Evolving Mature Product Platforms in Discontinuous Markets: An Analysis of IBM's Mainframe Software Business

by

Carroll E. Fulkerson, Jr.

B.S. Computer Science
University of Kentucky, 1987

Submitted to the Department of Aeronautics and Astronautics
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in System Design and Management

at the
Massachusetts Institute of Technology
June 1997

Signature of Author.....
Department of Aeronautics and Astronautics
May 9, 1997

Certified by.....
Steven D. Eppinger
Associate Professor of Management
Thesis Supervisor

Accepted by.....
Edward F. Crawley
Co-Director, System Design and Management Program

Accepted by.....
Jaime Peraire
Associate Professor, Chair, Graduate Office
Department of Aeronautics and Astronautics

Evolving Mature Product Platforms in Discontinuous Markets: An Analysis of IBM's Mainframe Software Business

by

Carroll E. Fulkerson, Jr.

Submitted to the Department of Aeronautics and Astronautics on May 9, 1997 in Partial Fulfillment of the Requirements for the Degree of Master of Science

ABSTRACT

IBM's System/390 (aka "Mainframe") computing business is a $20 billion dollar per year entity that provides the computing services for the mission critical business applications of Global 4000 corporations. Originally developed in the 1960s, the System/390 architecture has evolved and been extended over time in support of multiple product families and information technology paradigms. The latest technology paradigm, "network computing" , calls for seamless worldwide interoperability among heterogeneous computing systems. Driven by widespread acceptance of the internet and world wide web, this movement represents a convergence of disjoint technologies, development cultures, and consumer markets.

This thesis examines the challenges of evolving a mature product platform within frequently discontinuous environments through a case study of IBM's mainframe computing business. First, an overview of the evolution of information technology and the IT Industry is provided. The corresponding impact on competitive models is discussed. Next, the IBM S/390 Product family and development process is examined. Architectural and organizational implications are analyzed within the context of classic product innovation models and within the context of a platform innovation model that is formulated within the analysis. Finally, suggestions are made for competing within the new environment.

Thesis Supervisor: Steven D. Eppinger
Title: Associate Professor of Management

ACKNOWLEDGMENTS

I would like to acknowledge that my participation in MIT's System Design and Management Program and in formulating this thesis has been a trying, yet fulfilling, experience. I have learned a lot during the process and will surely benefit from this experience in the future. Many factors have contributed to this being a positive experience. In particular, I would like to acknowledge the following:

IBM Corporation. I would like to thank you for sponsoring me in this program. This sponsorship verifies your continued commitment to the individual. I hope that I am able to justify this investment via future contributions. In addition, I would like to thank my managers throughout this period: Tom Rozmus, Terri Virnig, Mary Blue. Your support was fundamental to me completing this program.

Peggy McKeon. Thank you for taking the initiative to insure that IBM and I were able to participate in the pilot program. I and those that will enter this program after me owe you a great deal. Thanks, also to Dr. Phil Summers, IBM Research, for facilitating this process.

To the students within the initial SDM Pilot program. Thank you for an intense and satisfying experience. I have learned lessons from each of you and value the friendships that have been established.

To the SDM Staff, Margee Best, Marcia Chapman, Ed Crawley, and Tom Magnanti. Thank you for your support throughout this entire experience. I expected a first class program prior to joining SDM and feel upon completior that this has been achieved. I wish each of you and the program continued success.

Steven Eppinger, Professor of Management and my thesis advisor. Thank you for your consultation and flexibility throughout this effort. Your insights on Product Development and Architecture have proved to broaden my awareness of these topics. I am sure I directly benefit from this thesis in the future. Thank you.

Mike Swanson, IBM System/390 Chief Software Architect and company sponsor. Thank you for all the help in preparing this thesis and for the guidance you have provided me throughout my career. You exemplify what this program is attempting to build.

To my daughter, Kaitlyn (15 months old as of this writing). While I value the SDM experience, it pales in comparison to the other event during this period - your birth. Thank you for reminding me each day (through your actions) of the important things in life.

Last but not least, to my wife Kelly. I do not know how I will ever begin to re-pay you for your support not only throughout this process but in all that I do. I am privileged to be your partner in life and look forward to continuing this experience. Thank you so very much. I love you.

- Carroll E. (Sonny) Fulkerson
Author

TABLE OF CONTENTS

ACKNOWLEDGMENTS 3

TABLE OF CONTENTS 4

LIST OF FIGURES 5

CHAPTER 1. INTRODUCTION 6

CHAPTER 2. PRODUCT PLATFORMS - BACKGROUND AND LITERATURE REVIEW 9

CHAPTER 3. SOFTWARE SYSTEMS AND THE IT INDUSTRY 14

SOFTWARE SYSTEMS AND ARCHITECTURE 14
SHIFTING FOCUS WITHIN THE I/T INDUSTRY 27
COMPETING WITHIN THE INFORMATION TECHNOLOGY INDUSTRY 33

CHAPTER 4. CASE STUDY: IBM'S MAINFRAME SOFTWARE BUSINESS 36

HARDWARE PLATFORM PUSH ERA 39
SOFTWARE PLATFORM PUSH 50
SOFTWARE SOLUTION PUSH 58
SOFTWARE ADAPT, SHAPE, AND PARTNER 63
SUMMARY OF CASE DATA 64

CHAPTER 5. A MODEL FOR EVOLVING MATURE PRODUCT PLATFORMS 67

PLATFORM ENHANCEMENT MODELS 68
INHIBITING TECHNOLOGY FUSION -- THE EFFECTS OF LEVELING 77
OVERCOMING INHIBITORS -- A MODEL FOR PLATFORM EVOLUTION 83

CHAPTER 6. CONCLUSIONS AND AREAS FOR FURTHER RESEARCH 87

BIBLIOGRAPHY 91

LIST OF FIGURES

FIGURE 1. PRODUCT PLATFORMS PROVIDE THE BASIS FOR MULTIPLE DERIVATIVE PRODUCTS..... 10
FIGURE 2. EACH USER ACTION CAN INITIATE A COMPLEX SET OF INTERACTIONS AMONG THE COMPONENTS OF A SOFTWARE SYSTEM. 20
FIGURE 3. THE STAKEHOLDERS OF AN INFORMATION TECHNOLOGY SYSTEM..... 22
FIGURE 4. AN EXAMPLE OF COMMUNICATION CHANNELS AMONG THE SUPPORT STAFF OF A LARGE SCALE INFORMATION TECHNOLOGY SYSTEM REFLECTING THE PRODUCT ARCHITECTURE. 24
FIGURE 5. MULTIPLE USERS INTRODUCE STOCHASTIC BEHAVIOR AND ADDITIONAL MANAGEMENT TASKS TO AN IT SYSTEM..... 26
FIGURE 6. TECHNOLOGICAL FOCUS WITHIN THE IT INDUSTRY HAS RESEMBLED A PENDULUM..... 33
FIGURE 7. THE IBM SYSTEM/390 DIVISION CONTAINS MANY FORMAL AND INFORMAL LINKAGES WITHIN THE LARGER IBM STRUCTURE..... 37
FIGURE 8. THE SYSTEM/390 BUSINESS UNIT HAS EVOLVED THROUGH FOUR KEY PERIODS. 38
FIGURE 9. AGGREGATE PRODUCT DEVELOPMENT MODELS PROVIDE A MECHANISM OF DESCRIBING PRODUCT ARCHITECTURES IN RELATION TO THEIR DEVELOPMENT PROCESSES..... 39
FIGURE 10. CONSISTENT TERMINOLOGY FACILITATES THE COMMUNICATION OF INFORMATION REGARDING THE SYSTEM COMPONENTRY. 42
FIGURE 11. THE DECOMPOSITION PROCESS IS JOINTLY PERFORMED BY DESIGN AND DEVELOP TEAM MEMBERS.47
FIGURE 12. THE HARDWARE PUSH ERA CONSISTED OF A WATERFALL DEVELOPMENT PROCESS WITH MINIMAL OVERLAP IN ACTIVITY..... 49
FIGURE 13. PARALLEL SYSPLEX INTRODUCED PROCESS AND ORGANIZATIONAL CHANGES, AS WELL AS, A NEW VERSION OF THE SOFTWARE PLATFORM..... 54
FIGURE 14. OPENEDITION ADDED A SECOND PERSONALITY TO THE MVS SYSTEM THROUGH ADDITION OF UNIX APIS. 56
FIGURE 15. THE OPENEDITION COMPONENT WAS DEVELOPED WITHIN AN AUTONOMOUS TEAM STRUCTURE IN KINGSTON, NY..... 58
FIGURE 16. THE SOLUTION MOVEMENT INTRODUCED PROJECT BASED TEAMS TO THE ORGANIZATION..... 60
FIGURE 17. TECHNOLOGIES AND PRODUCTS FROM THE SOLUTION ERA WERE MERGED BACK INTO THE FUNCTIONAL ORGANIZATION. 63
FIGURE 18. DESIGN TEAM MEMBERS SERVE AS CONSULTANTS TO PDTS..... 64
FIGURE 19. DERIVATIVE TECHNOLOGIES MAKE USE OF SERVICES PROVIDED BY THE BASE TECHNOLOGY PLATFORM. 70
FIGURE 20. GAINING THE ADVANTAGE OF NEW PLATFORM TECHNOLOGY OFTEN REQUIRES CHANGES TO THE DERIVATIVE PRODUCT. 71
FIGURE 21. PLATFORM ENHANCEMENT PATTERNS CAN BE DESCRIBED ALONG THE DIMENSIONS OF INTERFACES AND TECHNOLOGY ADVANTAGE..... 74
FIGURE 22. ADOPTING FOREIGN TECHNOLOGIES REQUIRES CONSIDERATION OF COST, BENEFIT, AND SOURCE PLATFORM DEGRADATION..... 74
FIGURE 23. APPLICATIONS MOVED BETWEEN PLATFORMS EXPERIENCE A SOURCE DELTA. 75
FIGURE 24. LEVERAGING OF KEY PLATFORM TECHNOLOGIES IS MOST OFTEN A PARTNERED ACTIVITY AMONG DERIVATIVES AND PLATFORM PROVIDERS. 77
FIGURE 25. PLATFORM LEVERAGE IS NOT BEING FULLY REALIZED WITHIN THE SYSTEM/390 ENVIRONMENT. 78
FIGURE 26. THE RECONFIGURATION OF PERSONS WITH CHANNEL DECODING ABILITY CAN BYPASS FILTERS... 86

