software implementation. Chapter five presents such a methodology. The methodology uses an implementation model that follows very closely the study and analysis model: both are *object-oriented*. An "electronic desktop" was implemented using an object-oriented dialect of Lisp, and a few of the objects found in the office were modeled in software using the desktop. Chapter five describes the architecture of the electronic desktop.

Because no two offices are identical, a successful OA system must be tailorable to the specific needs and preferences of each office in which it is installed. Chapter five concludes by describing how this can be accomplished at two levels. The smallest changes, reflecting particular personal preferences by a user, can be easily done by that user. For more substantial changes, a "functional OA toolkit" is outlined that could be used by more technically trained staff to supplement or modify the standard functional support system. This toolkit could also be used by OA vendors to construct new systems. Finally, chapter six summarizes the work presented in this thesis.

Chapter 2
Office Automation: Goals and Current Practices

2.1 The Ideal Office System

What is the goal of an OA system? Previous work done by MIT's Office Automation Group has proposed a very straightforward goal: to improve the realization of office functions, as measured in business terms [Hamm80]. Typically, though, office productivity is measured at the task level, such as the number of pages a secretary produces per day. The task method would consider any increase in the number of pages per day an increase in office productivity. Following this logic, purchasing a word processor for a secretary will allow more pages to be produced per day, or that secretary can do other tasks. Either way, the word processor will improve office productiviy.

Often, the purchase of a word processor based on a task analysis will produce little or no improvement in the office's overall productivity. Hammer and Zissman describe the typical consequences of installing OA equipment without considering the entire office's function and purpose [Hamm79]:

- Installing new technology where no problem existed.
- Parkinson's law, version 1: workers slow down due to the increased speed of the equipment.
- Parkinson's law, version 2: managers unnecessairly demand more work to fill the capacity of the new equipment. For example, doing 10 revisions of a memo when two had previously suficed.

- Having no idea whether the new equipment has really improved productivity, since no clear measures were ever developed or considered.

Every office has a *mission*, a function to carry out, a reason for existing in its organization [Hamm80]. Those things that improves an office's ability to fulfill its mission is trully automating the office. Word processing may be part of the OA system some office needs, but only after study shows that improving the quality or quantity of documents will actually help the office carry out its mission better.

There are undoubtedly a large number of offices where the currently available tools, such as word processing and electronic mail, will significantly improve productivity. Simply using these tools will not, however, achieve the best possible improvement. If no two offices are alike, and if OA efforts are judged by their effect on the whole office, then how can mass-marketed generic tools possibly represent the best that can be done for an office? From this argument it follows that *ideal office systems are office-specific.*

That ideal office systems are custom built for each office should be intuitive. How better to support an office then to build a system from the ground up with an office and even specific people in mind? Of course, there are very good reasons why every office does not have its very own OA system:

- It would take too much time to create.
- It would cost too much money.
- If it was ever delivered, it would only be optimal for a day or two before something changed.

Still, it is useful to consider some of the qualities an office-specific system would have, as a goal to strive for. The strategy is to automate the highly structured aspects of the work, and to provide whatever support is possible for the less structured aspects [Kuni82]. Structured activities follow some algorithim, although that algorithim may

be very complex. Unstructured activities, on the other hand, require some amount of human decision-making.

An office-specific system is built with knowledge of the entire office. Studying and supporting each worker individually is not enough; the office must be studied as a whole. Aspects of an office's work may have been considered unsupportable simply because they were done by separate people, or involved separate databases, or were processed by separate programs. When the entire office is considered, suddenly these aspects become supportable or even automatable.

An office-specific system understands the context of each task. It can supply defaults, suggest alternatives, and in general be more helpful than a generic tool that has no context at all. These distinctions can all be brought out better by comparing an office-specific system and some generic tools in some specific examples. First, we need to look more carefully at tools and their inherent limitations.

2.2 Current Practice In OA

2.2.1 Tools

In contrast with office-specific systems, most products offered by today's OA industry are tools. They are designed to improve the efficiency of an office employee, usually a clerk or secretary, in the performance of some common task. They are, essentially, electronic analogs of the conventional tools used to perform office tasks [Kuni82]. Word processing is a prime example. Most offices have to produce documents; often they can be produced faster on a word processor than on a conventional typewriter. Electronic mail, electronic filing, and electronic spreadsheet programs are other common examples.

In seeking out new products, the OA vendors have looked for the least common denominators of office work. This approach has naturally lead them to the current

variety of generic tools. The advantage to the OA vendor is that the market for a product is maximized and the required sophistication of the product is minimized.

As an example of the current thinking by the OA industry, consider the definition of office automation:

> What is office automation? In office operations, it extends the ability to automate paper-related text- and data-processing tasks. In single-function embodiments, it provides text processing via standalone electronic typewriters or word processors to typists and secretaries. It also provides decision support to managers and executives via personal computers. In more advanced forms, office automation integrates various functions [Murp83].

A few years ago, the word "integration" became very important in the office automation industry. In many cases, integrated office systems were the same old collection of tools packaged together with a series of menus that hid most of the operating system from the user. While making a tool easier to use is always desirable, it is still a tool. The next step in integration allowed programs to share and exchange data. For example, a chart created in one subsystem could be captured and included in a document, an improvement over the usual "cut-and-paste" method of illustrating documents. Again, while it makes the tools easier to use, it does not move them significantly towards the goal of being office-specific.

Not all currently available software is targeted at clerks and secretaries. Tools for the manager, executive, and professional workers are becomming very popular. For example, *management information systems* (MIS) gather information from an organization's data base and present it through predefined reports. MIS are suited to well-structured, recurring decision making [Thie82]. *Decision support systems* (DSS) are generally more interactive, and are tailored toward ad-hoc, less structured problem solving. They often have extensive modeling capabilities, allowing the user to ask "what if" questions.

Some of these management tools are very complex and sophisticated. Nevertheless, they are still tools. They are designed to have as large a market as possible, and so give up the ability to understand the particulars of any one office. They deal with a person's problem at the task level, rather than an entire office's problem. DSS are examples of products that are just starting to move above the level of tools. They are different because they address less structured activities than MIS and other tools. When the modelling and query capabilities, as well as the database itself, are tailored to the needs of a specific office, DSS begins to be a part of the office-specific solution advocated. Use of DSS are, however, only a part of the solution.

2.2.2 The limitation of tools

A good question is, why can't we come reasonably close to an office-specific system by using a carefully chosen set of tools? The answer will point out some of the limitations of a task-oriented approach to OA, and demonstrate some of the qualities of an office-specific system. The main distinction arises from the following principle: the more that is known about an office, the more that can be done to help that office. Tools generally have a very narrow view of the office, and so can only assist the part of office work that they see. An office-specific system, on the other hand, can see the entire office.

For example, consider a hypothetical bill collection office. In most cases, the office prepares a few warning notes, and if payment still is not received, sends out an agent. A few cases are handled differently. The office is judged by the amount of money it collects and the amount it spends trying to collect it.

Following the task-oriented approach, one analyst notes that a large amount of typing is done, and that most of it is repetitive except for the names, addresses, and dates. He recommends installing word processing stations. The system is implemented; the office staff now spends less time typing and more time chasing down those who won't pay.

Now suppose the same office is visited by another analyst who believes in office-specific systems. She notes that the office always sends out two form letters before dispatching an agent, unless the amount due is over $1000. In that case, an agent studies the case and drafts a custom letter. She designs a system that automatically generated the initial letters for the smaller cases, and directed those with large amounts due to the agents, putting the worst cases first. Of course, her solution assumes that the source information is already machine readable.

The goal is to automate the structured and to support the unstructured aspects of an office. The word processing solution, simply improved the efficiency of human performance of a highly structured task. The office-specific system, however, completely automated the structured tasks. In addition, it supported the less structured work of the agents by giving them the worst cases first.

The example above points a more general problem with the task-oriented approach to OA. Supporting an office by supporting its tasks assumes that the tasks were right in the first place. Adding OA support for these tasks sets them in concrete. Hammer and Kunin note that the task approach mistakes the "means for the ends, and artifacts for essentials"[Hamm82]. Understanding what tools are needed, if any, must come after getting a complete understanding of the office.

Another limitation of generic tools is that they force the user to mold his data and his way of doing things to the tool, rather than the other way around. For example, an electronic spreadsheet system presents a rectangular grid to the user which allows him to specify that a location is to be a function of other locations. If a user wants to use this tool, he must coerce his data to the format expected by the spreadsheet system. For example, if he wants to know how many engineers he can hire before needing to buy a new computer, he adds entries to column A and looks to see when column B - column C is less than column D. These are the consequences of designing products to

be the least common denominator of a variety of offices: it is restrictive to all offices and not as helpful to any particular office.

2.3 Summary

Generic tools have a number of disadvantages over office-specific systems. Automating specific tasks misses many of the best opportunities for improving an office's productivity. Task-oriented solutions often emphisize the highly structured tasks that could be eliminated completely if a system took the whole office into account. Tools designed for specific tasks are inherently less able to match the actual needs of an office, neither can they provide as much assistance to the user, as office-specific systems.

Although it has already been admitted that it is not generally practical to build office-specific systems, there should be a beneficial compromise between the two extremes. The compromise proposed in this thesis is based on a previously developed theory of *common office functions*, and is the subject of the next chapter.

Chapter 3
Functional Office Automation

This chapter explains what functional office automation is, and how it can serve as a compromise between office-specific systems and generic tools. The solution has evolved from several years of research in the study, analysis, and specification of offices and office work by the Office Automation group at MIT. The first section presents a very brief overview of their office model. Following sections show how that model can be practically applied.

3.1 Office Analysis And Specification

There are many ways to study an office. The approach taken depends on why the office is being studied. Research at MIT has led to a particular model of office work; their methodology is designed to obtain the information needed to apply their model to an office. In Analysis and Specification of Office Procedures [Kuni82], Kunin presents the full model of offices and office work. It includes an Office Specification Language (OSL) that allows both the structure of an office and its activities to be described formally. Along with OSL is OAM, an Office Analysis Methodology [Sirb83]. Although usable independently, the two share a common model. OAM was subsequently refined by Sutherland into an Office Analysis and Diagnosis Methodology (OADM) [Suth83.] Hereafter, this thesis will refer to the combined models of OAM, OADM, and OSL as simply the OSL model.

The OSL model begins with the premise that each office has some *mission* to fulfill, some reason for its existence in the organization. To accomplish this mission, an office

employs one or more *functions*. A function is a collection of all the activities associated with the management of some central *resource*, or focal object. A function creates new instances of the central object, it manages them, and finally it terminates them in some way. Functions, in turn, are implemented by one or more *procedures*. A procedure deals with one specific object or class of objects, processing it from one state to another. Finally, procedures are made up of *steps*, each representing some basic action such as "select" or "evaluate."

When discussing offices or resources, it is important to distinguish between *physical* entities and *abstract* entities. Often, different offices will have different physical implementations of the same abstract object. *Documents* and *forms* are good examples of this. It is the information on the document or form that is important, not its format. Likewise, the media is not important. For example, one office may use an ordinary filing cabinet while another uses a computer system, but the abstract object they are implementing may be the same.

Organizations may put some of the people working on a function in one office, while others work in an office next door, on the next floor, or in another country. A single person may have several *roles*, or there may be a single role distributed over several people. The analyst must discover the mapping between the abstract office and the physical offices; between roles and people; between abstract objects and physical implementations of them.

The concepts described here are difficult but important. It would be worthwhile to read a fuller description of the study methodology and some example descriptions of offices studied with the methodology. [Sirb83] contains both a paper describing the methodology and several studies of offices.

3.2 Basic Premises Of OSL

The OSL model rests on several premises. The first, the need to focus on the abstract object instead of its implementation, was discussed above. The second is that office work does have structure. If, by and large, office work followed no algorithm, not even a very complex one, it would be impossible to formally describe it. Saying that an entire office has structure is different than saying a particular activity is structured. An office can have structure without every decision or creative thought being algorithmic. In fact, some primitive activities in the OSL model's formal language includes some clearly non-algorithmic activities such as "decide" or "evaluate." The structure is at a higher level, providing an ordering of the activities into procedures and functions. Nor does the premise mean that nothing ever goes wrong. Exception handling, though, usually follows some algorithm of its own.

The third premise is that office procedures are basically simple. The complexity found in a typical office usually arises from one of two sources. A myriad of special forms and a book of rules for each can make an office appear very complex, yet when the forms are examined at for information content, and the processing procedures are elevated above implementation level, their OSL description is often much simpler. Frequently, offices have a number of "special case" procedures added to the "main line" procedures over the years. Taken as a whole, the office appears hopelessly complex, but when organized into a main line and special cases, it becomes much simpler. Physical offices also can seem complex when they are performing several unrelated functions. Once an office becomes responsible for something, it usually owns that task for life. 'Historical reasons" is a common justification for an office performing a task that seems unrelated to the rest of its tasks.

These premises have been borne out in the studies conducted to date. Many more studies will have to be conducted before they can be declared empirically true. Yet,

the arguments and actual experiences thus far are strong enough that they will be used to derive the basic premise underlying this thesis.

3.3 Functional Office Automation

3.3.1 The basic premise

Summarizing the OSL premises from the previous section:

1. Abstract objects are important, not their implementation.
2. There is structure to office work.
3. Office procedures are basically simple.

At first glance, two offices may seem completely different. Their operations may seem so complicated that they each appear to be one-of-a-kind. After sorting their operations out into main procedures and exception handlers, though, similarities may arise. Likewise, when the objects in the offices are viewed abstractly, more similarities may arise. In general, an OSL description of an office is much simpler than an English description of an office. Through simpler descriptions, many offices that were formerly considered unique may appear similar to other offices. Similar offices will likely form groups based on their functions, and this leads to the fundamental premise of this thesis:

> There are many offices that, while on the surface appear dissimilar, actually perform the same *business function.*

It seems quite reasonable to expect the basic structure of a function to appear in many different offices. In particular, the industry to which an office's firm or organization belongs should not have a significant effect on the functions used. Kunin notes in his conclusion that "a more extensive study of the structure of office functions may lead to the definition of a set of 'generic applications' " [Kuni82.]

It is not the purpose of this thesis to prove the premise by generating a taxonomy of business functions; such a task is well beyond its scope. Besides, how would one prove or disprove the general usefulness of the premise? We can not possibly study every office. A better approach is to accept the premise, see what practical guidance in OA design follows from it, and begin testing it. If it proves useful, apply it further, if not, try to discover in what way the premise is wrong.

The term *business function* used in the premise, and throughout this thesis, reminds us of the perspective we must take when thinking about offices and office functions: they must make sense from a business point of view. Discovering the underlying functions in an office is still something of an art, requiring careful consideration by the analyst. Keeping the office's *business* mission in mind will help.

3.3.2 Applying the premise

If offices can be grouped by business functions, then there must be fewer business functions than there are offices. If each business function supported only ten offices, then there would be only one tenth as many business functions as there are offices. Hopefully, a business function will be useful to many more than ten offices, bringing the number of business functions down to a managable number – say on the order of one hundred functions. This goal may seem more reasonable if the "80/20 rule" is used: 80 percent of the offices are constructed from a set of one hundred or so business functions.