CHAPTER 1. INTRODUCTION

This thesis examines the challenges of evolving a mature product platform. It accepts as fact that technological change is inevitable and that a firm's capability to adjust to these changes is not only a determinant of success but also of survival. As such, the focus is on how a firm chooses and eventually adopts new technologies. In particular, it examines how this process is influenced by existing product architectures, organizational characteristics, market dynamics, and technological change. The scope of the analysis is on the design and development organization* . While marketing channels and larger corporate issues are certainly relevant to this topic, inclusion of these in the discussion is beyond the scope of this paper.

A case study provides the basis for the analysis. The subject of the study is IBM's System/390 Software business where the author has worked for several years performing varying degrees of design and development work. The case details the evolution of the System/390 architecture, development process, and organizational structure. The relationship and linkages among these entities is subsequently analyzed. The case is supported through secondary data, informal interviews and surveys, and the author's own observations. Much of the data was gathered and analyzed as part of a re-engineering task force within the System/390 organization. It should be noted that confidentiality concerns prevent exposing raw data in this forum. Where possible, sensitive items that are pertinent to the analysis have been summarized in a format that captures their essence. Any assumptions that may have formed due to the author's own biases and experience have been attempted to be balanced against the existing literature, the collected data, and discussions with others. It is, therefore, thought that any preconceived notions of the author will not deter or have undo influence on the resulting analysis.

System/390 is a viable subject for this case study for several reasons. First, the majority of the product innovation literature is focused on physical products (even studies of the computer industry are often focused on the evolution of hardware such as PCs, microprocessors, and peripherals).

* The design and development organization is defined to be the group that makes the decision regarding which technologies to adopt and is ultimately responsible for implementing them. We assume these groups to contain the chief architects and engineers of the firm.

The method by which software is architected, integrated into products, and subsequently extended provides an interesting contrast to the physical world. Second, System/390 represents a very mature product platform with established architectures, processes, and culture. It is engaging to observe the effect and interaction of these within the new competitive landscape that has emerged with the information age.

The output of this exercise is a model for assessing and integrating new technologies in a stable architecture that exists within a discontinuous environment. The analysis and resulting model should prove useful in future research within the areas of product innovation, software design processes, organizational structure, and evolution of mature product platforms.

Chapter 2 introduces the concept of product platforms and their roles as competitive assets. The challenges involved in evolving and managing these platforms is examined through a survey of the existing literature.

Chapter 3 provides an overview of software products and the evolution of the information technology industry. The challenges facing software design and development are hilighted. Where applicable, these issues are contrasted against similar issues facing designers of physical products. This chapter also provides a terminology base and context for the subsequent case analysis.

Chapter 4 presents a case study of the IBM System/390 Software business. The chapter focuses on the evolution of the System/390 product architecture, development organization, and design process as it relates to the evolution of technology and competitive models within the information technology (IT) industry. While historical data provides insight into the System/390 legacy, the primary focus is on the period from 1990-1997. This timeframe is particularly interesting as it denotes an era when the subject elements have undergone significant change induced by turmoil within the IT industry.

Chapter 5 proposes a model for integrating new technologies into mature software platforms. The model is formulated from the lessons and issues that emerge from the case analysis and the existing literature. The model is discussed and validated through hypothetical application to the System/390 organization.

Chapter 6 provides conclusions and suggests areas of further research.

As a final note, it should be stated that the process of evolving organizations and architectures is continuous and complex. The unique circumstances introduced when dealing with individuals and technologies raise issues for which there is no singular solution. When viewed in retrospect, it is often easy to find fault with earlier decisions on these dimensions. It is not the intent of this paper to criticize historical actions, but rather to leverage the advantage of hindsight to abstract from them the lessons that may be useful in addressing future challenges.

CHAPTER 2. PRODUCT PLATFORMS - BACKGROUND AND LITERATURE REVIEW

Just as quality and manufacturing excellence were key to competitiveness in the 1980s, superior commercialization of technology will be crucial in the 1990s. This capability is necessary due to increasing changes in the business climate. The most notable of these changes being the increased proliferation of new technology in products and the speed with which they render existing technologies obsolete. A 1989 study by McKinsey and Company indicates that successful companies differ from other organizations in four respects. They get products or processes to market faster, use those technologies across a wider range of markets, introduce more products, and incorporate a greater breadth of technologies in them. They thus assert that these are the measures which determine success in commercializing technology and should be the goals which companies strive to achieve.[1]

Product platforms are valuable assets to firms hoping to achieve success along the dimensions described above. Product platforms provide a common technology base which serve as the core of multiple derivative products. Sanderson and Uzumeri's study of the portable cassette player market provides a classic example of the successful commercialization of technology through the use of product platforms[2]. Sony introduced more than 160 variations of its Walkman product between 1980 and 1990. Over halve of the models were able to be brought to market during the final twelve months of this period. These products were based on a platform that Sony refreshed with four major technical innovations. The platform itself was redesigned to support new products every 18-24 months. In addition to successfully commercializing its' technology, Sony has also achieved enhanced quality and lower production costs through leveraging a common technology base. The company remains the recognized leader within the portable cassette market. Figure 1 illustrates the concept of a product platform.

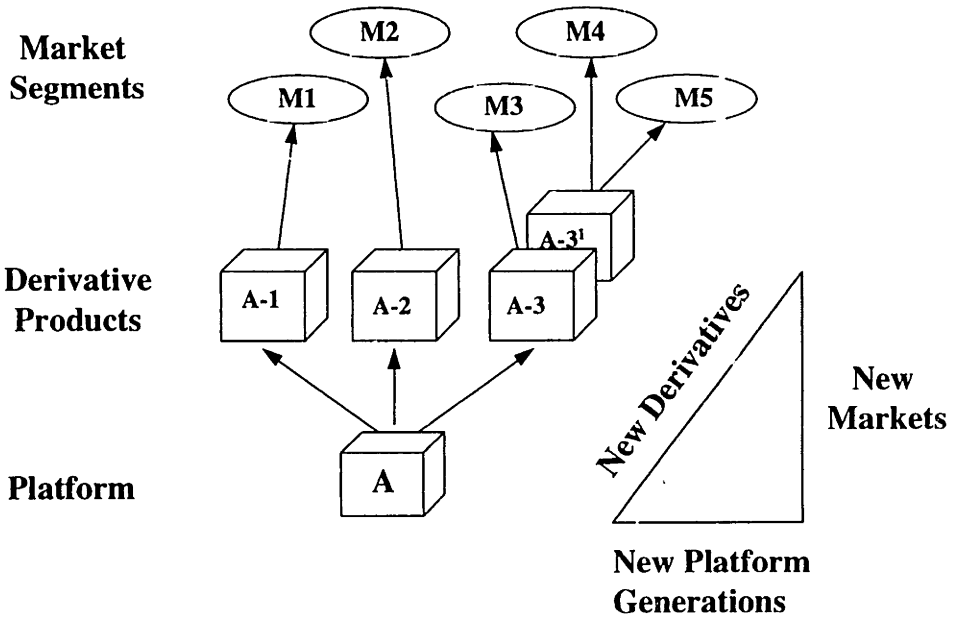


Figure 1. Product platforms provide the basis for multiple derivative products.

Wheelwright and Clark observe that there are a variety of patterns for evolving platforms and derivative products[3]. Steinway Piano company is sighted as an example. Unlike Sony who frequently changes it product platforms, Steinway introduced only one major new model between 1970-1990. However, each piano sold is customized to be a work of art. In terms of platforms and derivatives, it can be stated that Steinway produces numerous derivative products from a small and infrequently changing set of platforms.

Wheelwright and Clark further suggest that the process of creating a strategy for the development of platforms and for derivative products are interrelated activities. They state that platform generations are influenced by factors such as technology evolution, competitive offerings, return on investment, and the firm's available resources. While derivative products are most significantly influenced by their technology platform and the nature of the market niches they serve, they can also be used to extend the life of the platform or to "hold" market position through tactical offerings while new generations of the platform are being created.

The choice of when to invest in platforms versus individual products (or derivative products) is further complicated through the notion of core capability. Core capability refers not only to the capability of the product technology but also of the firm's competency in producing, distributing, and marketing the technology. Also embodied in core capability is an understanding of the characteristics and wants of the target customer set. Meyer and Utterback suggest that "product families and their successive platforms are themselves the applied result of a firm's core capabilities"[4]. Their work has shown that core capability is hard to obtain, directly correlates to a firm's performance, and can be readily lost. They list failure to adopt innovations and new architectures, lack of platform focus*, and the breaking up of design teams as key inhibitors of the development of core competency.

Prahalad and Hamel apply the concept of core competence to the entire corporation in declaring that "the real sources of advantage are to be found in management's ability to consolidate corporate-wide technologies and production skills into competencies that empower individual businesses to adapt quickly to changing opportunities"[5]. They suggest that core capability is an asset of the entire firm and that monopolization of key skills by a particular business unit is not only detrimental to the corporation as a whole but can also result in atrophy of skills within particular individuals.

While products, technology, and organizations are the assets which enhance the firm's likelihood of success, technological discontinuity is a force which can render these ineffective. Anderson and Tushman define technological discontinuities to be "innovations that dramatically advance an industry's price versus performance frontier"[6]. They observe that discontinuities trigger a cycle of technological change consisting of an era of ferment resulting in the emergence of a dominant design, an era of incremental improvements of the dominant design, and finally another discontinuity. The resulting change is viewed to be either radical or incremental in relation to a firm's existing technical assets and competence enhancing or competence destroying in relation to the firm's organizational assets. Their findings build on the earlier work of Campbell who describes technological change as a process of variation - driven by stochastic technological

*Meyer and Utterback refer to this as portfolio management which describes the process of over investing in derivative product development at the expense of platform enhancements. They state that this behavior is often the managerial response to competitive threats and ultimately results in a portfolio of mediocre products.

breakthrough, selection - driven by social, Political, and organizational dynamics, and retention - incremental advances through competence enhancing actions (i.e learning by doing)[7].

Henderson and Clark extend the notion of radical versus incremental innovation through the introduction of "architectural innovation"[8]. Architectural innovation is defined to be "innovations which change the way in which the components of a product are linked together (i.e. the architecture) while leaving the core design concepts (and thus the basic knowledge underlying the components) untouched". They observe that over time organizations gain efficiencies through the development of informal communication channels, information filters, and problem solving strategies that are reflective of the product architecture. They conclude that subtle changes in the product architecture can cause unforeseen disruption to both the product and the organization.

Christensen and Bower show that often established firms fail not because of competence destroying technological advancements within the industry but because the rational decision making processes of these firms favor servicing of the existing customer set at the expense of adopting new technology[9]. They state that "well-managed, established companies are ahead in developing new technologies - radical or incremental - as long as those address the next generation product needs of their mainstream customers. They are rarely, however, in the forefront of commercializing things that appeal to small or emerging markets". Their study of the rigid disk drive industry shows that technology within emerging markets often matures to displace incumbents within established markets. They further observe that "the technological changes that damage established companies are usually not radically new or different from a technological point of view but have the following common characteristics:

1. They represent a different set of performance attributes - ones that, at the offset, are not valued by existing customers
2. The performance attributes that the customers do care about improve at such a rapid rate that these products can eventually invade the existing market."

Finally, recent work by Gulati and Eppinger provide evidence that not only are decisions regarding the architecture of an organization and of its products inter-related but that the two entities often co-evolve[10]. This work is based on a field study of audio system development in a major American automotive firm.

CHAPTER 3. SOFTWARE SYSTEMS AND THE IT INDUSTRY

Much has been written about the difficulties of developing software systems and the resulting "software crisis". Many factors contribute to this situation - the most prevalent being that most software systems are complex systems. They therefore share the properties (and difficulties) common to all complex systems. In addition to these mutual characteristics, software systems also fall heir to other properties that increase their complexity. Regarding the creation of software systems and programs, Liskov and Guttag declare that "good programming involves the systematic mastery of complexity. Practitioners must be convinced that programming is not an arcane art, but an engineering discipline".[11]

To appreciate the challenges in developing and managing software products, it is necessary to first examine more closely the sources of complexity within these systems. This chapter provides an introduction to software systems and architectures. It discusses how the evolution of these systems has been influenced by technological discontinuities within the information technology industry and how these factors have ultimately influenced the manner in which firms compete within this sector.

Software Systems and Architecture

Rechtin states "a system is a collection of things working together to produce something greater. A system can be tangible like a skyscraper, airplane, or communication network, or intangible, like a computer software program or aircraft test program. The unique elements of a system are the relationships between its parts. A system has the further property that it is unbounded -- each system is inherently part of a still larger system"[12]. Booch extends this notion by suggesting that "complexity often takes the form of a hierarchy, whereby a complex system is composed of interrelated subsystems that have in turn their own subsystems, and so on until some lowest level of elementary components is reached. The choice of what components in a system are primitive is relatively arbitrary and is largely up to the observer of the system"[13].

Abstraction* and architecture are the tools which allow humans to understand complex systems. "An abstraction is a simplified description, or specification, of a system that emphasizes some of the system's details or properties while suppressing others. A good abstraction is one that

* Abstractions are also referred to as components, building blocks, and increasingly as "chunks".

emphasizes details that are significant to the reader or user and suppresses details that are, at least for the moment, immaterial or diversionary"[14]. Abstraction supports the fundamental process of decomposition whereby a complex problem is broken into smaller problems until the smaller problems eventually reach a point where they are manageable. Lessons from the earlier sighted research allow us to assert that the decomposition process is heavily influenced by multiple factors including organizational structure, existing problem solving techniques, and existing technical assets (i.e. existing abstractions, platforms, etc.).

Abstraction is a fundamental concept in the construction of software systems. In the context of these systems, an abstraction is a logical representation of some tangible entity or common task. For example, a programmer in the banking industry may choose to create a "savings account" abstraction. The abstraction would mimic its "real world" counterpart as it would contain identity and state such as an account number, an account owner, and a current balance.