If there truly is a relatively small number of business functions to deal with, then OA vendors can invest in understanding them and developing systems to support them. The resulting *functional OA systems* will be marketable, since the theory is that many offices perform the same business function. At the same time, the products will approach meeting the needs of an office as well as an office-specific system would. Functional OA systems represent a good compromise between office-specific systems and generic tools.

3.4 Questions to be answered

Although the theory seems plausible enough, there are a number of questions that could stand in the way of a successful, practical implementation. First of all, can offices be supported any better than present day generic tools can? The previous chapter gave some hypothetical, isolated examples of what a system could do with more knowledge of the office. More evidence is needed, however, to show that use of a functional support system or even an office-specific system leads to a substantial improvement in office productivity. The productivity gain over generic tools must be great enough to justify 1) a vendor to develop it and, 2) a customer to buy it.

Managerial activities present a special challenge to an OA system. There is at least some evidence that OA, even through generic tools, can improve productivity in the clerical and secretarial areas. Managerial activities, on the other hand, are often thought to be unstructured and unsupportable. Authors such as Henry Mintzberg feel that management work mainly uses verbal, face-to-face contact for communication, and managers tend to work with "soft" rather than "hard" information [Mint76.] If an office-specific system does not support clerical work significantly better than generic tools, and if management is not supported at all, then functional OA systems are not worth developing.

Even if specific offices can be substantially assisted by functional systems, the next issue is the commonality of the functions defined and supported for those offices. If two different offices are found to be performing the same business function, but the system designed for one is useless for the other, then we are left writing office-specific systems, which are impractical.

Suppose that research had discovered a number of business functions which occurred in a wide range of offices. To be successful, a functional support system has

to accommodate a number of differences between offices performing the same business function. For example:

- Industry: Perhaps the needs of one industry's personnel office are dramatically different than those in another.
- Corporate practice: Even though the offices have the same mission, differing corporate customs, policies, and traditions may make a system unacceptable in one or the other.
- Personal tastes, habits, and preferences: These attributes are particularly important for the less structured aspects of office work. There are managers who are computer shy and there are those who are "hackers."

One very practical question remains: how can such systems be built? Evidence is needed that the current technology in building large systems is adequate to construct a functional support system. A clear methodology for implementing these systems from a study of the functions is also needed.

This thesis addresses some of these issues. It gives examples of offices that can be supported by an office-specific system substantially better than generic tools. Furthermore, these offices are largely managerial, and examples of how the semi-structured and unstructured activity of a manager can be supported are included. A methodology for realistically designing and implementing a functional support system is presented, and the strategy includes tailoring a support system to an industry, organization, or person. The question of commonality is not, however, tested by this thesis. While the examples used in this thesis tend to support the commonality premise, much more research would have to be done to consider it reliable.

Chapter 4
Project Management at Hewlett-Packard

To begin testing the functional OA approach, a group of offices in a division of Hewlett-Packard Company (HP) were studied in detail. To keep the test as simple as possible, each of the offices chosen uses a single function to fulfill their missions: *project management.* They are all from the research and development labs. Even with these simplifications, the results of the study answer the first issue raised at the end of the previous chapter: offices can be supported substantially better with generic tools, even those with many semi-structured and unstructured tasks such as project management.

The first section summarizes the results of the study. A detailed description of the offices is contained in an appendix. The next section discusses what a project management support system could do for those offices. Examples of some of the abstract objects that a project management support system would use are in the final section.

4.1 Summary of the case study

The study involves five offices in the research and development (R&D) laboratory of a division that produces software for minicomputers. The offices each consist of a manager and a secretary or a fraction of a secretary. One office is the *laboratory manager*, the highest position in the R&D labs. A lab typically has 100 employees, including managers and engineers. Two are *section managers*, who report to the lab manager. A section typically has 25 engineers and managers. The last two offices are *project managers*; they report to a section manager.

While some variables are eliminated by studying only offices from a single area in a single organization, several challenges remain. First, the offices and the assumed business function are managerial. It is often believed that managerial work cannot be supported. Part of that belief comes from the personal nature of management. Second, each manager has a style; any successful support system must work with, not against, that style. The managers chosen for this study are especially varied in their style and background. Some work steadily on a task until it is completed, while others hop frequently from one task to another. Some are very computer oriented, willing to use any new computer tool that comes along, and they make extensive use of electronic mail. Others make virtually no use of computers at all, preferring paper and pencil. Even though this study is in no way sufficient proof that project management is a common business function, finding a common structure to the work done in these offices will be a strong positive sign.

4.1.1 Office mission and overview

All of the offices in the study had a common mission: to develop a strategy and implement it through the introduction of new products. The perceived mission was very consistent, even across different levels of management. There is some variance in how the managers *measure* their success in fulfilling their office's mission. The higher level managers tend to use success in the market place as a chief measure, while the project managers prefer measures such as how attractive the group is to work in, or a personal feeling that goals are being met.

The activities used by the managers can be grouped into four main areas: developing a strategy, managing their employees, managing their equipment, and of course, managing projects. Each of these are discussed briefly below. All of the levels of R&D managers spend some time in all of these areas. Higher level managers, however, usually spend more time on developing strategy, while lower level managers spend more time implementing strategy.

4.1.2 Developing a strategy

Developing a strategy is the least structured, least supported area of work in the offices studied. The textbook plan for setting a strategy is to assess where the organization is now, decide where it ought to go, and figure out how to get there [Sher82.] The managers are quite consistent about what information belongs in a strategy. A strategy begins with a *charter*, which stakes out the market segments that the manager's staff will address. Next, the charter is elaborated to describe the user, product, and technology focuses for the group. A strategy then sets out the short and long term goals for the group, statements such as "become a leader in business graphics by 1986." The goals answer the question "where should we go?"

The core of a strategy is its tactical plans. They address the question "how will we get there?" It is at this point that specific projects are described and resources are allocated for specific time periods. In the software industry, resources are principally people; equipment is usually not the limiting factor. All of the managers develop at least two tactical plans. One shows the coming year's plans, and another looks three to five years ahead. In addition to the tactical plans, most managers keep in their head, if not on paper, a "bag of projects". This is often where ideas for projects are stored until they become firm enough to be included in the tactical plan.

A secondary part of the strategy is a budget. At HP, budgeting is somewhat separated from the rest of strategic planning. A loose feedback path exists between strategy and budgets: the current budget affects what a manager can plan on accomplishing in the short term, whereas the goals of a group can affect the next cycle's budget.

When developing a strategy from scratch, a manager gathers information from a variety of sources and sketches out a strategy, more-or-less in the order presented above. The chief sources of input include:

- Objectives from higher management
- Information from subordinate managers and engineers
- Customer input
- Technology trends
- Competitive information
- Constraints (primarily budget)

Strategies are not usually developed from scratch, but rather they are revised periodically in response to outside economic developments or a change in direction dictated by higher management. Most managers said that they think about strategy all the time, though they would only sit down and work on it for a specific reason. Some accumulate ideas in their head, waiting until a presentation of the strategy is required to formally update the strategy documents. Others update the documents as changes occur, and make no special effort when a presentation is required. At higher levels, keeping the documents and overhead slides that are used for presenting and disseminating the strategy often become a major task.

Managers in this study use only two kinds of tools to support strategy development. Electronic spreadsheets are applied to the budget. Slide-making programs produce overhead slides that present the strategy. There is no assistance with the underlying strategic planning process.

4.1.3 Managing projects

The general goal when managing a project is to produce a quality product, on time, and within budget; the goals of managers at HP are no different [Rose81.] Managers accomplish this by approving initial plans for the project, making sure it progresses on schedule, and taking action when it does not. At HP, only the project managers

get involved in the day-to-day supervision of a project. Higher level managers receive regular progress reports and only get involved when a problem is reported that the project manager can not handle.

For software projects, there is a well defined *Software Product Life Cycle* (SPLC). The SPLC lays out steps to be taken in moving a project from beginning to end. It specifies a number of documents to be filled out along the way and the format of each. The basic outline of the SPLC is:

1. Investigation: Research the project, study the competition, build prototypes, prepare a schedule for the rest of the project's life.
2. Design: Prepare *external specifications* that describe the interface to a user or other software, design the internals of the product.
3. Implementation: Construct the product in software; debug it.
4. User testing: *Alpha test* the product with real users inside HP, *Beta test* the product with selected customers outside HP. Internally test the product in the quality assurance department.
5. Release and Post-release: Evaluate the product and its success in the market. Begin considering bug fixes and enhancement requests.

Individual managers vary somewhat in how closely they follow (or deviate from) the SPLC. Specific projects may also require a change in the SPLC. For example, a project that emphasizes prototypes blurs the distinctions between investigation, design, and implementation. By and large the outline above is followed by every manager and every project.

At HP, a project involves not only the R&D managers and engineers, but several other departments. Marketing, quality assurance, and manufacturing each assign people to work on a project. However, the project manager from R&D always has the overall responsibility for the projects successful completion.

One of the most difficult and yet most important tasks for a project manager is scheduling major project tasks and assigning responsibilities to individual members of the project team. Most managers studied used either the PERT system or the simpler GANT system, although these were usually employed to *present* an already planned schedule, rather than to analyze and develop one [Wies77]. The schedules were usually broken down by engineer and by month. The SPLC provides some "major milestones", such as investigation completed, external specification completed, etc.

Having made a schedule, the manager has to determine the teams performance relative to that schedule. With a well made schedule, it is straightforward to determine whether the project is on target. The schedule indicates when something should be done, and it either is or it isn't. The R&D lab studied requires documents each month, showing actual *vs.* scheduled progress toward major milestones, along with a summary of recent accomplishments and problem areas. As slippages occur, schedules must be updated, a very tedious task.

Since the overall supervision of a project rests with the R&D managers, much of their time is spent coordinating with other departments. The R&D project manager usually has responsibility for such tasks as arranging meetings and getting documents circulated and signed. Secretaries assist with some of these tasks.

Each project has associated with it a project notebook. This is usually implemented as a 3-ring binder, and contains a copy of every important document, report, or note, associated with a project.

This description is not meant to make managing projects appear completely algorithmic; it is not. A manager will deal with hundreds of *ad hoc* problems, most small but always a few serious ones, during the course of a project. However, the activities highlighted here, scheduling, monitoring, reporting, and coordinating, often take up the most time.

The managers made little or no use of any kind of support tool to assist them. One did make use of a simple list-keeper program during the later stages of a project to keep track of things that had to be done before releasing the product. There are now on the market several software packages designed to support the scheduling process. One example is the LisaProject system for the Apple Lisa [¡citation¿.] A manager graphically creates tasks, inputs their duration and assigned staff, and connects the tasks by their precedence relationships loosely following the PERT model. LisaProject fills in the dates and calculates the critical path automatically. The schedule can be dynamically edited and the dates and critical path instantly recalculated. More sophisticated systems also assist with the monitoring of a project, by letting a manager "check off" tasks as they are completed, revising the schedule as critical tasks are not completed on time, and even printing out reports of important tasks that are behind.

4.1.4 Managing people

Developing a strategy and the day-to-day managing of projects are the traditional responsibilities associated with project management. A less obvious, but no less important, responsibility is managing people. When asked what the fundamental resource they had to carry out their mission, all of the managers studied answered: people. In another industry, the answer might have been capital equipment or raw materials, or simply money. Any industry, though, needs people to carry out its mission, and the management of people is qualitatively different from the management of equipment. People must be recruited, trained, evaluated, and ultimately released through transfer, retirement, promotion, or whatever. Managers at all levels are judged in part on their ability to attract and keep top people, and this takes work on the manager's part.

Managers have a number of channels to locate candidates for employment. HP has an active college recruiting system, and a centralized computer facility makes information on any of them available to all managers. Employment advertisements are

sometimes used when more experienced help is needed. Each manager also has a number of personal sources, such as friends at other firms or universities, that provide leads. Transfers from other parts of HP are also common, and are facilitated through internal postings of open positions.

HP has a formal evaluation process used company-wide, which provides for written yearly evaluations. Most HP R&D labs also use a standard quarterly evaluation system. Many managers also prepare monthly objectives with their employees; these are used at the end of the month for informal evaluation of individual progress. All of these mechanisms are used both as feedback to the employee and to drive the salary system. Engineers and (separately) managers are *ranked* each quarter to produce a number from 0 to 100 indicating relative performance. The rankings begin on a project by project basis and are merged and sorted at each level of management. At each level, a manager has to justify why her employee should be ranked above some other manager's employee, and thus the direct relation to evaluations and monthly objectives.

The rankings each quarter drive the salary system. The ranking, or performance level, is placed on the y-axis, while "years since first degree" is placed on the x-axis. Actually, a corporate-wide computer system generates the *target salary* from the x and y coordinates. Managers are free to plan pay raises during the coming year to make an employee's actual salary match the target salary.

The cumulative time involved in preparing the various evaluations and objectives, preparing for and attending the ranking sessions, and working out salary plans for each employee, is substantial. The time spent recruiting employees varies, but on the average is not large. Word processing is the only tool used by managers to assist with evaluations and rankings, and there are not any on the market designed for this purpose. In preparing an employee's monthly objectives with him, a manager usually makes use of the current project schedule to determine what should be done. To produce quarterly

evaluations and prepare for ranking sessions, the three preceding month's objectives are the main source of information. Likewise, the four quarterly evaluations are the main source of information for preparing the yearly formal evaluation.

4.1.5 Managing equipment

Managers are responsible for acquiring and maintaining the equipment needed by their staff to successfully complete their projects. At the studied R&D lab, large time-sharing computers are requisitioned and maintained on behalf of the entire lab by a special group; managers simply include a share of the cost for this group's equipment and services on their budget. Other equipment is requisitioned by a specific manager and charged to a specific account.

Major capital purchases are usually planned in advance on the *capital budget*, which is separate from the overall budget mentioned in the strategy section. The link between the two budgets is via a *depreciation* entry on the main budget that pays for equipment purchased on the capital budget month by month. The two budgets are controlled by somewhat different forces in the organization and grow or shrink according to different financial variables of the company.

This particular lab did most of its work via terminals connected to the time-shared minicomputers, and so had very little capital equipment of its own to manage. Managers keep track of their equipment and see that it is serviced as needed. When something becomes obsolete or broken beyond repair, it is discarded.

4.2 What a Project Management Support System could do

The preceding section outlined the inner workings of the studied offices. Each manager in the study approaches the job differently, yet on two important points, they were all very consistent: what the basic tasks are, and what information and support

is needed to accomplish them. This section will consider what a Project Management Support System (PMSS) could do that the currently available tools do not.

A number of tools were described that are currently available and useful to the managers studied: word processing, slide-making, simple databases, and schedulers being the major ones. One could combine them, give them a common user interface and a standard data format that allowed exchange of data between subsystems, and call it a functional support system. In fact, Apple's Lisa is essentially just such a system. As argued in a previous chapter, though, simply packaging some task-oriented generic tools under a common user interface does not significantly alter their functionality.

The sample of a PMSS that will be presented demonstrates two different ways a functional OA system could better support an office than generic tools. In some cases, a PMSS could better support a specific activity because it knows more specifically what that activity is. In others, improvement arises from a PMSS having information from other areas in the office that a tool would not have.