In addition to state, an abstraction is defined by its interface*. Interfaces provide the mechanism which allow abstractions to be accessed and manipulated by other entities. For example, the saving account abstraction may contain interfaces which "make a deposit" or "display the account balance". Interfaces within a software system allow information to be provided by the invoker (i.e. the user) in the form of parameters. For instance, the "make a deposit" interface of the savings account abstraction may require the user to provide information such as the account number and the amount of the deposit. The behavior of the interface and the format of the required parameters is defined and documented by the interface creator. In effect, the interface acts as the contract between the abstraction and its user. Adherence to the contract captures the essence of abstraction as it insulates the details of an abstraction's implementation from the user of the interface.

Software abstractions are implemented as programs - where a program is a series of logic. The users of interfaces to software abstractions are themselves abstractions or programs. Unlike physical systems, the components of software systems are logically connected. Logical attachments are possible due to the capability of computer programs to dynamically link to other programs as they execute. That is, one program can request the services of another without having previously established a static connection to that entity. For example, the creator of a program that

* Software interfaces are generally referred to as application programming interfaces (APIs).

is to be subsequently executed can request the services of the "savings account" abstraction by merely "coding" the application programming interface of the abstraction within its own logic. At the appropriate point within the execution of the newly created program's logic, the underlying computer system will transfer control to the banking account logic.

While logical connections allow for added flexibility in the construction of software programs, this benefit is not without a cost. Because there is no limit to the number of users (i.e. attachments) to a software interface, it can be stated that the cardinality of these interfaces is unbounded. This implies that an unlimited number of programs may choose to utilize the services of an abstraction within their logic. The inability to regulate the cardinality of interfaces prior to execution time complicates the implementation of software abstractions as they must be prepared to process multiple requests simultaneously.

If abstraction supports the process of decomposing complex systems then architecture provides the mechanism which allows for their subsequent integration. Architecture is best described in terms of form and function. "The functions of a product are the individual operations and transformations that contribute to the overall performance of the product"[15]. For instance, the function of a stereo is to "play music". Form describes the "physical" elements which implement the product's function. In this example, the form of a stereo system would include items such as the receiver unit, coaxial cables, and speakers. Architecture defines the rules by which the individual abstractions interact to perform a specified function. It could therefore be stated that the architecture of a stereo system dictates how its receiver, cables, and speakers interact to "play music".

While the previous example describes the overall architecture of a stereo system in relation to its major components, it is important to recognize that architecture exists at all levels of a system. For example, the speaker component of the overall architecture is itself a system. As such, the speaker system contains components such as woofers and tweeters and also has an architecture defining how these entities interact to provide the function of the speaker.

Similar to their physical counterparts, software systems also consist of components or subsystems whose interactions are defined by an architecture. The architecture of non-physical products, however, is not as readily apparent as that of physical products. For example, one could infer from

observation that the primary subsystems of a bicycle include the wheels, the frame, and the pedals. The interfaces and the nature of the interactions among these components could further be deduced through observation. This is not the case, however, with software systems.

Due to their intangible nature, software architectures are generally "described in terms of horizontal 'layers' and interfaces between the layers. A layer provides a set of functions and capabilities to the next higher horizontal layer in the architecture. Software companies usually call the lowest layer the system kernel; the highest layer provides the functions and capabilities that end users see and use. Sometimes functions and capabilities from below the highest layer are also available to the end user. The creation or definition of clean abstract interfaces between layers of functionality helps insulate the product's functions from underlying details".[16]

Traditionally, software has been categorized to belong in one of two broad categories: System Software and Application Software. These layers of software act as intermediaries between the end users of the system and the hardware which is ultimately performing the actions. The following describes the primary layers in the technology value chain of an Information Technology system:

Hardware Hardware describes the physical elements of a computing system. This includes items such as the central processing unit (CPU) , internal memory, and storage devices. The computer's hardware is able to be communicated with via its instruction set. The instruction set defines operations which manipulate the storage within the computer in a fashion that may be logically assembled into programs. Printers, terminals, storage devices, and keyboards are physically attached to the main computer. Each of the devices accepts input commands which direct it to perform a particular operation (i.e. print this input stream, display a character on the terminal at physical location "X,Y"). These device drivers provide a very low level semantic for manipulating the specified device.

System Software System Software provides the services which allow applications to be constructed.

The most prevalent layer of system software is known as the "Operating System" or "Kernel". It is the responsibility of this layer to control the hardware resources of the computer such as the central processing unit (CPU), internal memory, and the attached peripheral devices. In addition to managing these physical entities, the operating system also provides interfaces (or abstractions) which allow programs above it in the hierarchy to manipulate these devices in a logical manner. For example, most operating systems support the notion of a "file system" for storing data on an external storage medium. While the storage mediums themselves provide interfaces for reading and writing to the disk, these interfaces deal specifically with the physical organization of the disk and as such are at too low of a level to facilitate efficient, generalized programming. Through defining a set of logical entities (files, directories, etc.) and interfaces for manipulating them, the operating system is able to provide a set of building blocks to application programs which preclude them from being concerned with the details of the storage device where the data ultimately resides. The operating system's implementation of the file system abstraction assumes the responsibility of performing the necessary disk operations in response to application programs invoking the interfaces to the abstraction.

Because the details of the underlying implementation are hidden from the application program, it follows that the operating system could choose to support different physical devices behind the same abstraction barrier. This modularity allows computer systems to take advantage of advances in hardware technology without disrupting the application programs and users it supports.

Also classified as system software are abstractions that extend the function of the base operating system. These functions are commonly referred to as application enablers or subsystems. Examples of application enablers include database management systems, programming languages with libraries of common functions, and graphics subsystems.

A computing system generally contains numerous subsystems supplied by multiple vendors. Subsystems are constructed from the primitives provided by abstractions existing lower in the hierarchy.

Application Software

Applications are what bring computer systems to life and provide their added value. Application software includes programs such as word processors, spreadsheets, or customer service applications such as an order-entry system.

Similar to subsystems, many applications may exist on a single computing image. Also similar is the fact that these applications are provided by multiple vendors and leverage the services provided by the lower levels in the hierarchy.

End Users

End users are the individuals who utilize the services provided by the system. That is, they are the individuals who interact directly with the application software that is installed on their computer system.

The interaction between the various layers in the software hierarchy is illustrated by example. Figure 2 provides such an illustration. The example traces the interactions among the various system componentry as a result of the end user requesting a function from a word processing application via clicking on a menu item. It is important to notice that the abstractions are able to utilize the services of any other abstraction at an equal or lower position in the hierarchy. The example also highlights the ability for software components to "discover" each other at execution time (as opposed to requiring pre-definition or physical attachments of the components).

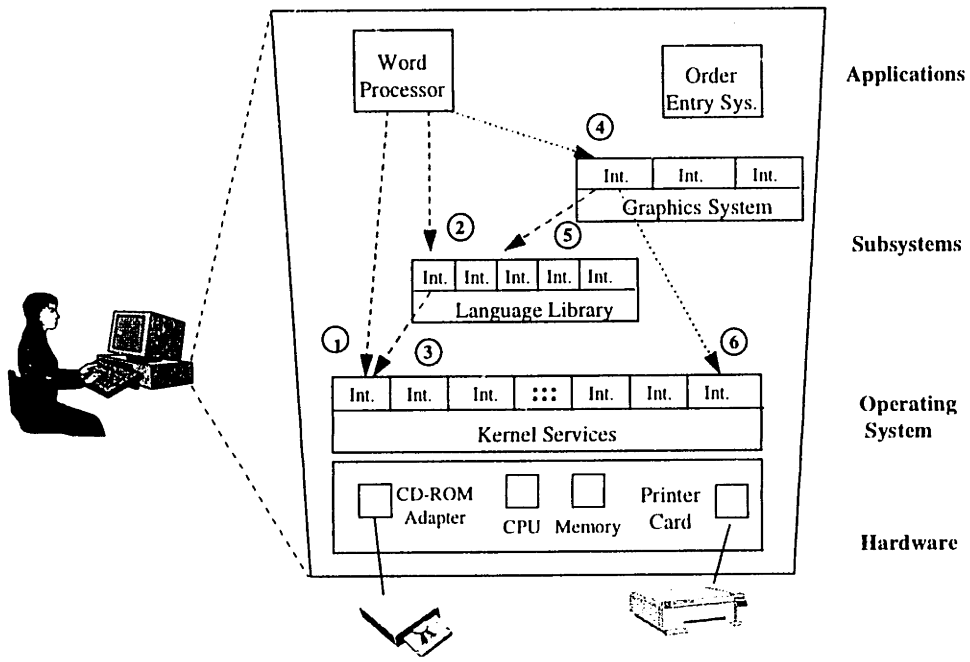


Figure 2. Each user action can initiate a complex set of interactions among the components of a software system.

It is useful to distinguish between system software and application software due to the fundamental differences in the objectives, processes, and skills required to design and develop each. Cussamano and Selby describe the architectures of application software as "feature-driven" while characterizing system software as "performance-driven"[17].

Feature-driven architectures concentrate on usability and features for the end user. As such, soliciting input from these parties is an integral part of the development process. Feedback is typically gathered through the use of prototypes. The low incremental cost of software development makes prototyping an economically feasible activity. Application software is further suited for this type of activity as the building blocks provided by the lower levels of the value chain allow for the expeditious construction of models. As a result of these properties, application program development is best facilitated by a spiral development process in which the product is iteratively developed and provided to users for feedback[18].

Because it must support a larger and more diverse range of functions, system software is more generalized and heuristic in nature than application software. As such, it remains less focused on features and more concentrated on performance. Performance in the context of a software platform refers to the ability to provide consistent, quantifiable, qualities of service in response to various system loads and conditions. Performance of system software also encompasses the notion of reliability. Because the abstractions provided within the system software are in the critical path of a number of higher level applications, the impact of a failure of one of these components is much greater than a failure within a single application. Similarly, system software must protect itself from being damaged by application errors. For these reasons, a primary objective of system software is to provide for maximum fault tolerance and isolation of errors*.

The complexity and interrelationships within system software significantly influence its development process. Like application software, these types of systems are increasingly being developed according to a spiral development model. However, the cycle time for an iteration within this model is much greater for systems software. Where application software is often developed in months (or sometimes even weeks), the cycle time for a new generation of a system software platform is generally measured in years.

Another unique characteristic of software systems is the fact that the customer most often serves as the final integrator of the system components. As such, the system owner incurs not only the responsibility of systems integration, but also the task of performing maintenance and problem determination activities. In the case of a personal computer owner, this task may include installing a new version of a word processing application upon the existing operating system software or monitoring the availability of disk space. In large, corporate computing environments, systems integration and administration activities are typically performed by a support staff consisting of persons with expertise in various system components (i.e. the hardware support team, the operating systems staff, the database administrator, etc.). This notion coupled with the fact that the system componentry is provided by multiple suppliers allows it to be asserted that the construction and operation of a computer system is a continuous process which is collaboratively performed by a

* Isolation of errors refers to the system's ability to prevent errors in a particular piece of componentry from affecting other parts of the system. For example, if an application fails is it the objective of the system to allow that component to fail while allowing others to execute uninterrupted.

loosely coupled group of customer personnel and vendors. Figure 3 illustrates the components of an information technology system in relation to the primary stakeholders.

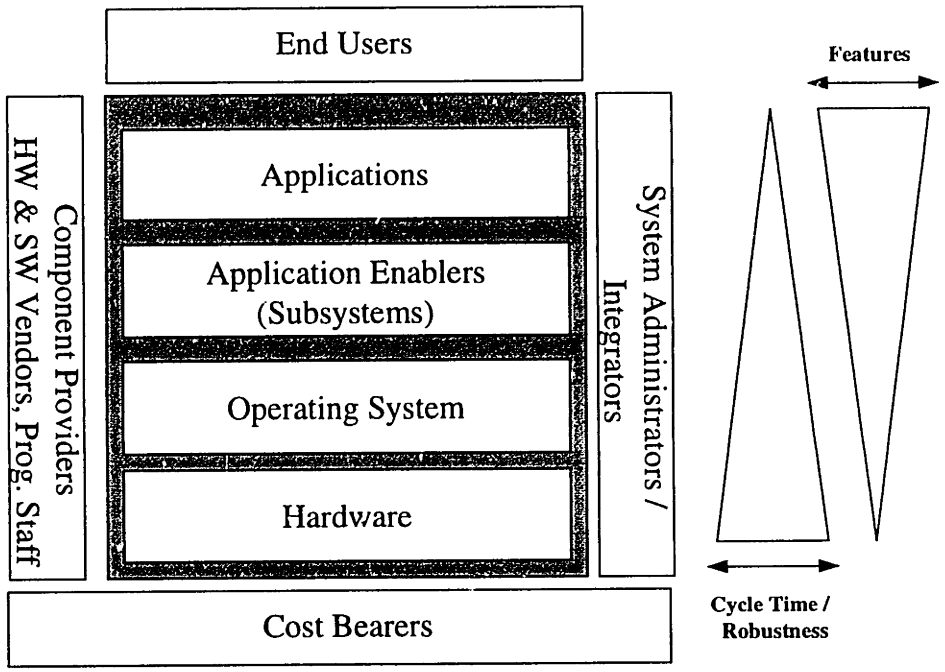


Figure 3. The Stakeholders of an Information Technology System.

While it could be argued that the emergence of pre-packaged systems, such as personal computers shipped with pre-loaded operating systems and application suites, largely removes the burden on systems owners, this is not the case in actuality. The differing development cycles and lifecycles of hardware, systems software, and applications coupled with the constantly changing needs of end users reduces the feasibility of wholesale system replacement. For instance, a personal computer user may extend her system multiple times through the installation of new applications upon the existing operating system and hardware.

Extending systems through the addition of new applications and components may also produce adverse system effects. As stated earlier, software programs often make use of common abstractions provided by the lower levels in the hierarchy. When those programs are deployed in an

environment that is able to execute multiple applications simultaneously*, it follows that the usage pattern of the underlying abstractions (and of the subordinate abstractions which they invoke) may be modified.

The components of software systems are readily replaceable. Because software components are logically bound to the system by their interface, it follows that replacing a component merely consists of placing a new component into the system with a similar interface. The fact that integration of the new component is not subject to formal physical design parameters further encourages this process. Brooks states that "Rarely would a builder think about adding a new sub-basement to an existing 100 story building to do so would be very costly and undoubtedly invite failure. Amazingly, users of software systems rarely think twice about asking for equivalent changes. Besides, they argue, it is only a simple matter or programming".

In addition to system boundaries being extended through the addition of new applications and system upgrades being performed through comprehensive replacements of existing components, a third and more subtle pattern of transformation also occurs. This pattern consists of the reconfiguration of the existing components of the system and is often accomplished through re-packaging. "For example, Microsoft has added many small application functions, including a primitive word processor and calculator to the Windows 3.1 operating system. This makes it hard for simpler products to compete.[19]" As illustrated by the example, reconfiguration does not adhere to the application/system software boundaries that have been previously discussed. It should also be noted that this cannibalization of functions and components does not always occur from the bottom up. Increasingly, applications are being expanded to contain more complex system functions. Lotus NOTEs provides an example of this in that it contains database capabilities in addition to its networking and application logic.

This final pattern is particularly interesting as it relates directly to Henderson and Clark's concept of architectural innovation. From their work, they have shown that over time an organization develops problem solving capabilities that are facilitated through the formation of informal

*Multi-tasking operating systems with Symmetric multiprocessing hardware (i.e. a computer with multiple CPUs) are examples of environments where this behavior is possible. Once restricted to large scale mainframe environments, these features are becoming common in most systems.

communication channels, shared knowledge, and filtering mechanisms among the persons within the channels. They also demonstrate that these channels often reflect the product architecture.

Discussions with large customers indicate that this phenomenon is prevalent within support staffs. Because these groups are generally composed of experts in various component areas, it follows that the reconfiguration of the system's componentry changes the nature of the interactions among the support staff. The fact that reconfiguration may occur through components consuming the function of others further complicates the scenario. Feedback indicates that the strongest barriers within the IT shop still exist along the lines of system vs. applications. This dynamic is further complicated through the advent of downsizing and outsourcing of the IT function. Figure 4 depicts the notion of a software support staff and its communication channels in relation to the product architecture, suppliers, and end users.