In addition, the entire PMSS could be used in two different ways. It could be viewed as a tool to assist a new person in the office, by providing extra guidance when performing tasks, or a default structure for required files and documents. This applies not only to clerks and secretaries, but to managers as well. In fact, it may apply especially to managers. A new manager often has a whole range of new responsibilities, and there is seldom a manual for the position. The other way of viewing a PMSS is as a tool for the experienced office worker, providing information and support at levels that surpasses what is possible through task-oriented tools.

The following sections consider each of the areas of project management discussed in the previous section, and present examples of how a PMSS could support them. Keep in mind that these are only a few examples, chosen to illustrate the different ways a functional support system could be better than generic tools.

4.2.1 Strategy

As described, the strategy area of project management may seem unsupportable. It involves creative thought that is usually based on soft rather than hard information. Yet, there are several ways that setting strategy could be supported by a PMSS. First of all, it could help organize and present the strategy. The study found that there was, independent of personal taste, a fairly standard set of documents that made up a manager's strategy. It also found that they spent a significant part of their time updating, sorting, and making their strategy presentable. Managers often have hundreds of slides; just finding the slides needed for a specific presentation could be an hour's work.

4.2.1.1 Organization and presentation

A PMSS could provide the needed organizational support for a strategy. It could also make it easy to update and make presentable copies of any part of the strategy. A possible model is that a strategy consists of a number of components, each of which has a current information object and a number of note, slide, or document objects attached to it. Each component object of the strategy could be edited on line in an editor designed for that object. Components such as the objectives may be implemented as an ordinary text file, with an ordinary word processor used to edit it. The budget component would be edited with a spreadsheet-like program, and a 5-year tactical plan may be edited with a system like LisaProject. When a manager wants to start with a new component, rather than edit an old one, the PMSS could provide a template to be filled in.

Managers usually need more than one presentable version of a component object, since they deliver their strategy to a variety of audiences. The PMSS slide maker could begin by filling in the company logo and the author's name and the date. The object that the slide is being made from could decide what, if any, the default contents of the

slide should be. For example, if a manager had just updated his charter and wished to have a slide made of it, the PMSS could copy the text into the slide object. The manager could then add whatever additional markings he wanted. In the case of a tactical plan, the PMSS could draw a PERT or GANT chart that illustrated the plan, and then let a manager add any annotations as needed.

When a new manager is told to prepare a strategy presentation, she may be unsure about what is expected. A PMSS could help by providing an organization that could be used as a starting point. Over time, the manager may vary it to suit her personal tastes.

4.2.1.2 Creation

Although it is certainly not the place of a PMSS to create strategies, it could provide support to some of the activities involved. A good instance is in the budget setting process. The current spreadsheet programs are useful, but a PMSS could do much more. Most of the numbers in the budget are not decided independently, but depend on some other numbers. Spreadsheet programs could handle the simple relationships, such as "cell C4 = cell C3 - cell B4." Consider, for example, the entry for depreciation: take the current inventory of capital equipment, determine for each their lifetime and the amount to depreciate this year, add these amounts, and enter the total.

Expert systems could also be applied to tasks such as budgeting. An expert system is "a computer program that embodies the expertise of one or more experts in some domain and applies this knowledge to make useful inferences for the user of the system" [Wate83]. The technology is new enough that a more precise definition does not exist. However, the following characteristics, quoted here from [Brac83], of expert systems will help differentiate them from ordinary computer programs:

- Expertise. High-level rules, avoidance of blind search, and high performance.

- Reasoning by symbol manipulation.
- Intelligence. Fundamental domain principles and weak reasoning methods.
- Difficulty or complexity (of the domain.)
- Reformulation. Conversion from a description in lay terms to a form suitable for expert-rule applications.
- Reasoning about self in various forms, especially for explanation.
- Type of task (that an expert system is put to.)

An expert system can be constructed as a set of *rules* that operate over properly encoded data. For a PMSS, the data consists of abstract objects in the office, and the rules describe relationships between the objects in an if-then manner. An example rule is "if the group works with printed circuit boards, then allow $3000 times the number of people working with them for design work." One advantage in using rules rather than a spreadsheet relationship is that an expert system can explain how it arrived at a conclusion. Such information is necessary to a more experienced user who wants to modify the rules.

4.2.1.3 Tactical plans

A PMSS can also support the development of tactical plans. LisaProject works in very simple terms: nodes with some duration and precedence relations between the nodes. The actual way tactical plans are laid out involves a progressive refinement of the definition of each project. Managers begin with only a vague idea of what a project will do. Estimates of project size, when it needs to be on the market, and what other efforts it must follow are added next. When the size of the staff is known, then the duration of the project can be estimated. A PMSS can support the progressive refinement of tactical plans by automatically putting to use as much information about a project as a manager can give it. For example, if the duration and start date of a

project are known, then this can be used to fit it in to the tactical plan automatically. Rules can be written that examine a tactical plan and inform the manager if something does not make sense, such as requiring a project to complete before it has begun.

4.2.2 Managing projects

A PMSS can organize a manager's project notebook. As mentioned, each software project is expected to follow the Software Product Life Cycle (SPLC). In reality, most projects deviate slightly from this standard. The general framework, however, is useful for all projects. Major checkpoints in the SPLC (investigation complete, design complete, alpha test complete, etc.) are referred to both on the schedule and in progress reports. A PMSS can provide a customizable SPLC that is connected to the other aspects of project management, such as progress reporting. When a new project is started, its manager can tailor the SPLC to the specific needs of the project. Once created, it can serve as an intelligent checklist, which not only records what has been done, but can remind a manager of other tasks that need to be done.

The SPLC specifies a number of documents that are to be completed along the way. Some are simply textual reports that have a very well specified organization. For these, a PMSS can provide not only a word processing subsystem, but load it with the template for the document. Others, such as the *lab product datasheet*, consist of a single sheet with a number of windows to be filled in with facts about the project. They are frequently updated and are widely dispersed. Most managers make use of a slide-making program to prepare these. A PMSS, with the knowledge of a project described in the strategy section, could present a graphical editor that already has not only the borders and headers drawn in, but much of the information.

This example illustrates a general point about functional OA systems: information that would otherwise be kept in several unrelated areas, can be kept in a single place. Updating the information in one place will automatically update all other documents

that make use of it. In the example above, if a manager changed the name of a project in the project repository described in the strategy section, then a new lab product datasheet could be produced with no further work, since it would lift the name from the project repository information.

A PMSS, loaded with the information in the project database, especially its schedule, can handle much of the reporting that managers currently compute by hand. If individual engineers simply enter into the PMSS when a task listed on the schedule is completed, then all of the slippage reports can be automatically generated. The other progress reports can be semi-automated as well. For example, the goals-for-next-period in one month usually become the objectives-for-this-period in the following month.

4.2.3 Managing people

The managers studied all spend a substantial part of their time preparing the various performance evaluations (monthly, quarterly, and yearly.) Setting monthly objectives for an employee is usually based to a large degree on the schedule for the project that the employee is working on. A PMSS should display for a manager the upcoming project-related tasks for which the employee is responsible. Other important objectives may be to take a class, or perhaps the employee has vacation scheduled. Such information could be saved with the employee's record in the PMSS, and merged with the project information for a manager preparing a monthly objective.

At the other end, a PMSS can support a manager preparing an evaluation. For each type of evaluation, the information needed is roughly the same: what has the employee done, and how did that compare with the planned objectives. When the monthly objectives are prepared on-line, then they become available for browsing when preparing a quarterly evaluation. As for other documents, a PMSS can know the format of each type of evaluation, and provide an editing environment appropriate to that type of document and fill in the headers, name, date, etc. automatically.

4.2.4 Summary: functional OA support techniques

A variety of ways that a project management support system can better assist a project manager have been presented. They tend to fall in a few classes, which are likely to be typical of any functional support system:

- Assistance to a new employee.
- Better context of specific tasks, and support tailored to that context. The idea of a generalized editing system, where each object is edited in an editor that makes the most sense for that object.
- Ability to use information created in one activity in the support of another automatically.
- Storing a piece of information in one place, and making the process of changing it in that one place sufficient to update all other objects that use it.
- Embodying knowledge of office operations in rules, and using these rules to assist the employee throughout the office.

4.3 Major objects found in project management

This section describes some of the major *objects* found while studying the project management offices at HP. The objects presented here are primarily from the strategy area, although most objects are used throughout the office. They will be presented in "pigeon OSL", utilizing more English so that detailed knowledge of OSL will not be necessary. Actually, *classes* of objects will be defined, and the individual objects are *instances* of a class.

The purpose of this section is to illustrate the use of abstract objects to describe the entities in an office. The OSL descriptions are not in complete detail, and not all of the classes will be defined. Compare these specifications with their descriptive

counterparts in the preceeding sections. Notice how OSL forces an analyst to be specific and explicit about the structure of classes. In the next chapter, these examples will be used to demonstrate the conversion of OSL objects to software objects.

Class STRATEGY models the entire strategy of a manager. It only holds references to other objects, and has no primitive information of its own.

```
Class STRATEGY
  Charter: CHARTER
  Focuses: FOCUSES
  Goals: GOALS
  Projects: PROJECT
    multivalued
  Short-range-tactical-plan: TACTICAL-PLAN
    where TACTICAL-PLAN.range <= 1 year
    multivalued
  Long-range-tactical-plans: TACTICAL-PLAN
    where TACTICAL-PLAN.range > 1 year
    multivalued
  Objectives: OBJECTIVES
  Budget: BUDGET
  Slides: SLIDE-FOLDER
```

This class illustrates an important point in OSL specifications: single vs. multiple values. A single-valued component may have only one object as its value. A multivalued component, on the other hand, has as its value a *set* of objects, each being an instance of the specified class.

There are usually many ways to express what is found in an office in OSL. Objectives, for example, is described as a component that takes a single instance of class OBJECTIVES as its value. An alternative would be to make objectives be multivalued, where each value is an instance of a different class OBJECTIVE that describes a single objective. The former allows a single object to be responsible for managing all objectives. Projects, on the other hand, is modeled as a multivalued component. This is because a number of other objects, principally the tactical plans, serve the purpose

of organizing the projects. The projects component allows direct access to a project without any organization, and allows tenative projects to be retained without being part of a tactical plan.

Making decisions such as those above can be very difficult. One way to decide is to consider the procedures and steps used in the office and map them into operations on objects. Usually the operations necessary will dictate the organization of the classes. These distinctions may seem neglibile, but will become important in the next chapter.

Several of the components of a strategy do not require any special operations, and can be modeled as a block of text and a set of slides that were prepared to describe that text.

```
Class CHARTER: STRATEGY-COMPONENT

Class GOALS: STRATEGY-COMPONENT

Class FOCUSES: STRATEGY-COMPONENT

Class STRATEGY-COMPONENT
  Text: TEXT
  Slides: SLIDE-FOLDER
```

Projects are more complex. Most of the documents associated with a project, such as the external specifications, quality plan, etc. are components of the notebook.

```
Class PROJECT
  Name: STRING
  Number: PROJECT-NUMBER
  Manager: HP-EMPLOYEE
  Staff: HP-EMPLOYEE
    multivalued
  Schedule: SCHEDULE
  Notebook: PROJECT-NOTEBOOK
  Slides: SLIDE-FOLDER
```

A tactical plan simply relates a set of projects to each other and to a calendar. Information need not be duplicated in the tactical plan, since it can be read from the consituent projects.

```
Class TACTICAL-PLAN
  Range: TIME
  Start: DATE
  Projects: PROJECT
    multivalued
  Relationships: PROJECT-RELATIONSHIPS
    multivalued

Class PROJECT-RELATIONSHIPS
  Project-1: PROJECT
  Project-2: PROJECT
  Date: DATE
```

An objective is modeled as a textual description along with time and priority information. They are organized by instances of class OBJECTIVES The class contains operations to prioritize objective entries.

```
Class OBJECTIVES
  Entries: ordered set of OBJECTIVE
  Slides: SLIDE-FOLDER

Class OBJECTIVE:
  Task: TEXT
  Priority: NUMBER
  Start: DATE
  Est-duration: TIME
```

Finally, most objects have one or more slides associated with them. Slides are graphical representations of an object, which may have additional annotations added to them. Slides are organized into a browsable collection by SLIDE-FOLDER instances. SLIDE-FOLDER, in turn, as built out of a more primitive class, FOLDER.

```
Class SLIDE-FOLDER: FOLDER
```

```
  Elements: SLIDE
    multivalued

Class SLIDE
  Parent: THING     % i.e., any object
  Height: DIMENSION
  Width: DIMENSION
  Components: GRAPHICAL-OBJECT
```

4.4 Summary

The term *business function* was used frequently in chapter three; one of the goals of this chapter was to present an example of one. Project management involves much more than the day-to-day management of a project, and because a functional OA system has the full context of the office work it is able to support the office work much better than generic tools. Managerial support was emphasised, since the ability to support managerial work has been doubted. Finally, examples of OSL specifications were presented to demonstrate clarity and explicitness in OSL specifications as compared to equivalent English descriptions. The OSL examples also serve the discussion in the following chapter, which describes how OSL objects can be related directly to software.

Chapter 5
Implementing a Functional Office Automation System

The previous chapter described a group of offices and outlined a functional OA system to support them. It set out a number of ways in which a functional system surpasses generic tools. A complete project management support system (PMSS) is likely to be large and complex, and so we proceed to address the problem of how to design and implement such a system.

This chapter presents an architecture for a PMSS, and more importantly, a design methodology for creating other functional OA systems. The strategy is based on successfully modeling (in software) the abstract objects discovered while studying the function. The first section briefly describes object-oriented programming languages, and shows how an OSL class definition can be converted to a software equivalent. The second section describes an electronic desktop architecture that supports the manipulation of objects. Finally, a functional OA toolkit is described that supports customization at two different levels.

5.1 Object-oriented programming

5.1.1 An object-oriented language

An object, in the programming sense, is a collection of data that can be operated on through a very clearly specified set of operations. The internal representation of the data is known only by those operations. Usage of an object, including simply accessing part of its data, is impossible except through one of the operations defined for the object's type.

An implementor of a new type of object first defines what the object looks like on the inside. This involves listing the component fields the object should have. Then, the implementor defines any needed operations for the new object type. Operations are something like Pascal procedures, except that operations are specific to a type of object, while Pascal procedures are not associated with any types or variables. *Inside* the body of an operation, free access to the fields of an object are permitted. *Outside* the body, access to an object is accomplished by "sending the object a message", where the message indicates the operation desired and the arguments, if any.

Several object-oriented language dialects are available today. They include Smalltalk [Gold83], which has been implemented on various machines, and Zetalisp, which runs on Lisp Machines [Wein81]. This thesis uses the Zetalisp dialect for examples because a variety of Lisp systems have adopted its object programming facility. In Zetalisp, types of objects are called *flavors*, and the actual objects manipulated are *instances* of a flavor. The operations on objects are called *methods*.