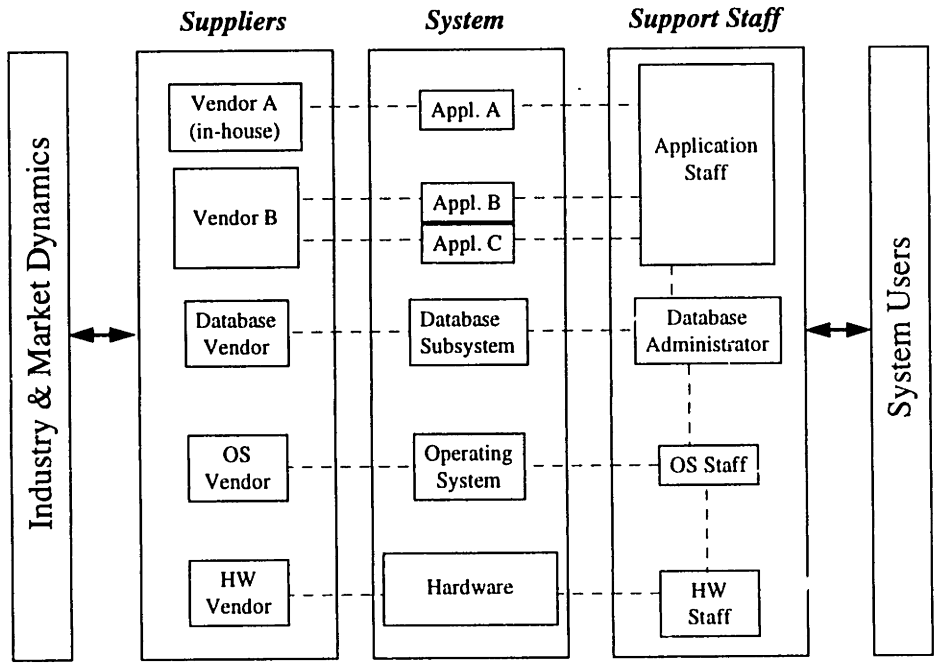


Figure 4. An example of communication channels among the support staff of a large scale information technology system reflecting the product architecture.

To this point, it has been shown that there are significant differences in the characteristics of system and application software. These attributes are particularly distinguishing in single user

environments. However, when a software system is supporting multiple users, application programs are subjected to many of the problems previously thought to be exclusive to system software. Just as it was important for system software to be reliable due to the number of dependent applications, it is important for an application supporting a large number of users to be similarly reliable. The following example illustrates this point.

Consider a large banking operation with hundreds of branches throughout the United States. Each branch employs a number of tellers who process client requests using a customer service application that is located on a single computer at the company's headquarters. The application is accessed via data entry terminals at each teller station. In this instance, a single application supports numerous users. It therefore follows that the cost of an application failure in this environment is much greater than in the single user environment. While this example has been fabricated, the cost of outages in large, commercial environments supporting multiple users are real. Recent studies estimate the cost of downtime for business systems to be approximately $1400 per minute[20].

Multi-user systems also increase the likelihood of stochastic behavior within the system. Consider the plight of a company (specifically, the plight of its IT system and staff) in the pharmaceutical industry. The company has a centralized IT system which receives data from point of sale devices located at retail outlets. The transmission occurs via satellite. On average, the system processes approximately 12,000,000 requests (or transactions) per day. During peak periods (typically - the lunch time and end of working day drive period) requests rates often rise to 400 transactions/second. Certain periods of the month produce request rates in excess of 16,000,000 transactions/day. In short, multi-user systems that support large business processes are subject to discontinuous input patterns. These patterns are often reflective of societal flow. It is imperative that IT systems supporting these processes be able to respond to these conditions.

The potential for random behavior is further illustrated when considering the interaction of the components within the pharmaceutical system. While the system processes a large number of requests per day, each of those components results (as described earlier) in calls to components lower in the hierarchy. In the case of the pharmaceutical system, each sales request results in multiple updates to a database. The database component itself processes approximately

500,000,000 requests per day while generating on average 1715 requests/second to one of its subsidiary components. Figure 5 illustrates the concept of a multi-user commercial system.

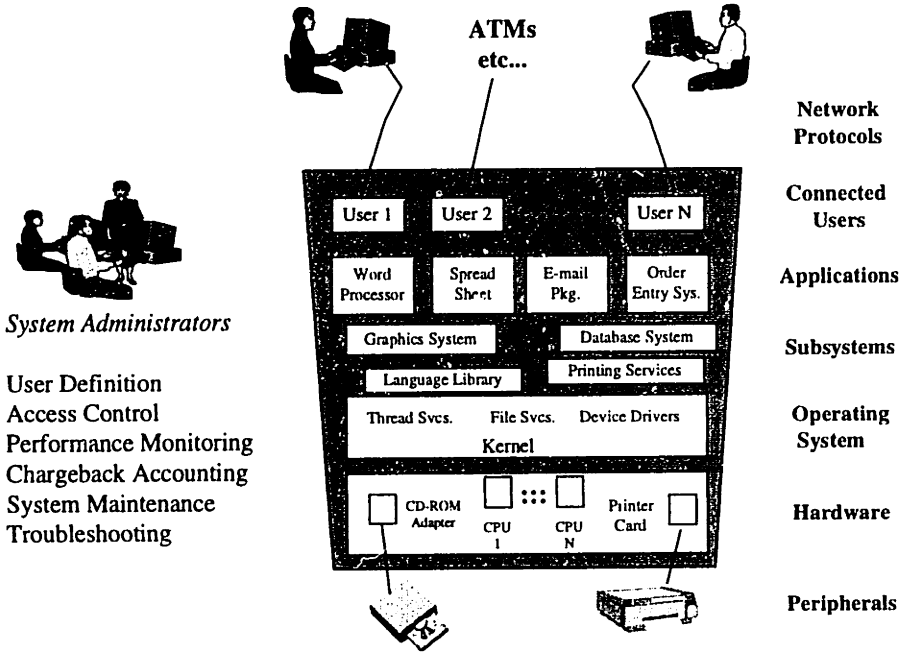


Figure 5. Multiple users introduce stochastic behavior and additional management tasks to an IT system.

The potential for chaos within multi-user systems becomes even more probable when the access points for the users are not known beforehand. While the previous example illustrated an environment with varying degrees of discontinuity in system usage patterns, there still remained potential for estimating the bounds of the system - that is, the input request rate was ultimately bound by the number of retail outlets and the rate at which sales could be processed by the point of sale devices. This is not the case, however, in worldwide web and internet environments. A programmer from a publishing firm recently pointed out that if a particular website happens to get highlighted by one of the numerous "hot site" publications, the number of requests per day for that system can go from 200 to 200,000. In this case, the hosting server would not have knowledge of the accessing users or their usage characteristics beforehand.

Multi-user systems also introduce additional management tasks. In particular, users need to be defined to the system and given a set of privileges for accessing resources. Additionally, usage of the system needs to be monitored for the purposes of chargeback accounting. In environments where the users are known beforehand, this task is achievable yet burdensome. This becomes much more of a challenge in open environments such as the internet. Mechanisms for identifying, authenticating, and managing users in this environment, such as public key encryption technology and certificates maintained by third party providers, are currently being developed but have yet to reach an acceptable level of maturity. This again represents a change in technology that similarly disrupts the value chain.

Shifting Focus within the I/T Industry

While the flexibility achievable through software allows for a continual redefinition of the roles of the components and layers within a system, external factors such as market dynamics and discontinuities ultimately determine the overall requirements of a system and the rate at which they change. As stated earlier, discontinuities represent breakthroughs in technology or process that significantly advance an industry's price versus performance ratio. The result of these advances is often new paradigms or segments within an industry. Drawing on Christensen and Bower's observation that new technologies frequently mature to invade existing markets, it follows that the performance characteristics of a dominant solution are comprised of the set of dimensions that were introduced with each discontinuity throughout its evolution.

In the broadest sense, the IT industry can be characterized as having supported three primary paradigms: Host based computing, Desktop computing, and Network computing (the current industry focus). Each of these paradigms have introduced new technologies, processes, and competitive behavior. The following provides a brief profile of each of these paradigms highlighting their key contributions to the industry.