The following example defines a flavor *box*, methods *set-color*, *set-size* and *set-position* that alter a box's parameters, and a method *draw* that draws it.

```
(defflavor box
  (color        % The color of the box
   x-origin     % The coordinates of the upper-left corner
   y-origin
   width        % The dimensions of the box
   height       %
   )
  ()            % No inheritance (to be discussed)
  )

(defmethod (box set-color) (new-color)
  (setf color new-color))

(defmethod (box set-size) (new-width new-height)
  (setf width new-width)
  (setf height new-height))
```

```
(defmethod (box set-position) (new-x new-y)
  (setf x-origin new-x)
  (setf y-origin new-y))

(defmethod (box draw) () % Takes no arguments
  (set-color color)
  (move x-origin y-origin)
  (draw (+ x-origin width) y-origin)
  (draw (+ x-origin width) (+ y-origin height))
  (draw x-origin (+ y-origin height))
  (draw x-origin y-origin))
```

Note that a user of an instance of flavor box need not know whether it stores a coordinate and dimensions, or the upper-left and lower-right coordinates, or whatever. Now assume *my-box* is an instance of flavor box and that the messages set-color, set-position, and set-size have been sent to it to initialize its instance variables. To send my-box the message draw, the code in Zetalisp would be
`(=> my-box draw)`.

Object-oriented programming offers two principal advantages over traditional (i.e., Pascal-ish) programming. First, it allows the implementor to hide the details of implementation from the user. The methods of a flavor and their arguments become a protocol specification. Over time, circumstances may require the underlying implementation to change, but the user of a flavor does not need know or care. Second, it provides *generic* operations. Because many objects can each have a method with the same name, when the same message is sent to these different objects they each invoke their own code to manipulate their own particular internal structures. For example, suppose that another flavor, *circle*, has been defined, and that it also has a message called draw. A higher-level program might simply have a number of instances, some circles and some boxes. It could send them all the draw message, and each instance would use the method defined for its type.

5.1.2 Using object-oriented-programming

A natural way to implement a functional OA system is to define flavors in software that mirror the classes of objects found in the actual office. Likewise, the operations performed on objects in the office can be implemented as methods for a flavor. The methodology for constructing a functional OA system then becomes:

1. Study offices using OAM.
2. Describe the business function using OSL.
3. Implement the corresponding flavors and methods in software.

From this point on, the terms *classes, objects, and operations* will refer to the abstract entities, whether they are in the office or modeled in software. The terms *flavors, instances, and methods* will refer specifically to software. Note that there will generally not be a one-to-one mapping between flavors and classes, or objects and instances.

Some of these classes, and certainly some of the operations, will be very complex. For example, chapter four described some of the components of the project notebook, and how a PMSS could present each in an editor suited to the component. In general, a functiona OA system should be able to edit any object in an editor designed for that object's class. For some classes, this may mean implementing many operations, each operation possibly being complex. There is also the matter of a user interface to the system. Fortunately, design for sections such as the user interface need not be repeated for each functional OA system. In addition, a powerful feature of most object-oriented programming languages can be used to reduce the complexity of implementation: *inheritance.*

Flavors can inherit both the internal structure (i.e., the fields) of other flavors and their methods. A flavor can even have several parents (the term *flavor* originates in

the notion of "mixing in" several flavors to get a new flavor.) Many times, an abstract object in the office can be supported by defining a new flavor that inherits a set of more primitive flavors, and possibly adding a few special methods. Primitive flavors might include a textual object, a graphical object, and a "folder" object.

Consider the class `STRATEGY-COMPONENT` described in the previous chapter. Its field *text* was specified to be of class `TEXT`, and field *slides* was of class `SLIDE-FOLDER`. `SLIDE-FOLDER`, in turn, held objects of class `SLIDE`. The implementation of a flavor TEXT corresponding to the OSL class `TEXT` need not begin from scratch. If it inherits the primitive text flavor, most of the work is done. Likewise, the flavor SLIDE representing the OSL class `SLIDE` can inherit the primitive graphical flavor to provide most of its functionality. Both flavors STRATEGY-COMPONENT and SLIDE-FOLDER could be entirely handled by inheriting the primitive folder flavor.

An example of how an OSL class will look in software will be presented. It will be easier to understand after looking at a model for an electronic desktop that manipulates objects.

5.2 A desktop architecture

Office employees can not be expected to write programs in Zetalisp to work with their support system. While one approach to office automation is to define the user interface and then implement the subsystems that do the work, the situation here is the reverse. The challenge now is to create an interface that allows instances of various flavors to be manipulated easily without constraining the way flavors present themselves. This section presents *ObjectDesk*, a desktop architecture specifically designed to work with objects.

Several "object-oriented" office systems have been introduced into the market recently, most notably Apple's Lisa and VisiCorp's VisiOn. They present to a user an

interface that electronically models his desktop. Textual descriptions or graphical *icons* represent items such as folders, pads of paper, and even a wastebasket. Icons can be "opened" to view or modify their contents. Both offer some ability to exchange data between applications.

VisiOn was designed to support development of software by independent vendors. It rigidly defines the interactions a program can have with a user, and thus programs developed independently will have a consistent user interface. VisiOn also manages the windows programs are displayed in, and the interchange of data between programs.

While VisiOn could be used to implement a functional OA system, ObjectDesk has several advantages. First, VisiOn is designed to support entire products, whereas ObjectDesk works with individual objects. Products need a fair amount of software added to them before they can run on VisiOn, whereas a class requires very little extra software to run on ObjectDesk. VisiOn presumes that a relatively small number of different products will be installed, while ObjectDesk can deal with many. Second, new classes can be defined using inheritance, often drastically reducing the amount of new code that must be written. Products can not inherit from other products. Third, VisiOn retains a very strong grip over the user interface in an attempt to make different products behave similarly. ObjectDesk gives much more control to individual objects, and depends instead on "rules of good citizenship" and inheritance to obtain a consistent user interface. Finally, ObjectDesk itself is much simpler. Since it hands over much of the responsibility for interaction to individual objects, ObjectDesk can be written in a few thousand lines of Zetalisp code.

Apple has a "toolkit" for constructing Lisa applications that comes closer to the ObjectDesk approach than VisiOn. In particular, applications are objects, and are created by inheriting from more generic applications and adding new methods or redefining inherited ones to achieve the desired behavior. Hopefully, similar technology will be developed for a variety of computers and operating systems.

A disclaimer is in order before proceeding: this is not a human factors thesis; absolutely no claim is made that the desktop model presented here is the "right" user interface, or even that it is close. It was built solely for the purpose of demonstrating the usefulness of the methodology.

5.2.1 Philosophy of the desktop

There were several goals for the desktop model:

- It should be easy to use, both by an end user and by an implementor of a new class.
- It should allow objects to be accessed at any time and in any order. A user should be able to switch between objects at will.
- It should encourage integration between different classes, defined as the ability to share information at a very high level. An array of bits that make up a picture is considered rather low-level information, the instructions that could re-create the image somewhat higher, and knowing how to communicate with the object that generated the picture the highest.
- It should be very flexible and extendable, making as few assumptions about how classes will present themselves or represent internally themselves as possible.

There is obviously some conflict between these goals. For example, being flexible will tend to contradict encouraging integration. Consider a slide object and a document object. They will likely have very different ideas on how commands should be input, how the internal data should be stored, and how to display themselves on the screen. How can a slide incorporate a part of a text document, or a document include part of a slide, when they are so different internally and externally?

ObjectDesk arbitrates between the goals by establishing a simple policy: the desktop model establishes *what* a class has to be able to do, but never *how* it should do

it. To operate on the desktop, a class has to implement a set of operations that the desktop uses to control it. Each class also implements standard operations that enable other classes to work with it. A number of auxiliary flavors have been implemented that meet the required protocol, each designed for a basic type of user interface. Most classes have been implemented using one of these support flavors to manage window space, menus, and the like. This is why there are more flavors than classes. A class is usually implemented with a principal flavor whose methods implement the class's OSL operations, while one or more support flavors implement the methods required to run within the desktop environment. The following sections present an overview of the implementation and use of the desktop; the last section discusses how these support modules can form the basis of a toolkit.

5.2.2 The user's perspective of the desktop

Every object on the desktop, including the desktop itself, has a *window* in which to operate. A window is a rectangular screen area, occupying some or all of the available display space. A window normally has several panes inside it:

- A header pane, giving the name of the object displayed inside, and possibly some status information, such as whether it is currently selected.
- A noun pane, containing a list of component objects.
- A verb pane, containing any *idiosyncratic* operations implemented just for this object's class.
- A working pane, used to display the contents of the object itself.

Flavors are not required to have any of these panes. Folder-like classes usually do not have a working pane, and terminal classes do not have a noun pane.

The desktop, behaving like a good object, has a noun pane displaying all of the top-level components of the desktop, and a verb pane displaying operations that can

be applied. The desktop's verb pane is special in that its operations can be applied to any object, and are called *system operations*. That is why individual objects only display their idiosyncratic operations.

The desktop object plays two independent roles. It is the *desktop manager*, collecting input, relaying output, managing windows, and so on. At the same time, it behaves as any other object, and obeys the same protocol required of any other object. The desktop manager's purpose is basically to parse input into messages to be directed to an object lying on the desktop, that object possibly being itself.

One object on the desktop is, at all times, the *selected object*. All input is either part of a message to be sent to the selected object, or the picking of a new selected object. When using a cursor-tracking input device, selection of anything inside the selected object's window or from the system verb pane is part of a message to be sent to the selected object. Selections outside the window are considered the picking of a new object. The only exception occurs when an operation is already running and is prompting for additional arguments. As an argument, picking a window other than the selected object's window means picking the whole object displayed within that window.

The following description of some of the system verbs will clarify the user interface. Remember, all of these are interpreted in the context of the selected object.

- OPEN: Open the picked object in its own window.
- CLOSE: Close the object, and destroy its window.
- DELETE: Delete the picked object from the selected object.
- COPY: Make a copy of the object, request a window as an argument, and add it to the components of the object displayed in the window.

5.2.3 The implementation of OSL objects

Inheritance can be put to use from the beginning to make the implementation easier. All flavors used to implement OSL classes have, directly or indirectly, the flavor DESK-OBJECT as a parent. The definition of flavor DESK-OBJ is:

```
(defflavor desk-object
  (desk                 % The desktop we are currently part of
   noun-alist           % Alist of noun print-strings and objects
   verb-alist           % Alist of verb print-strings and methods
   window               % The window object we display ourself in
   )
  ()                    % No inheritance
  )
```

The noun and verb association lists (alists) are the main variables. The noun alist maintains a mapping between the component objects and the printable names to use on the menu. The verb alist maps the methods available to a user to printable names. Zetalisp allows a method to be invoked automatically when an instance is created. Each flavor writes an initialization method that creates the noun and verb alists that match the OSL specification.

An alternative design would have each flavor provide for its components with individual instance variables, so that they looked more like their description in OSL. The problem with this design is that it does not allow for any change in the specification of the components of an object. It is also much harder to implement, since operations can only be implemented with reference to specific instance variables which will be different for each flavor. Using the first approach, methods that manage the window, menus, and any other methods that need to operate on the entire component list need only be written for DESK-OBJ and inherited by all other flavors.

5.2.4 The desktop manager

This section describes the inner workings of the desktop object, and in particular, its role as desktop manager. The top-level of the desktop manager reads input *blips* from the input system, and formulates messages for the objects it manages. There are several types of input blips:

- (KEYBOARD character): a character from the keyboard.
- (MENU type item): Menus have a type, such as noun or verb; when picked they return both their type and the item picked.
- (POINT button x-coordinate y-coordinate): A point, in pixels relative to the selected window and the button that was pressed.
- (WINDOW object): A window other than the selected object's window, and the object inside that window.
- (ABORT): A special blip telling the desktop and the selected object to cancel the current operation (normally used when arguments have been requested.)

The input subsystem gets blips by polling each registered input device. When a device has input, the input subsystem classifies it as either a keyboard character, a cursor motion, or a mouse-button (keys on a keyboard can be defined as cursor motion or mouse-button keys.) Keyboard input is returned immediately as a keyboard blip. To process cursor motion or mouse-button inputs, special desktop methods are invoked.

5.2.4.1 Cursor motion

The process-cursor method takes either a direction or an absolute point, and determines the new absolute cursor point. It then determines which object currently owns that spot. If it is the selected object, a message is sent telling it to highlight that spot,

and to return the input blip that should result if the select button is pressed without any further cursor motion. Each class is free to implement its own style of highlighting. If the cursor is over any other window, the desktop itself returns a window blip corresponding to the object over which the cursor is positioned.

To illustrate this mechanism, consider a typical folder object, containing a header, noun, and verb pane, and assume it is opened and selected on the desktop. The noun and verb panes are menus, and would like to reverse-video a menu item whenever the cursor is over any part of that item for feedback. The cursor moves, and the input device detects the motion. It sends a message to the desktop, which decides that the cursor is over the selected folder. The desktop sends a message to the folder, supplying the cursor location in window-relative coordinates. The folder determines that the cursor is over the noun menu, reverses the videos on the new item, and un-reverses it on the previous item. Finally, the folder returns a blip (MENU NOUN item) to the desktop, which is remembered as a *potential selection*. Pressing the select button would then make the actual selection.

5.2.4.2 Button pressing

When an input device receives input that is considered a button pressing, it acts on the potential-selection. At present, the only button defined is the *select* button. Any other button pressing causes a POINT blip to be returned using the current cursor coordinates. (Objects can define uses of the other buttons.) The action the desktop takes depends on the kind of input blip:

- KEYBOARD: A message is sent to the selected object, `(=> object KEYBOARD-INPUT character)`.
- MENU: If the type is NOUN, it remembers the item as the current noun, and sends a message to the selected object `(=> object NEW-NOUN item)`. The purpose of this message is to let an object alter its display, typically by reverse video, a menu selection, to confirm the button press. The

object need not remember this noun. If the type is VERB, the message `(=> object item current-noun)` is sent, where item is the item in the verb menu blip.

- POINT: The message `(=> object POINT button x y)` is sent to the selected object.
- ABORT: The message `(=> object ABORT)` is sent to the selected object.

With this mechanism, implementors are given a high degree of freedom to create a user interface most appropriate to the object being implemented, while at the same time keeping both the desktop and the requirements of an object to be manipulated on the desktop very simple. It also gives objects maximum control over their own display and interpretation of input, while allowing many different kinds of objects to be accomodated. Through required system operations and standard noun-verb command structure, a common user-interface is encouraged. Use of the toolkit, which contains many modules to help objects manage their windows and menus, will also contribute consistency to the overall user-interface.

When an operation requires additional arguments, it must read them using the same input subsystem as the desktop. Objects are also strongly encouraged to watch for the ABORT input blip, and to return control immediately to the desktop when one is received. The desktop cannot just take control itself, since the control flow must be unwound back through the object.

There is an assumption that the operations implemented for objects will be relatively short in duration, and that operations should not read unlimited numbers of arguments. For example, an object should not have an idiosyncratic operation EDIT, which proceeds to read input blips and update the document accordingly until a keyboard blip containing a control-Q is received. Rather, an object that can be edited should have as individual operations the editing commands, returning to the desktop

after each one. The reason is that as long as an operation is running, the desktop cannot help a user who wishes to select a different object.

5.3 A functional OA toolkit

The desktop architecture presented is adequate for the manipulation of objects by a user. The protocol between the desktop and its objects is very simple allowing great flexibility in object implementation. Unfortunately, a side effect of the desktop's liberal policy is that objects must do a large amount of the "grundge work" involved in managing a window, menus, and the like. Ideally, an implementor of a new class of objects would spend a significant percentage of time on the idiosyncratic operations that perform the "interesting work" and very little time on operations supporting grundge work.

It is not just the engineers working for the OA vendor that need support. The need to customize a system to an organization or a particular person has been acknowledged in this thesis, and in most cases it will not be practical for this to be done by the OA vendor. Rather, individual users and organizations will have to be able to tailor a system to their own needs and preferences. This section discusses solutions to these problems.

5.3.1 User customization

The simplest form of customization is that done directly by its end users. The users are presumed to have neither knowledge nor interest in computers or programming languages. The customizations available at this level make use of ordinary operations on objects, invoked through the standard user interface. Two types of customization possible at this level are the setting of defaults and the redefinition of relationships between objects.

Some objects may have a complex internal structure that can be presented in several different ways. Objects may also have data that is editable in more than one way. Users

will likely have preferences for how an object presents itself, and how an object can be edited. It is entirely possible for a textual object to emulate the behavior of any number of popular word processors, since the object itself interprets the keyboard and cursor keys.

Many objects will contain references to other objects. Taking a step back from a single object, a network will become evident showing the path or paths that must be traversed to locate an object. For example, chapter 4 presented an organization of a strategy and its components. Individual users will likely have different preferences for just how the components of a strategy should be tied together. For example, one manager may prefer that each project be a top-level component of the strategy object, rather than only accessible through a "project-list" component of a strategy. He might prefer it in both places. Such reorganization can be accomplished by using a system verb LINK. LINK functions similar to copy, except that only one instance of the object exists; the various links to the object only make it appear to be in more than one place.

Making these types of customizations available to a user increases the complexity of the objects for the implementor. The LINK mechanism may seem simple, but the object on the receiving end has to be able to dynamically update its list of components and any menus it uses to describe the component list. Others, such as multiple presentation formats, require a substantial effort on the part of the implementor.

5.3.2 Building new objects with a toolkit

Tailoring a standard system to a new organization may require the creation of new classes of objects and/or the addition of idiosyncratic operations to existing classes. The same situation will likely apply at the OA vendor, where new support systems are designed and implemented. The personnel involved in this work are assumed to be "technical" people, probably with a degree in or experience with computers. A longer

term goal is to reduce the technical knowledge required to implement a new flavor, but in the short term, it requires programming.

A simple toolkit need only contain a number of classes that manage different types of windows. An implementor of a new class need only select the window type that most closely matches his needs. An object using a window from the toolkit can redirect most of the messages from the desktop to the window object. The toolkit would also contain several classes that implement different kinds of menus, to be used by the window classes as needed. Consider the following window descriptions:

- FOLDER-WINDOW: presents a noun menu listing the components of a folder, and a verb menu containing idiosyncratic operations.
- VALUE-WINDOW: like a folder-window, but also includes a pane where the object's current values are displayed. The value pane is actually a menu of type NOUN. A class can have a operation such as SET which uses selections from the value menu as the noun.
- EDIT-WINDOW: a specially designed window to display and edit a textual object. It contains noun and verb panes, like a folder-window, but reserves most of the area for its editing pane.
- GRAPHIC-WINDOW: a folder-window with an extra pane designed for graphics. The graphics pane menu items are graphical components, such as boxes, circles, vectors, and text. These graphical menu items can be selected with a cursor like other nouns.

This level of support makes the job of implementing new flavors significantly easier, and is suitable for the OA vendor's R&D staff, or perhaps a technical consultant working in an organization using a support system. The goal should be to make it possible for those who do OAM/OSL analyses to directly implement their findings. These people will typically not have extensive programming experience. This research has shown

that the *data* aspects of OSL can be directly and usefully translated to a software equivalent in an object-oriented language. Unfortunately, it is not as mechanical a process to derive the necessary set of operations for the software representation of a class from the OSL description of an office's functions, procedures, and steps. An OSL analyst will have to sit down with a competent programmer and decide what operations should be implemented.

Chapter 6
Conclusion

6.1 Summary

This thesis began by claiming that the goal of OA is to improve the realization of business functions. It furthermore claimed that an OA system that would maximally improve the office's performance would necessarily be office-specific. At the other end of the spectrum, current offerings by OA vendors consist of generic task-oriented tools. Vendors can sell a lot of them, since the tasks are found in almost every office, but the tools are quite unsophisticated. Many of them are just electronic analogs of tools that have been around for a long time.

The goal of this thesis was to find a compromise: a cost-effective, mass-marketable solution that came reasonably close to being office-specific. The solution proposed was based on earlier research of offices, principally the work on the Office Specification Language. That work suggested a premise: there are many offices that, while on the surface appear dissimilar, actually perform the same business function. From this premise, the idea of constructing functional support systems emerged.

Before that idea can be considered a viable solution, several questions must be addressed:

- Can office workers, particularly managers, be supported significantly better than they are by conventional tools?
- Are the business functions truly common?

- Can a support system be easily tailored to fit the needs of different industries, organizations, and individuals?
- How do we build them?

Demonstrating commonality in business functions is beyond the scope of a single thesis. This thesis set out to study the remaining issues, in the hopes of getting early feedback on the viability of functional support systems.

6.2 Conclusions

The results of the research were positive. Studying project management offices at Hewlett-Packard revealed two things:

- Substantially more support can be given for office workers in project management, *particularly* managers, than is currently being provided by generic tools.
- Significant differences in managerial style do not alter the core requirements of a project management support system (PMSS).

Even the seemingly unsupportable aspects of project management, such as designing and maintaining a strategy, can be supported. Support does not necessarily mean automate; in the case of designing a strategy, support means organizing, modeling, and documenting. Expert systems technology can be utilized to transcend modeling, to actually examining a budget or a tactical plan, identifying items that might not make sense or contradict policy.

The study also found that a PMSS could have a second role as an assistant to a new manager. It is common at HP for engineers to be promoted to manager without any formal training in their new responsibilities. A PMSS could support a new manager by providing a standard organization for strategies, tactical plans, schedules, etc.

The managers studied were quite different in their style of work. It became clear that even though the core of a PMSS was common, facilities for customizing the environment to suit personal preferences were mandatory. The working style of the managers also dictated some of the user interface, specifically, that users must be able to hop from one part of the system to another at will. Subsystems that were difficult to hop in and out of would not do.

The next question to be answered was "how does one build such a support system?" The methodology and desktop architecture presented in chapter 5 addressed that question. It is the first attempt to link OSL results directly with software, and it is not a complete link. The methodology serves well to indicate the flavors (classes) of objects and the contents of their internal data structure. It does this by relating the objects in the OSL world to objects in the software world. It is weaker in showing what specific methods (operations) should be implemented, relying more on the software engineer to study the OSL specification and decide what the methods should be. The methodology does not appear to be wrong, just not fully worked out. Although further studies will have to be done before the methodology can be considered effective, the results of this thesis indicate that it will be.

To demonstrate the practicality of the methodology, an electronic desktop was designed. It was created specifically to manipulate objects. A simple desktop manager and primitive flavors mentioned in section 5.1 (text, graphic, and folder flavors) were implemented in a Zetalisp dialect, and several of the strategy classes described in section 4.3 were implemented, primairly by inheriting the primitive flavors. The desktop architecture proved to be very good at meeting the goals discussed in chapter 5: manipulating objects with widely varying ways of presenting themselves, encouraging a common user interface, encouraging data sharing, and providing an environment in which objects can be implemented easily.

A result of this research was an approach to handling customization and reduction of the complexity of building new support systems. A toolkit, consisting of modules that create and manage different kinds of windows, can be used to simplify construction of new classes of objects. The inheritance capabilities of modern object-oriented languages such as Zetalisp make it easy to create a new flavor that is the same as an existing one, except for a new or different method here and there.

6.3 Future research

What was sought in the beginning was a way of bringing the qualities of an office-specific support system to a marketable product. This research revealed that the key to bringing functional OA to the market is the commonality of the business functions. Thus, the chief recommendation for further research is to study all kinds of offices. A starting point might be to take project management as described in the appendix, and move outward. That is, study offices away from Hewlett-Packard, away from software, in smaller firms and in larger firms, in government contractors, and even university professors.

A second area that needs further work is the methodology for constructing a support system. The methodology described here only helps organize the software, relating OSL objects to software flavors, and indicating the flavor's data structure. The next step is to proceduralize the derivation of the methods of each flavor. The ultimate goal is to define a language that not only is suitable for analysis and description purposes, as OSL is, but also directly executable by a computer.

Appendix A

Detailed Study of Project Management

A.1 Introduction

The participants in this study were three first-level managers, two second-level managers, the lab manager, and a secretary. They were chosen because, while they all worked in the same area organizationally, their styles of management varied considerably. This study should be considered an example of a project management office, and not the definition of one. At the very least, this description is specific to Hewlett-Packard, which has some specific ideas about roles and responsibilities. Parts of this description will be specific to software project development, and even to small software project development. Furthermore, example projects, strategies, and names of organizational units are fictional, and are included for illustrative purposes only.

A.1.1 Mission

The overall mission of the "project management office" in the R&D lab is to develop a strategy and implement it through the introduction of new products. Each manager at each level of management has a charter, indicating the market segment(s) he is responsible for. A lab manager might have the charter for "HP3000 business applications", a section manager might have it for "HP3000 office applications", and a project manager might have it for "HP3000 office graphics applications." With the charter as a guide, each manager is charged with setting a strategy, describing where his team is trying to go and what projects will be used to get there. Finally, each manager has responsibility for seeing to it that the required projects are begun and completed to specification, on time, and within budget. Lower managers get more into the day-to-day details of projects, while higher managers spend more time on the strategy aspects.

Although the managers studied were consistent in their description of their mission, the metrics they use to measure their success varied. Example measures reported by one or more of the managers studied include:

- Some measure of product success in the marketplace. Usually they compare how well their products are doing against third-party software written for the same HP machine. Simply comparing all of the, say, database packages in the world would not be fair, since they don't run on all machines.
- Feedback from the customers. This can come directly from the customer, as well as indirectly through the marketing and field support areas.

- How attractive this group is to work in. When attrition is low, and there are lots of inquiries (from inside and outside HP) for job openings, it generally reflects well on the manager.
- Direct and indirect appraisals by peer and superior managers. A simple "your group is doing great stuff" from another manager means a lot. A manager's own performance evaluations and salary growth also provides feedback.
- A personal, gut feeling that goals for the group are being met.

Both of the section managers and the lab manager put market success as their highest criterion. The project managers, on the other hand, tended to put group attractiveness and feeling that goals were being met first.

A.1.2 Organization

Figure 1 shows the overall corporate organization. The fundamental business unit at HP is the division. HP believes in having many small divisions, with a typical division having around 500 employees. Divisions with related products are organized into groups. Groups, in turn, are organized under a vice president responsible for a broad area, such as computers.

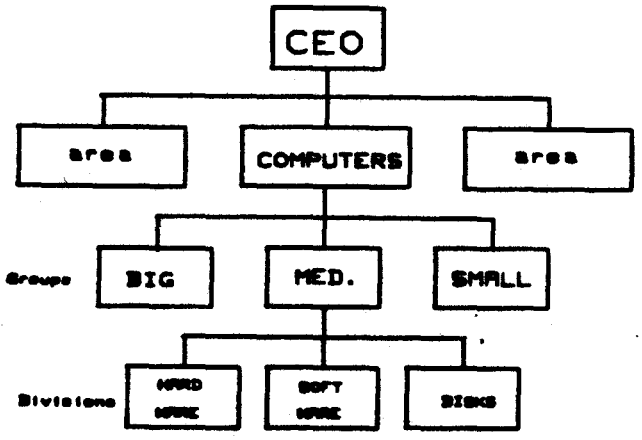

Figure 1

Figure 2 shows the internal organization of a division. The divisions are headed by a division manager. The heads of the major functions in a division (R&D, marketing, etc.) are called functional managers. In R&D, the functional manager is often called the lab manager (or the XXX lab manager, if there is more than one.) In a software division, where there is relatively little manufacturing work to be done, most of the employees work in the R&D lab. Besides R&D, the other functional areas found in most divisions are marketing, manufacturing, finance, personnel, and training. The lab is divided into sections and section managers (SM). An SM has several project managers (PM). Finally, a PM usually has six to eight engineers, or MTSs, reporting

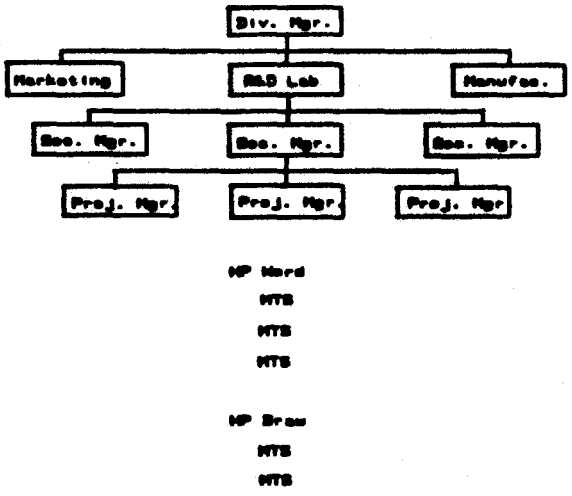

Figure 2

to him. In addition, there is typically one secretary for every two sections.

A PM may be responsible for several projects, or there may be several PMs involved in aspects of a single large project. An entire lab may be devoted to an operating system. The lab in this study tended to have two to three engineers per project, two or three projects per PM, three to four PMs per SM, and usually three to four sections in a lab.

Although HP always follows this organizational structure, it tends to shuffle projects, sections, and even divisions around. Projects can be transferred to a different division, and the project team may or may not be transferred with it. It is not unusual to work for a different division every year or two, without changing bosses or desks.

A.1.3 Environment

HP is a major computer vendor, so it is not surprising to find a computer terminal on almost every professional employee's desk. Most employees have a standard 24x80 character display terminal connected to an HP3000 timesharing minicomputer. The HP3000's are networked throughout the world. There is an electronic mail system that can send simple text messages or complete disk files across the network. Some employees have desktop microcomputers, mainly Z80 based systems running standard microcomputer operation systems such as CP/M. Between the desktops and the HP3000's, virtually every employee has access to the usual range of electronic office tools - word processing, presentation graphics, decision graphics, databases, laser printers, etc. The right package or terminal may not be available right on everyone's desk, but within a set of cubicles, someone will have it. Secretaries are particularly well equipped, with a word processing terminal, a graphics terminal, a plotter, and a letter-quality printer at every secretarial station. (At least part of the justification for this is that they are key alpha test sites for upcoming software products.)

Although the tools exist, they often go unused by the managers. Sometimes, performance is the problem. Several of the packages are slow to load and run, and it is not possible to keep several different packages around as suspended forks. (Note: it is usually not the product's fault that it runs slow, since they are trying to run on machines that have many people compiling on them. This is not usually the case in a real customer's environment.) It is very difficult to leave the word processor to read an electronic mail message and return. Another reason for not using a package is that it doesn't meet the exact needs a manager has. Some managers have a particular way of doing things they are happy with, or a particular form they want the output in. It can be difficult to beat the functionality of paper and pencil. Managers with secretarial support can be reluctant to learn a new software package.