Host-based Computing

Host (aka Mainframe) computers were introduced in the late 1950s as the first commercial computing machinery. "The term mainframe was introduced very early in the computing era to differentiate the primary computing system, which supported the most business-critical applications, from those performing less critical functions. As these computing systems evolved

into increasingly large general-purpose systems, they also evolved toward centralized support by Information Systems organizations with the technical expertise to support them and the software that ran on them.[21]" Host computers generally exist within a controlled area referred to as the "Glass House". Access to this area is often restricted to the support staff - which controls all changes to the system. Changes to the system are tightly controlled and usually entail a test period of months before being moved into the production environment.

Mainframes evolved early to support large numbers of concurrent users who exist either locally or remotely via a proprietary network*. End users of host based systems generally accessed the system via character-based terminals. These terminals, while reliable and efficient, provided very little flexibility to the end user. While rudimentary items such as editors existed, few personal productivity applications were available to end users.

Initially, the application software for host based systems was largely developed by the programming staff at each installation. Typically, the programs were written in Assembler language - a primitive language that maps very closely to the machine architecture. The applications themselves supported business processes such as order entry and payroll systems. Increasingly, application enablers such as database management systems and transaction monitors have emerged to facilitate the construction of large on-line transaction processing (OLTP) systems. It should be noted that many of today's key enabling technology products were originally developed as customer field experiments.

IBM and its System/390 architecture has remained the dominant player within this segment. Historically, the segment has been very vertically integrated with IBM supplying the hardware, operating system, and the majority of the key subsystems. As previously stated, application software was primarily developed by customer application programmers. "In the late 1960s, once the System/360 became the dominant mainframe solution, IBM began to unbundle pricing and selectively open the system, in part because of government pressure. Published standards permitted competitors and component suppliers to produce a wide range of IBM compatible products and programs that were interchangeable with, and sometime superior to, IBM's own.[22]"

Today mainframes remain the primary execution environment for large scale mission critical business applications. These systems are able to support thousands of business transactions per second from an even larger number of concurrent users with greater than 99% system availability. It is estimated that greater than 2/3 of the world's operational business data exists on mainframe computers[23].

Desktop Computing

Desktop computing emerged in the 1980s with the introduction of the IBM Personal Computer. The story of the ensuing technology wars in which Microsoft (with its Windows Operating System) and Intel (with its microprocessor technology) emerged as the dominant technology providers is well documented.

Perhaps the greatest contribution of the personal computer is that it has "humanized" computing[24]. This humanization can be largely attributed to graphical user interface technology which allows users to interact with computers and applications in an intuitive manner. With this technology, new classes of applications have also emerged ranging from video games to personal productivity packages such as word processors and spreadsheets. More important than the emergence of the applications themselves is the fact that affordable personal computers have allowed computing to move from the corporate environment into the mainstream of society.

The desktop era has also produced a new model for developing and distributing software. Packaged software that is able to be purchased at a retail store, installed on a workstation, and immediately used represents a drastic change from the original centralized mainframe model. Similarly, the low entry barriers for desktop software development has created an environment conducive to entrepreneurs which has allowed third party software vendors, often in the form of start-up companies, to become pervasive. Desktop software itself is generally developed in high level languages such as C or C++. The enhanced graphics capabilities introduced in this arena have also allowed for the proliferation of visual builder tools that facilitate application development.

---

* Proprietary network are either owned by the corporation or executed on leased public lines. In either case, only company personnel are able to access the system and only from designated terminals.

Also a product of the desktop computing movement is the notion of formalized beta programs. The large number of vendors, products, and system configurations that are available in this arena make it impossible to test a piece of software in every environment. For this reason, desktop software is often subject to quality problems. In an attempt to confront this problem, beta programs are organized which allow users to obtain "pre-release" versions of the software with the understanding that the code is in an unfinished state. In doing this, the company is able to effectively utilize the user community as a test resource. Microsoft used 15,000 beta testers for its Windows 3.1 product and 75,000 for it Windows NT 3.0 product[25]. This concept is one that is not feasible in the physical world where re-work to the product represents significant capital expense. It is also less attractive on multi-user software systems where the cost of an outage is significantly higher.

Finally, the number of competitors and nature of the competition in the desktop segment has resulted in time to market becoming more prevalent - even when it is at the expense of initial quality. This mentality is reflected in the statements of the chief IT architect for a large European company. The architect recently stated, "We know that the first release of a Microsoft product generally will not work to our satisfaction, but we buy it anyway. We use that time to get familiar with the product and report bugs. The second release will be a little better and often by the third release it is able to do what we want. In effect it allows us to grow with the product"[26].

While no longer the primary technology focus within the industry, continued advances in personal computer hardware price/performance ratios coupled with the increased sophistication of end users allows the desktop market to continue to grow.

Network Computing

The third and arguably most disruptive paradigm within the industry evolution is known as "Network Computing". Network Computing proposes a model which allows heterogeneous computing systems to interoperate in a seamless fashion. For example, a personal computer user in New York City may transparently access information on a mainframe computer in Singapore. In this example, the mainframe computer acts as an extension of the user's desktop system. This extension of the logical boundaries of an IT system is evident in statements such as those by Sun

MicroSystems CEO Scott McNeely who proclaims "The Network is the Computer"[27]. Key in enabling this movement is the fact that high speed digital communication networks are becoming ubiquitous. Also implied in this model is a need for standards based communication protocols which allow disjoint systems to interoperate over these networks. While Network Computing has become most prevalent in the mid-1990s, the attempt to integrate systems in the manner described above has been ongoing for many years.

Unix systems originated in the 1960s and were billed as the first open systems - a system with a public architecture developed jointly by a consortium. Unix systems are very pervasive in academic environments but until the last decade have had very little commercial presence. Many of the technologies developed in this arena, such as the TCP/IP communication protocol and MIT's Kerberos Authentication mechanism, have become key in the construction of network applications.

The current network computing movement was most recently preceded by a similar movement known as "Client/Server". With a few notable exceptions such as Lotus NOTEs and SAP R/3, Client/Server applications have been limited due to their cost, complexity, and poor reliability. This lack of success is largely due to the fact that each vendor attempted to solve the problem independently by creating client and server versions of their application that executed on different machines and used a proprietary protocol for communicating. Orfali, Harkey, and Edwards state "Client/Server has applied a giant chainsaw to centralized monolithic applications, slicing them into two halves. Unfortunately, you simply wind up with two monoliths instead of one: one running on the client and one running on the server. Enablers and visual builder tools help leverage and deploy applications more quickly. But, by and large, today's client/server applications remain difficult to build, manage, and extend.[28]"

The advent of the World Wide web has brought network computing into the focus of not only the IT industry but society as a whole. The World Wide Web, with its Hypertext Transfer Protocol (HTTP), has largely forced abandonment of the proprietary standards of the Client/Server era. The Web has proven that a heterogeneous compute model with large servers providing services to thin clients via a graphical interface (i.e. web browsers) over a standardized transport protocol is feasible. More importantly than providing a reference model, the web has served to enlighten lay persons to the realization of access to anything, anywhere, anytime. The evolution of Network

Computing strongly supports Anderson and Tushman's notion that the selection of a dominant design is largely a socio-cultural process.

While the worldwide web has marked a significant technological and societal advance, it should be noted that to date it has served primarily as a mechanism for viewing static data in the form of web pages. This technology, however, has been recognized to not only be useful in support of existing processes but as (and more importantly) an entirely new channel for doing business. Meeting this challenge, however, requires that the technologies deployed within network computing be enhanced to provide the same qualities of service (i.e., reliability, scalability, security) as existing large scale applications have grown to require. This opportunity has spawned a new era of design competitions among firms hoping to satisfy this requirement.

This again supports Christensen's theory that a disruptive technology introducing a new desired characteristic is often accepted and incrementally advanced to a point of equivalency along the performance dimensions of existing technologies. In this case, network applications are being enhanced to invade the traditional mainframe segment containing mission critical business applications. Figure 6 illustrates the paradigm shifts that have occurred within the IT industry and the fact that Network Computing has resulted in the convergence of many of the key technologies.