A.1.4 Overview

The work of the project management office can be divided into four main areas. This section will briefly introduce each of them. They will be discussed in detail in the following chapters.

Project management at HP involves more than just managing projects. Managers are expected to design a strategy for their group that is consistent with the overall corporate strategy. They also have to manage their people, helping them set objectives, evaluating how well they meet them, and setting an appropriate salary. They get involved in the recruiting and interview process. Managers perform numerous ad-hoc activities: serving on review committees of other projects, preparing workshops for the benefit of other managers, etc. Finally, and most importantly, managers do take responsibility for the day to day management of their projects – setting schedules, checking progress, coordinating with others – what ever it takes to get a project done.

All of the levels of R&D management can potentially get involved in any of the above areas. Generally, though, lab and section managers concentrate on strategy for an area. Project managers, on the other hand, concentrate on actually managing projects and may even do some of the implementation. The major responsibilities are outlined below.

A.1.4.1 Designing a strategy

Making a strategy involves assessing where the organization is now, deciding where it ought to go, and figuring out how to get there. Ultimately, it should guide the selection of products, and the projects used to create those products. Understanding this requires collecting a large variety of information, most of it unavailable in any documented form. The kinds of information needed include competitive trends, current and coming technologies, budget levels, objectives and other directional information from the rest of the corporation, and current human resources. A major challenge in designing a strategy is to make it fit properly in the overall corporate strategy. Resolving inter-organizational conflicts falls into this area. Most of the time, a strategy evolves slowly in response to changes in the incoming information. Occasionally, such as after a complete reorganization, a strategy is designed from scratch. There is a rough

standard for the information expected to be found in a manager's "strategy folder."

A.1.4.2 Managing projects

At HP, a project manager has complete responsibility for getting a project from the idea stage to manufacturing release. Most projects come directly from the strategy plans, and are usually begun as soon as human resources are available. The "Software Product LifeCycle" describes the official process for getting a project from investigation to release, specifying at each stage what should be done, what documents should be produced, and who has to approve them. Most managers take this as a guide, tailoring it to their particular needs. In their senior engineer role, managers work with the project engineers to design the product – its features, interface, and architecture. As this is an engineering responsibility, it will not be discussed further here. Besides planning the product, a manager has to plan the project: setting schedules, dividing the job up amongst the project team, and so on. While a project is running, a PM must monitor the project's progress against the schedule and report on its progress. There are numerous people in other functional areas that a PM must coordinate activities with - manufacturing, marketing, manuals and training, and other projects (possibly in other divisions, possibly in other countries!) As a project nears completion, the PM's responsibilities as coordinator multiply. Following release, a PM usually retains responsibility for maintaining a product, including bugs reported after release and possible enhancements. At some point, a project will be declared "mature" and no more engineering work will be done on it.

A.1.4.3 Managing people

Managers at all levels have several responsibilities for their employees. First of all, they recruit, interview, and select new employees. To manage existing employees, there are three main procedures that operate more-or-less independently, but are expected to match up in the long run. They set objectives with and produce performance evaluations for their employees. The performance evaluations provide feedback to the employees, and are also used by managers to prepare for a ranking session. Four times a year, managers get together and assign a performance percentile to each engineer. The rankings are based on a direct comparison of the engineers, and the results are expected to fall roughly into a bell curve. The resulting percentile, along with an employee's years-of-experience, yields a "target salary." The actual salary, however, may not match this target salary for a number of reasons. Planning the actual raises over the coming year is the third procedure. Additionally, managers have an ongoing responsibility to the welfare of their employees: to their development, both technically and personally, to their careers, and simply to their well being. Managers stressed the importance of communication with others, particularly with their employees. Finally, employees eventually quit, transfer, retire, get promoted, or are fired.

A.1.4.4 Managing Equipment

Besides a staff, managers must make sure that all of the necessary equipment for a project's completion is available when it is needed. This is usually much less of a concern in software R&D then in, say, VLSI research. Managers acquire equipment,

either directly or through the "engineering services" department, a lab- or division-wide service that purchases equipment, maintains it, and charges managers accordingly. Managers have to keep track of equipment assigned to them, and arrange for any needed maintenance. When they wear out or become technically obsolete, the equipment is scrapped.

A.2 Strategy

A.2.1 Components of a strategy

What is a strategy? There are undoubtedly many interpretations. Very generally, it is a statement of where you are now, where you want to get to, and how you are going to get there. The managers studied, however, have a more precise idea of what a strategy is. Both project and section managers had the same general outline of a strategy. For them, a strategy has the following parts:

- A charter, staking out the market segment the group's efforts will address. A charter must be very clear and specific, since one of its main purposes is to eliminate duplication of effort. Inter-organisational conflicts often center around a charter. For example, "business graphics for the HP3000". The level of the manager dictates the breadth of the charter.
- The group's user, product, and technology focuses. These are an elaboration of the charter. The user focus indicates what kind of customers the group's products will be aimed at. Using the charter above, the user focus might be medium to large businesses, where the HP3000 is likely to be found. The product focus would elaborate on "business graphics". The technology focus would describe the technologies that the products would be based on, such as "pen plotters" or "ink-jet plotters".
- The short and long term goals for the group. These are not project-oriented goals, but goals that apply to the group as a whole. For example, "Be a leader in business graphics by 19xx."
- The tactical plans. This section describes how the group is going to get where it wants to be. It consists of:
  - A "bag of projects." This contains every idea for a project the manager has, whether they are current, running projects or just far out ideas. This folder may exist only in the manager's head, or there may be a folder with some handwritten notes.
  - A long range plan. This shows the projects the group expects to undertake over the next three to five years, along with the targeted completion date. As projects get further into the future, they may be rather vague, without specific staff or other necessary resources allocated. Individual managers have some variations to this basic plan. One keeps several plans, preparing for different future staffing levels. Another includes a rough idea of the projects ROI. Managers also vary in when a project is definite enough to be included on the plan.

— A short range plan. This is usually for one year, showing actual projects and who is working on them.

- Other documents or diagrams, showing how the group's efforts fit in with some larger plans. The business graphics project manager might have a diagram showing how graphics will be integrated with word processing.
- A set of objectives. This is different from the goals described earlier, and usually are for the coming year. The objectives usually refer to specific projects or people, and are clear enough to be measured. For example, "start and finish HPFoo this year."

A.2.2 How the strategy is set

Ordinarily, the strategy evolves slowly over time. In fact, one manager said that if a strategy constantly needs radical changes, then it probably wasn't a very good one to begin with. Sometimes, though, a strategy needs to be created more-or-less from scratch. A reorganization, where parts of divisions are broken off and rearranged, is the main reason for creating a fresh strategy. When this happens, each level of management will have to create each of the components of the strategy listed above, and they will generally complete them in that order. The process is often iterative, particularly for the shorter-term plans. A manager has to balance what he would like to do with the resources available to him.

HP is a "Management by Objectives" company. There is a top-down refinement of objectives from the CEO on down to the project manager. At each level, managers take the objectives given to them, decide how they apply to their group, and then set their own objectives. This philosophy continues down to the lowest levels – a manager sets objectives, and the implementor is free to find a way to meet them. Although objectives flow downward through the organization, engineers and lower-level managers are encouraged to participate in the setting of the higher-level objectives. A project may be started because it fulfills an objective sent down from above, or an engineer may get an idea for a project and try to push its approval upwards.

The specific sources of information used by a manager to set strategy include:

- Objectives sent down from above. At the corporate level, this usually consists of a list of "hot spots" that need pursuing in the next year or so. They become more specific – actions that will be taken, measures (with numbers) to be achieved. These objectives are usually received in written form, possibly a copy of a slide (an overhead transparency).
- Information sent up from subordinate managers and engineers. Often an engineer will want to investigate something as a tentative project. A manager usually gets this kind of information through casual conversation. Occasionally, an engineer or manager will prepare a paper on a subject for wider distribution.
- Customer input. HP prides itself on being responsive to customer needs. Customer's provide both short-term information on specific HP products

and longer-term information on customer needs. Some of this information comes indirectly through the field support teams (located in the plant). The information can also be gathered directly from the customer. HP often has customers visit the plant, and a project or section manager might attend a customer presentation. Sometimes, HP goes to the customers plant to give the presentation.

- Current environment. The two main components are technology trends and competitive information. A manager needs to know what technologies are available now, and what new technologies will be available in the future. Competitive information can be necessary for several reasons. When the objectives call for a significant advance in the technology, simply doing what everyone else is doing is not good enough. When a fast, me-too product is called for, pricing information might be important. There are a variety of sources of information for both technology and competitive trends used by the managers:
  - Conferences and trade shows.
  - Journals, such as ACM Communications, and trade papers, such as Computerworld.
  - Informal communication with people. Most managers have a network of people that keep them up to date on specific subjects. Section managers, who do not have time to attend shows and read all of the journals in their area, rely on this means of gathering information. These people are not necessarily the employee of the manager, and might not even work for HP. PMs, while taking more of a responsibility to keep personally up to date, also make use of a network of people to gather information.
  - Marketing department. This is, surprisingly, a very minor source of information for R&D people on the current environment. Some managers make no use of it at all. Others will officially ask the department to analyze something (as opposed to informally asking a key person in the department what she thinks.) While the department does seem to have some of the needed information, there is a general reluctance on the part of the lab to seek it out. One reason is that, at least at HP, it is the R&D staff, not the marketing department, that is expected to keep up to date with competitive and technological trends.
- Constraints. In addition to the information gathered above, several types of constraints factor in to the strategy setting process. The main constraint is staff size. The principal resource in software development is people. Computers, terminals, and disk space are important but are usually not the limiting factor. Wages are the largest part of a software manager's budget, so the budget is the main constraint on the short-term plans. Other constraints include resources in other departments, such as manual writers and marketing support.

Each project in the strategy plans may be analyzed in several different ways. The traditional way to judge a project is by its potential Return On Investment (ROI.) It

may also be judged for the contribution to the market position it will give to a larger product, such as a large computer, or for its opening of new markets. Some managers make estimates of these and include them in the strategy plans. Overall, though, HP is not a "numbers" company.

The managers studied differ on how to view constraints. One manager doesn't really believe in a budget constraint. He makes plans assuming he has "enough" people, and if he doesn't get all he needs, it just takes longer to meet the goals. The goals themselves do not change. Another manager prepares several tactical plans, allowing for different staffing levels.

The managers also differ on how they work on strategy. Most of them think about it all the time. Where they differ is on how they actually sit down and work on it. Some can clear the desk, pull the "strategy" folder out, and begin work on strategy for a while. Others, usually SMs, claim that they actually work on strategy continuously. This cannot be literally true, since SMs have other definite responsibilities. One possible model is that designing and maintaining a strategy is the primary responsibility of an SM, and that everything else is an interruption in an otherwise continuous process.

A.2.3 Maintaining the strategy

Once a strategy is in place, the job is not finished. A strategy is constantly evolving. There is a distinction between updating the current plans in the manager's mind, and updating the slides or a paper to reflect the changes the manager has been thinking about. Typical triggers to conceptually change a strategy include:

- A competitive development. A vendor might announce a product which directly competes with a manager's own currently planned product, requiring a change in the features, timing, or price of the manager's product.
- A new technology arises. If, for example, the $99 multi-color high-resolution ink-jet plotters became a reality, the business graphics manager might have to revise his strategy.
- A reorganization. Since there is a downward flow of objectives, a manager suddenly in a new organization might have to modify his strategy.

Any one of these events might cause a manager to begin thinking about revising his strategy, but so long as he doesn't have to tell anyone about it, the plans can stay in his head or on scrap paper. When a manager has to do a presentation, however, the accumulated changes since the last time a presentation was made have to be put down formally, in slides or possibly into a paper. Almost all of the time, the resulting strategy is communicated through a presentation, or a one-on-one meeting. Once a year, SMs give their plans in writing to their boss, for incorporation into a division presentation. Some managers respond to individual inquiries with a quick HPMail message.

Some managers keep ideas in their head or on scrap paper until an upcoming presentation forces them to get the ideas incorporated into the formal strategy. Others

update their slides and papers as the ideas come in, and do nothing special at presentation time.

A.2.4 Budgeting

The budgeting process is almost separate from the rest of the activities a manager performs. There is, however, a link between the budget and the strategy. Basically, the budget constrains the short term goals of a group, and the goals of a group can affect the next round of budgeting.

Budgeting is a yearly process. The marketing departments throughout HP go through a quota and forecast setting process, which results in an estimate of how much HP will make next year. A standard percentage of this is assigned at the corporate level as expense. This expense represents the corporate budget, and is divided up amongst the groups and divisions. At the divisional level, it is split up amongst the functional areas. The R&D manager allocates it to the sections. There is no attempt to formally budget to the project or project manager level.

The downward phase of the budget setting process gives each manager a target – a total dollar figure for his branch of the organizational chart. When it hits the section level, SMs prepare a detailed budget that should add up to the target. Technically, budgets are divided up into location codes, and usually a section manager has a single location code. The SM has to allocate his money to categories such as payroll, travel, and floorspace charge. The line-items are fed into the accounting system's grand computer. The accounting system combines the SMs' detailed budgets into a single R&D budget, and the lab manager may make some changes. Likewise, the system combines the functional area's budgets, and gives a report to the division manager, who may make changes. Although project managers do not have a budget of their own to manage, they usually plan with their SM for expenses.

In a software lab, the largest expense is for people.. This is the main link between the budget and the rest of the strategy: the number of people that are available. The only other major variable expense is the capital budget. All major equipment purchases, roughly those over $1000, must be planned for on the capital budget. Like the main budget, there is a corporate wide fund for capital purchases which is distributed to groups and divisions. The fund is treated special for accounting and tax purposes. Each PM prepares a list of equipment he would like to buy, and gives it to his SM. The SMs, in turn, combine these lists and prepares the capital budget, with each piece of equipment appearing as a line-item. The proposal goes through a series of revisions, until it is accepted by upper management.

The relationship between the capital budget and the main budget is through a depreciation entry on the main budget. While the equipment is usually paid for in cash at the time of purchase, it is written off on the main budget over its estimated lifetime. There is a table with standard lifetimes for products. To prepare the depreciation entry for the main budget, each piece of equipment is checked to see if it is fully depreciated yet. If not, the coming year's depreciation is added in to the total depreciation for the

section, and this becomes the entry. The accounting department has a computerized database of all capital equipment owned and its current depreciation state, and supplies each manager with the necessary information.

There are actually two ways capital equipment can be owned. A separate department, called engineering services, can "own" the equipment. It figures out the depreciation, plus associated salaries for operators, equipment maintenance, etc., and comes up with a total figure. The figure is divided up amongst the participating sections. This is the way most large equipment is handled. The other way is for a section to "own" it directly, taking care of the depreciation, maintenance, etc. charges itself.

Most managers use a spreadsheet program to assist the preparation of the main budget. Each manager has his own model. The accounting department supplies a handbook, giving standard figures or percentages for the various line entries. Most of the expenses can be estimated satisfactorily by taking a constant times the number of people in the section. This works well for office supplies and telephone charges. Some managers are more detailed in their estimates of entries such as salaries (which make up most of the budget.) They may input the actual salaries with planned raises for the year.

Each month, a report is sent from accounting to SMs showing actual and target expenses in recent months. There is usually nothing surprising on it, since everything was planned for in advance. If there was something on it the manager didn't understand, he would call accounting and find out what it was. Accounting finds out about actual expenses from all over HP. Office supplies, payroll, purchasing, and so on all send information to accounting detailing actual expenses. Expenses are actually charged to a specific project number within a location code, making it possible to get reports detailing the expense of a particular project, but managers do not seem interested in that level of detail.

Once a budget is set, there usually are not any changes in it. The major reason for any change in the budget is a significant change in actual sales from the forecast. The rule is, "in good times, manage to targets; in bad times, manage to your allowed percentage of sales." If sales are only 80% of the forecasted sales, then a manager should try to spend only 80% of the targeted expenses. This is difficult when the major expenses are for salaries, but planned additional hiring or equipment purchases can be delayed. If the companies actual performance is significantly different from the forecast, a new budget may be prepared. A PM may have to change his tactics when something budgeted for is not approved at purchase time.

A.2.5 Preparation of the strategy documents

Managers are frequently asked to give presentations of their strategy. Since at HP, presentations are always accompanied by overhead slides, it is no wonder that the standard media for keeping the components of the strategy is an overhead slide. The usual arrangement is for a manager to sketch what the slide should look like on paper, and then a secretary will use a computer program and plotter to produce the slide.

Managers give presentations for several different audiences - customers, upper management, and peers and engineers in the R&D lab. Sometimes, while the basic ideas have not changed, a new slide must be prepared to address a particular audience. Some managers have hundreds of slides, enough to make it difficult to find the slide needed for a particular presentation. They are usually kept in 3-ring binders, sorted by subject (strategy, specific projects.)

To make the slide, a drawing program is used to design the slide, and a computer plotter draws the finished slide on transparencies. The drawing program program is fairly sophisticated, allowing multiple fonts, sizes, and colors, as well as lines, arcs, and figures. Some managers take an interest in having very impressive slides, and will go so far as to design group logos for inclusion on the slide. (Perhaps being affiliated with graphics products has something to do with this.)

Sometimes, a written version of the strategy is required. A manager will usually write out by hand the text of the document, and his secretary will prepare it with a word processor. Some managers type their own documents.

A.2.6 Tools used

Besides slide-making and word processing, there are not many tools available to help a manager prepare a strategy. Some of the project management tools, to be discussed in the next section, can be used to work on strategy. PERT can be used when a number of smaller projects are interrelated. Software estimation models can be used to get an early idea how long a project will take. Spreadsheet programs can be used to help with the budget. However, no managers studied make use of software estimation models or PERT for setting strategy. Perhaps because of the lack of any standard tools, each manager tends to have his own format for presenting such things as the budget or the schedule. One manager has a format that shows projects, engineers, time, total staffing levels, and major product releases on a single chart. He prepares them himself using office graphics packages. Although managers do use spreadsheet programs, they enter by hand a lot of information that already exists in another form, such as depreciation and salaries.

A.3 Managing Projects

The goals of a project manager with regard to any particular project are to produce a quality product, on time, and within budget. The priorities of these goals varies with the project. Generally, budget is not a goal, but a given. Projects are not usually started with a goal of "complete this project for one million dollars", but rather "you've got 4 people, finish it in a year." "Quality" should not be interpreted just as whether or not it works, but also how well, and with what features. All three of these goals can be traded off against each other.

A.3.1 The Software Product LifeCycle

HP has a published guide to getting a project from beginning to end: the Software Product LifeCycle (SPLC). It provides

- A flowchart through the phases of project development
- Descriptions of the documents required along the way
- Assorted information and tips for managing projects, principally in the area of testing.

It is not a how-to guide as much as a set of standards. For example, it indicates that a schedule is needed, and at what phase of the project it should be prepared, but does not contain reference material on creating one. The guide is followed, more or less, by all software projects in all divisions at HP. Each lab sets its own guidelines for the amount of flexibility a project manager has in deviating from the SPLC.

The major phases, according to the SPLC, of a project are:

1. Investigation
2. Design
3. Implementation
4. User testing / Product introduction
5. Release
6. Post release

The SPLC contains a Product Development Checkpoint Completion (PDCC), which contains a checklist of items to be completed at each stage, and a place for each member of the business team to sign off, giving approval for the project to move on to the next stage. Since there is some leeway in the SPLC, some managers use the given PDCC and make changes directly to it to reflect the new lifecycle, while others create their own form.

The SPLC specifies a number of documents that are to be completed at various stages of product development. The SPLC also describes the format they are to take, and the typical contents of each section. Some of these documents, such as the quality plan, are rather straight-forward. They may be copied right out of the SPLC or from the last project. Others, such as the investigation report, are a major piece of creative exposition. They are frequently joint-authored and heavily revised and edited. Typically, each engineer will write part of it, and the PM will assemble and edit it.

A.3.2 Starting a project

Despite the formalism of the SPLC, there can be a fuzzy boundary between a project sitting in the strategy folder, and one being actively investigated. The typical transition is that a project idea has been waiting in the strategy folder for some resources to become available. As they do, some investigation begins. It may continue somewhat "underground" for some time, that is, without calling a lot of attention to it, putting it on the organization chart, creating it in the accounting sense of opening a project number, or starting a PDCC. The problem with going public is that other managers will begin to peck at the project, wanting it to do this or that.

Managers can usually keep a project underground through the investigation phase. By the time it moves into design phase, it should be "officially" started. A project number for the project is opened with the accounting department. A business team for the product is formed, initially consisting of the PM, someone from marketing, and someone from product assurance. A PDCC is started, and signatures from the functional managers in R&D, marketing, and product assurance as well as the division manager are required to move into design phase. A project notebook is created. It contains a copy of every document, memo, progress report, etc. that relates to the project.

Getting a project number and a PDCC started are the official ways a project is begun. PMs consider a project active when it is staffed, regardless of the official state it is in. SMs consider it active when it can be turned over to the PM. An SM normally gives the PM a list of objectives and a timeframe for their completion. Staff size is usually known at this point too.

A.3.3 Scheduling

A schedule indicates what will be done when and by whom. It has measurable milestones in it, so that the project's actual progress can be checked against the schedule. Scheduling is the responsibility of the PM, who works with his engineers to create a schedule. The schedule is normally completed during the investigation phase, or very early in the design phase. The SM approves the schedule, but normally does not need to make changes in it.

A PM faces a number of constraints in preparing the schedule. Market considerations often imposes a rough target date for completing the project. The nature of the project produces the list of tasks that have to be done. Staff size, and the characteristics of the particular engineers involved, impact how fast they can be done. Hardware resources, both quantity and quality, also impacts how quickly progress can be made. The principal constraint on how fast a project can be completed is, usually, the staff size.

The actual form of the schedule as well as the method of creating it varies somewhat by PM. One method involves breaking down the project into small, one to two week tasks. With a known staff size, this can be put together to give the approximate completion date. This is compared with the target completion date, and if a substantial difference exists, the PM and SM will have to renegotiate for a change in either staff size, target date, or product features.

Another manager puts less emphasis on up-front scheduling. He begins with the starting date, and fills in the major milestones described in the SPLC as time windows. As the project moves along, the dates are made more specific. Yet another feels that schedule making is an art, and a different approach is needed for each project.

Two common tools to assist scheduling are software development estimation models and PERT systems. A software estimation model predicts how long a software project

will take, given some measure of how difficult the code is, the development environment, the size of the development team, etc. The models used are based on empirical data from hundreds of previous software projects from the entire industry. They take in to account, in a statistical way, effects such as the "mythical man-month".

Managers are not generally interested in such a system. Some just plain don't believe in them. Others want a more specific set of questions used, such as performance factors for specific engineers on the team. Another is worried that if the model was used to generate completion dates, then a project would take exactly that long, even if it really could have been done faster. Managers indicated they wouldn't mind using it as a "sanity check" on their own schedule, or as a defense when a boss wanted to know why something takes so long ("See, the program says it takes this long.")

PERT is a network scheduling aid. PERT assumes that a complex job can be broken down into smaller, independent tasks, and that the length of these tasks can be estimated. The manager lays out each of the tasks in a graph, showing dependencies and parallelism. Analysis of the graph shows where the critical path is – the path along which if anything slips, the whole project will slip. All of the other paths will have some amount of slack time, and the earliest and latest a project can start or finish. This information can help a manager better allocate resources.

HP currently only has a program that graphs a PERT-chart based on an input file describing activities and their dependencies. It is frequently used as a presentation tool. It is not interactive, nor particularly easy to use. It is capable of producing much of the standard PERT information, including free and slack time for each activity, the critical path, etc., but this tends to be ignored. There was interest in an interactive PERT-chart system similar to Apple's LisaProject.

A.3.4 Monitoring projects

Accurately determining how far along a project is is very difficult. Major milestones, such as "investigator phase completed" or "functional completion" are relatively easy to judge. Knowing when 75% of the code has been written, or when 99% of the bugs have been found, is another matter. Just as manager's differ in their approach to scheduling, they differ in the way they assess a project's progress. If the project schedule is broken down into one or two person/week tasks, and if the schedule is kept up to date, it is possible to make reasonably precise estimates of progress. Other managers just use the major milestones, and use informal means to judge progress in between them. Since the projects in this division tend to be relatively small, this is often sufficient. Participation in "code walk-throughs" is a good way of judging progress during implementation, and actual usage of the resulting product can provide feedback on its readiness for manufacturing release. The quality assurance (QA) department has some automated testing methods and accompanying statistical models for predicting the percentage of bugs found. During the testing phases, as bugs are found, there is often a wall covered with individual bug reports, classified by severity, letting everyone instantly know how many known problems exist in the product. After release, a centralized and computerized bug tracking system provides individual and summary reports on bugs.

Once a project has been scheduled and is running, there are some standard forms filled out by the PM every month to report progress. These forms are passed on up to the section and lab managers for review and possible action. At the start of each month, the PM lists the objectives for the coming month for each of his projects. Objectives should be concrete enough to be evaluated at the end of the month. An example would be "complete coding of the terminal interface module." At the end of the month, what actually was done for each objective is listed. There is also a space for issues (this is what is most carefully read by higher ups.) The final section is for next month's goals, which also become the new goals section for next month's form.

Another form plots graphically the scheduled progress versus actual progress towards key milestones. These are filled out by hand, and are rather tedious. They could easily be derived by computer if the schedule was on-line. The PDCC makes a good checklist of things to do for a PM. An SM expressed interest in being able to see the PDCC on-line for any of his projects.

Ideally, the schedule drives the monthly goals, both for the project and for individuals. If the schedule is up to date, the PM can just read right off the schedule what each engineer should be working on this month. Keeping the schedule up to date is difficult, though, since there is no automatic scheduling facility available.

Project managers also keep the accounting department informed of the project's progress. A form is sent at the beginning, at each change in the project's phase, and at the conclusion of the project. Unfortunately, these phases are not identical to those in the SPLC. Each project has a project number, and each engineer's time is charged to a specific number.PMs do not usually worry much about what account to charge an engineer's time to, but simply use the numbers given by the SM (who is at the lowest level where a budget is set.)

The monthly reports, the schedule, and a copy of all of the documents and correspondence associated with the project are kept in the project notebook.

A.3.5 Typical responsibilities along the SPLC

No two projects are run exactly alike, nor are the problems a manager is faced with during a project ever quite the same. This section describes the major phases of the SPLC, and the types of responsibilities a manager has in each. Most of these duties fall on the PM, but higher levels can get involved, particularly when a project is "hot" throughout the corporation. The major documents created during the lifecycle are also mentioned below. They are often joint-authored by a group of engineers, with the PM editing and combining their work. Most of these documents are heavily revised, even 6 months after first release.

One responsibility that goes on throughout the lifecycle is that of coordinating. Projects have a business team that consists of representatives from all involved functional areas, but the bottom line responsibility for getting things done falls with the PM. He organizes the meetings, gets after departments that are not meeting their

commitments, and whatever else it takes to get a project out the door.

A.3.5.1 Investigation

The biggest challenge a manager faces during investigation is selling the idea. When the idea comes from above, this may be easy. When an idea is being forced up from below, it can be very difficult. An investigation proposal is supposed to be prepared before even beginning investigation. Often a lab product data sheet and some explanatory notes will suffice. The data sheet is a standard 1-page form summarizing the product, its features, and its expected availability. It is frequently revised during the course of the project. The data sheet is often given out to people asking about a project.

During the investigation phase, the PM and the project staff are expected to find out if it makes sense to actually develop the idea into a product. There is usually some competitive analysis performed, possibly with the aid of the marketing department. A prototype may be built. There may be a large amount of communication with people throughout HP to gain insight into the technical and business aspects of the product. On the other hand, the group may keep to themselves, to avoid attracting attention to the project.

The conclusion of the investigation phase is the preparation of the investigation report. This document is fairly important, and also the least mechanical. It usually involves a large creative effort. There are also several presentations about this time to a variety of audience types. The major decision as to whether or not the project will continue is made at this point.

A.3.5.2 Design, implementation, and testing phases.

Products are designed from the outside in. First, the functionality of the product is worked out, and the user interface (if it is an end-user product) or programatic interface (if it is for use by other programs) is designed. They are documented in the external specifications. This document usually goes through several revisions. It is important because it is used by other departments, such as manual writers, who need to work with the project team. A basic outline is given in the SPLC.

Following the external design, the internal design is begun. This is supposed to be before any actual implementation is begun. However, there can be some overlap between design and implementation. During and after the actual implementation, the internal maintenance specification is written to guide others who may have to work on the system in the future. There is a standard format for this document, and it is usually done by the engineers. A manager, besides any implementation responsibilities, is usually involved with a series of review meetings. He may also give presentations. The minutes of the review meetings are usually typed up and distributed, with a copy going to the project notebook.

Products are tested in a variety of ways. Alpha and Beta testing involves giving a pre-release version of the product to model users inside and outside HP, to get some feedback on the products design and to catch bugs. Products are also tested in the lab

using some packages that exercise the software.

A.3.5.3 Finishing a project

Preparing a product for manufacturing release (MR) is a frenzy of activities. The PM's chief problem is coordinating the activities of people in different departments, often in different buildings, possibly in different states, and sometimes in different countries. There is sometimes a specific deadline to meet. There are a series of meetings to discuss last-minute problems. Signatures approving the product's MR have to be obtained from people who aren't easily tracked down. Since the end result is usually just a magnetic tape, interfacing to manufacturing is relatively simple. A form is sent to manufacturing, along with magnetic tapes containing the product.

The secretaries help out in scheduling meetings and getting things signed. Since HP uses modular cubicles, most meetings require reserving a conference room. The secretary calls the person in charge of room reservations to get a room as close to the date and time desired as possible. When the exact date and time are known, the secretary will often send an HPMail message to those attending to let them know. They are expected to RSVP, and if they don't, the secretary will have to call them up.

People often give documents to their secretary, asking that she get something signed. One secretary keeps a log book of every such document. In it, she records what it is, who gave it to her, and who has to sign it. Then she delivers the document to whoever has to sign it, and waits. If someone comes back a week later and wants to know what happened, the logbook identifies the guilty manager.

Towards the end of a project, one manager uses a list-keeper program to remember individual tasks that have to be done, who they've been assigned to, and their importance. Daily to-do lists are printed out, and at the end the PM can look back over who did what.

When a project is finished (MR'd), the records of the project are gathered into a box and held onto for an indefinite length of time. A PM may have responsibility for fixing bugs after release. HP has a centralized tracking system for software problems, called STARS. Reports of possible bugs from both inside and outside HP are forwarded to the STARS office, which enters the problem into the database. STARS keeps track of the bug report throughout its processing. It sends reports to the appropriate people, and also generates statistical information on the number of bugs in a product or product line. A manger or engineer can query the STARS system to find the state of any bug, or get any other type of analysis

A.4 Managing People

A.4.1 Recruiting and hiring

Project plans indicate when a larger staff is needed. New people must be budgeted for, by allowing for their salary. When the group is ready to hire someone, an employee requisition is filled out. This form describes the job, the required and desired

qualifications for this job, and the salary range. The requisition is approved by the lab manager, and then it goes to the personnel office.

Having money in the budget is a major limitation on hiring, but not the only one. Even if a manager had the financial resources, has to have enough interesting work to do, now and in the future. Sometimes, even when the need is great and the money is available, new people cannot be hired because the manager cannot manage any more people, or the group may be saturated with recent hires. New people take time away from the rest of the group's work, as they are assimilated into the project and brought up to speed.

There are many sources for new people. Which ones to use depend on what kind of person is sought. For example, sometimes only a transfer from another part of HP is acceptable. Other times, a recent college graduate with a BSCS will do. The sources include:

- College recruiting lists: HP has a computerized corporate-wide database of people interviewed during college recruiting trips. A manager may participate on one of these trips, but he goes on behalf of all of HP, not just for himself.
- Personal contacts at universities: A manager can call an old friend and ask if he has anyone to recommend.
- People in other parts of HP: The personnel offices throughout HP regularly pool their information on outstanding employee requisitions. A master list of open requisitions is made and posted at each division. Employees may peruse these lists and contact the manager about the job. One manager warns of a danger when employees transfer: getting the former boss mad. He makes sure an inquiring employee has told his present boss about the inquiry before talking with him.
- People answering want-ads: Occasionally, HP places newspaper ads when more experienced people are desired. The manager negotiates with his personnel department for the placing of an ad. Personnel, in turn, may get together with other personnel offices to place one big ad. Personnel keeps a file of those answering those ads.
- Direct inquiries from people.

The amount of support provided by personnel depends on the source of the candidate. For example, personnel needs to know when any contact is made with a college student, so the computer database can be appropriately updated. From some sources, the personnel office performs the initial screening. In general, personnel handles the paperwork for the applicants, and takes care of various administrative details such as travel arrangements.

When an interesting lead is found, the next step is usually a phone conversation. If the manager is satisfied, the candidate is invited for a plant visit. The manager is responsible for planning the candidate's day and playing host. The day usually

consists of the candidate meeting with a series of engineers and managers in the area. Sometimes, two managers will cooperate, sharing a visit.

Afterward, the interviewing team usually meets to decide right then on whether or not to make an offer. The salary offer is based on the number of years of relevant experience, determined by rules in the policies and procedures manual, and an estimate of his future ranking (see below).

A.4.2 Performance evaluations

The evaluation process begins with the setting of objectives each month. Although not formally required by HP policy, most managers work with their employees to set objectives for the month. One common form is the same one used to report monthly project status. The first section, current objectives, are set at the beginning of the month. At the end of the month, the next section is used to show what actually happened. The third section is for discussing special issues, and the last is for next month's objectives, which become current objectives on next month's form. Other managers use a different form, or no form at all, but they still capture the same ideas.

One input to the objective setting process is the employee's project schedule. It should show just what each employee should be working on at any given time. Of course, this presumes that the schedule is kept up to date. Previous monthly objectives are also used. One manager uses his calendar to note special things that need to be done. The forms are kept by the manager.

Most managers also prepare quarterly evaluations. Like the monthly objectives, these are not required by corporate policy, and each manager can use whatever form he likes. Sometimes a lab manager will specify a particular form be used, to facilitate the quarterly ranking sessions (explained below.) These forms summarize an employee's performance in several categories, resulting in a two or three page document. A manager will usually prepare in advance of the meeting with the employee, with handwritten notes of what will go into the final evaluation. Previous quarterly evaluations and the most recent monthly objectives are the basic input to the process. Those objective sheets should indicate what the employee has been doing, and how well he has been meeting objectives. The forms are not part off the personnel office's formal records, but are only for the manager's and employee's use.

SMs evaluate their PMs in much the same way as PMs evaluate their employees. They usually use a monthly objective and quarterly evaluation system similar to those used with engineers. The criterion, however, are different. For example, an SM may take the project status reports into consideration in a PM's evaluation.

Corporate policy dictates that each employee is to have a formal, written evaluation six months from hire, and every year after that. There is a standard form used for the purpose. It is several pages long, and has space for written comments as well as "good..fair..poor" checkoffs. These forms are important, and a manager will spend a fair amount of time planning what will written on them. The previous quarterly

evaluations and the previous yearly evaluation are the typical information sources. The manager usually prepares the form in pencil, and then meets with the employee to discuss it. Afterward, the form is typed by the secretary, using a template for the word processing program. It is signed by both the manager and the employee, and sent to the next higher manager for review. From there, it goes to personnel and joins the employee's permanent file.

A.4.3 Rankings

Every quarter, all of the employees in the lab are ranked against each other, resulting in a percentile score. First, each PM ranks his employees. The quarterly evaluation prescribed by at least one lab manager gives criterion and a weighting for that criterion. If the PM gives the employee a score from one to nine, and multiplies them by their weightings, the result can be used as a percentile. Thus, there is some overlap between the quarterly evaluations and the ranking process. The two do not coincide, however, but rather the rankings tend to fall in the middle of the quarter, while the reviews fall on the calendar quarters. The most recent quarterly evaluation feeds the ranking process. The mapping is informal, however. A manager needs to rank his employees, and whatever method works if fine.

An SM will meet with all of his project managers, and merge the rankings made by the PM. Finally, the SMs and PMs all meet with the lab manager to do a lab-wide merge. At this point, the percentiles are expected to fit into a bell curve. If no manager ranked an employee at 10%, then during the lab meeting, someone will be shifted down there. During these lab meetings, some lab managers resolve conflicts by looking directly at the scores assigned to the quarterly evaluations. Managers usually bring supporting data to the ranking sessions, such as recent objectives and quarterly evaluations of his employees. One manager goes through this material before the session and prepares a summary of each employee's accomplishments. The results of the ranking session is fed into the personnel office's computerized salary system. In a separate but similar process, all of the PMs are ranked by the SMs and lab manager.

A.4.4 Salary administration

Salary at HP is theoretically determined by taking two numbers, the percentile rank and the number of years of experience, and looking on the wage curve for the desired profession. The salary can be read directly off the graph. In actuality, the actual salary paid is usually different from the target salary for a number of reasons.

The personnel office has a computer system which knows about every employee, their current ranking, their years of experience, and their recent history of raises. It also has all of the wage curves stored inside it, so it can produce the target salary automatically. The manager's job is to plan a series of wage increases over the year so that the actual wage meets the target wage. The pay curves are issued annually, so a manager is supposed to meet the target salary by the expiration date of the wage curve. Increases are only allowed quarterly.

The differences between target and actual salary usually arise because of a lag between performance change and salary change. It is HP's philosophy to pay for consistent past performance, so a sudden turnaround in ranking, maintained for several quarters, will see the salary slowly rise to meet the new ranking. A promotion usually results in a lag as well. HP does not give large lump sum raises.

Each manager is given a printout showing all of his employees and the above information. A manager takes into account how long it has been since the last raise, the size of the raises needed over the year to meet the target salary, and knowledge of upcoming events to plan the raises. For example, a manager might wait a quarter to give a raise, so that it matched up with a project's completion.

A.4.5 Payroll procedures

There are two basic forms that relate to getting paid. One is the timecard. For professional employees, they are filled in by hand and signed by the supervisor. Ordinarily, the secretary keeps all of the timecards until the end of the period, fills them in, and asks people to sign them. They go to payroll after being signed. The other form is for charging an employees time to a particular project number or numbers. These are also kept by the secretary, who fills them out the same way each week unless instructed otherwise. They have to be signed, and then they go to accounting.

A.5 Managing Equipment

Managers must make sure that projects have the necessary physical resources in addition to the so-called human resources described above. It is possible to combine these two areas into a single "resource management" area, since at a very high level both involve acquiring, managing, and ultimately disposing resources needed to complete projects. However, the mechanisms, constraints, and information sources used to manage equipment are sufficiently different from those used to manage people that it warrants a separate section.

Before equipment can be purchased, it must be planned for on the capital budget (described in the strategy section.) If an item is not on the capital budget, the same process is followed. It is just much more difficult to get approval. Assuming it is on the budget, it will have a capital budget number. A purchase requisition is prepared, describing the equipment, its estimated price, and perhaps a vendor. Major purchases require signatures by the SM, the lab manager, and the division manager. The requisition goes to accounting, to verify the capital budget number, and then to purchasing. When it arrives, it is assigned an asset number by the maintenance department.

Managers are on their own for the care of their equipment. Their main responsibility is to not lose it. One manager uses a list-keeper program to keep track of all of the equipment assigned to him. For each item, he records the engineer it was assigned to, the HP asset number, and the serial number. Managers either repair their own equipment or have the electronic maintenance department fix it. Equipment is kept until it is broken beyond repair, or technically obsolete.

A.6 Personal management

This is, of course, going to vary somewhat by the individual. It will also vary by the level of secretarial support. There is one secretary for every two sections. Lab and division managers have their own secretaries. PMs can rely on their sectional secretary to do some support work. Engineers are expected to handle most of their own affairs.

A.6.1 Calendar

Managers at all levels spend much of their time away from their desk. PMs often keep their calendar in a small, portable book and take it everywhere. Since the secretaries do not handle the calendar for PMs, there is only one copy needed. SMs and higher managers, however, generally have their secretary manage their calendar. One SM still keeps a portable calendar as well as a large desk calendar which remains with the secretary. They have a protocol worked out to avoid the obvious problem of having two independent copies of the calendar. Some managers use calendars as a reminder facility for items in the future.

There is a simple calendar facility built into the mail system. One manager does use it for recurring events, such as a business team meeting on the first tuesday of every month. He still uses a regular desk calendar for ad-hoc events.

A.6.2 Correspondence

Material is usually received through the manager's in-basket, containing material from HP interdepartmental and US mail. The other main source is HPMail, the electronic mail system. The managers retain all non-junk mail in their filing system, either by the project it relates to or to a general file.

When an SM originates some correspondence, he will usually write it out by hand, and give it to his secretary to be typed up. Occasionally, he may use HPMail directly to send a quick message. The secretary may type it on a regular typewriter, or use a word processor. The word processing facility is capable of "drawing" the letterhead when the output is sent to a laser printer. The secretary keeps a copy of everything she creates. A PM will usually create his own memo using his favorite text editor, or HPMail.

A.6.3 To-do lists, ticklers

To-do lists are common, although each uses them in a slightly different way. One keeps one per day, and will have three or four days worth of lists on the desk at a time. Anything further out then that goes into the calendar. Another keeps one list, good for the next four or five days. The managers liked the idea of a tickler, but had not managed to get one set up yet.

A.6.4 Personal filing

This is very dependent on the individual. One manager has a multi-level filing system. Things on the desk are important, in progress items. Next come active file folders, containing information on projects on going but not currently being worked on, such as task forces. These might be in a desk organizer on the side. Databases, such as accounting codes and telephone lists are also kept out in easy reach. Finally, there are currently in-active files, kept in filing cabinets.

The types of files likely to be found include:

- Files for each employee. A PM has two files per employee, one that is more-or-less public, and one requiring censoring (evaluations, etc.)
- A personal file, containing to-do lists, personal evaluations.
- Wage information
- Files for each project. They may contain extra copies of documents in the project notebook, and out of date material relating to the project.
- Competitive information.
- Recruiting information
- Archives
- A catch-all file for those not fitting elsewhere.

References

[Brac83] Brachman, Ronald J. et. al. What Are Expert Systems? In Building Expert Systems, Frederick Hayes-Roth et. al. editors. Addison-Wesley, Reading, Ma. 1983.

[Gold83] Golberg, Adele and David Robson. Smalltalk-80: The Language and its Implementation. Addison-Wesley, Reading, Ma, 1983.

[Hamm79] Hammer, Michael and Michael Zisman. Design and Implementation of Office Information Systems, New York University Graduate School of Business Administration, May, 1979, pp. 13-24.

[Hamm80] Hammer, Michael M. and Marvin A. Sirbu. What is Office Automation? Proceedings of the National Computer Conference Office Automation Confrence, AFIPS, March, 1980, pp. 37-49.

[Hamm82] Hammer, Michael and Jay Kunin. Productivity. *Computerworld OA 16(13A)* (March 31, 1982).

[Kuni82] Kunin, Jay S. Analysis and Specification of Office Procedures. Technical Report 275, Massachusetts Institute of Technology Laboratory for Computer Science, February, 1982.

[Mint76] Mintzberg, Henry. Planning on the left side and Managing on the right. *Harbard Business Review,* July-Aug. 1976.

[Murp83] Murphy, John. Office Automation. *Mini-Micro Systems 26(14),* Dec. 1983, pg. 221.

[Rose81] Rosenau, Milton D., Jr. Successful Project Management. Lifetime Learning Publications, Belmont, Ca., 1981.

[Sher82] Sherman, Philip M. Strategic Planning for Technology Industries. Addison-Wesley, Reading, Ma, 1982.

[Sirb83] Sirbu, Marvin A. Jr. et. al. Office Analysis: Methodology and Case Studies. Tecnical Report 289, Massachusetts Institute of Technology Laboratory for Computer Science, March 1983.

[Suth83] Sutherland, Juliet. An Office Analysis and Diagnosis Methodology. Technical Report 290, Massachusetts Institute of Technology Laboratory for Computer Sci-

ence, March 1983.

[Thie82] Thierauf, Robert J. Decision Support Systems for Effective Planning and Controlling. Prentice-Hall, Englewood Cliffs, N.J., 1982.

[Wate83] Waterman, Donald A. and Frederick Hayes-Roth. An Investigation of Tools for Building Expert Systems. In Building Expert Systems, Frederick Hayes-Roth et. al. editors. Addison-Wesley, Reading, Ma. 1983.

[Wein81] Weinreb, Daniel and David Moon. Lisp Machine Manual. Massachusetts Institute of Technology Artificial Inteligence Laboratory, 1981.

[Wies77] Wiest, Gerome and Levy Ferdinand. A Management Guide to PERT/CPM. Prentice-Hall, Englewood Cliffs, N.J. 1977.