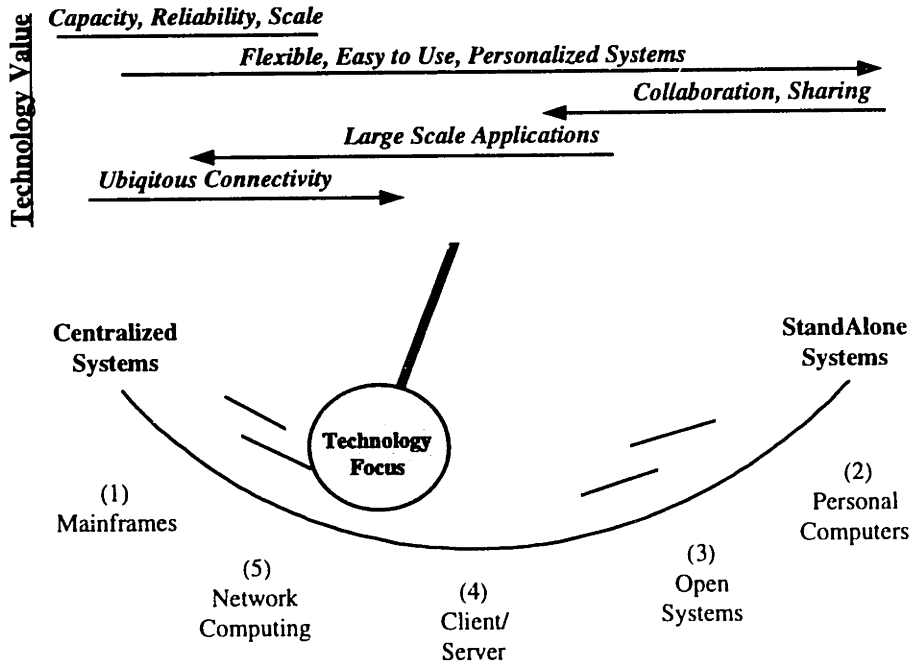


Figure 6. Technological focus within the IT industry has resembled a pendulum.

The intensity of competition and pace within the IT industry has once again increased largely due to the increased market pull that has been created by the Network Computing movement. This intensity is reflected in the notion of the "web year". Web years reflect the cycle time of products within this segment. Currently, one calendar year is approximately equal to four web years. In order to compete in this intensified environment, companies are beginning to formally and informally partner in the development of solutions. Evidence of this partnering exists with alliances between startup companies such as Sun MicroSystems and Netscape. It is also evident in formal acquisitions such as IBM's purchase of Lotus Development, Transarc Corp., and Tivoli Systems.

Advances in this arena are generally being pursued in two modes. The first mode is product focused and represents the coupling of existing applications and technologies to form new solutions. Unfortunately, coupling disjoint technologies does not always produce the desired effect. These adverse effects most often occur when technologies are blended which were developed in environments with different design objectives -- leading to a sub-optimal overall solution.

The second mode is more platform focused and represents a movement to define a new computing platform for next generation network applications. This movement is leading to the development of advanced architectures such as Microsoft's Distributed COM architecture, the OMG's* Common Object Request Broker Architecture (CORBA), and Sun Microsystem's Java Virtual Machine.

Implicit within both movements is significant activity being performed by all vendors as they attempt to develop new products and/or enhance existing products and platforms to participate in the emerging computing model. Undoubtedly, some firms are better positioned to perform this task than others.

Competing Within the Information Technology Industry

* OMG is the Object Management Group. This represents a consortium of over 500 companies who are seeking to define a distributed object computing architecture.

The previous sections have illustrated that software architectures are layered, flexible, and dynamic. It has shown that while these characteristics provide significant benefit in the construction of systems, they also pose challenges in managing and communicating the boundaries of not only the components within the system but of the system itself. Highlighted is the fact that management of these boundaries is a shared activity among stakeholders existing in multiple firms and that changes in the system architecture often result in secondary effects within the value chain. Also illustrated is the fact that the IT Industry has undergone unprecedented technical change that has not only affected the business models, technologies, and products of the competitors who compete in this arena but of society in general. Given these issues, the principle question becomes "What are the critical success factors that allow firms to not only survive but succeed in this environment?"

Morris and Ferguson assert that "a new paradigm is required to explain patterns of competitive success and failure in information technology. Simply stated, competitive success flows to the company that manages to establish proprietary architectural control over a broad, fast moving, competitive space[29]." They further state that "conventional wisdom argues that in an open systems era, proprietary architectural control is no longer possible, or even desirable. In fact, the exact opposite is true. In an open systems era, architectural coherence becomes even more necessary. While any single product is apt to become quickly outdated, a well-designed and open-ended architecture can evolve along with critical technologies, providing a fixed point of stability for customers and serving as the platform for a radiating and long lived product family." They indicate (as does Tushman) that in these environments firms tend to engage in architectural competitions.

Hagel extends the notion of architectural competition through the concept of a "web". "Webs are clusters of companies that collaborate around a particular technology. Probably the best known is the Microsoft and Intel personal computer web, in which hardware and component makers, software developers, channel partners, and training providers combine to deliver the overall value proposition of a Windows PC. Webs emerge from the turmoil wrought by uncertainty and change. They spread risk, increase flexibility, and reduce complexity for individual participants. They are characteristically the work of a single architect (or shaper), which (unlike a monopolist) maximizes the size of the web by giving away value to other companies.[30]"

Two roles exist within web based competition: shapers and adapters. Adapters (as the name implies) do not attempt to influence events but rather remain positioned to link with webs that are being developed by others. Shapers, on the other hand, attempt to mold their environment. Success factors for shapers include: ownership of a key platform technology, reliance on economic incentives (rather than contractual relationships), and active management of increasing returns dynamics.

The last factor bears examining as it describes a concept that displaces many strategic frameworks with processes based in microeconomics and organizational theory. "Increasing returns economics rejects the conventional wisdom that industries are rapidly prone to diminishing returns as a result of competition between firms for scarce resources. This view states that returns from marginal investments quickly shrink as competition mounts. Soon firms cut back their investments to levels justified by average industry profits, and industry structure stabilizes. Under increasing returns, however, returns from marginal investments go up rather than down.[31]" This phenomenon is shown to exist in knowledge based products such as software and drugs. "Organizational coevolution is a major source of increasing returns because adding more firms to a group not only enlarges it, but actually draws other firms in as well." The process of "locking-in" other web participants (as opposed to locking out competitors) is key in these strategies.

In effect, competing in the IT industry requires careful management and frequent updating of a firm's technology base. Adler and Shenhar warn, however, that "a common mistake in assessing an organization's technological base is narrowing the review to matters of technical competence[32]". They assert that the technological base consists of: technological assets, organizational assets, external assets, and project management. Their work indicates that of the four dimensions - organization usually proves to be the limiting element. They further propose that within a firm's organizational assets exist a hierarchy consisting of skills, procedures, structure, strategy, and culture and that the degree of technological change introduced requires changes deeper in the hierarchy (i.e. small changes may only change skills, large changes may require changes in all layers through culture).

In the subsequent chapters, we examine a large mature software platform, the IBM S/390, which has in recent years undergone large changes within its technical base. It has experienced similar

changes within its other assets. The examination will review these changes from a historical perspective and attempt to understand linkages. The retrospective view also provides evidence which will allow a model to be formulated for how to integrate new technologies into mature software platforms.

CHAPTER 4. CASE STUDY: IBM'S MAINFRAME SOFTWARE BUSINESS

IBM's System/390 computing business is a $20 billion dollar per year entity that provides the computing services for the mission critical business applications of Global 4000 corporations. The System/390 business unit serves as the cornerstone of the corporation as it generates 15% of the corporation's total revenue and an even larger percent of its profit. In addition, large systems serve as the basis for a majority of the revenue generated from services.

The System/390 business unit exists as a division within a larger entity known as the server group. The server group contains divisions responsible for producing the base hardware and operating systems support for various categories of computing. Little known is the fact that IBM is also the world's largest software company producing $13 billion per year in revenue. In an effort to increase its presence within the network computing arena, IBM has recently purchased controlling interest in companies such as Lotus Development Corporation, Transarc Corporation, Tivoli Systems, and NetObjects. Figure 7 illustrates the System/390 division in relation to other key IBM entities.

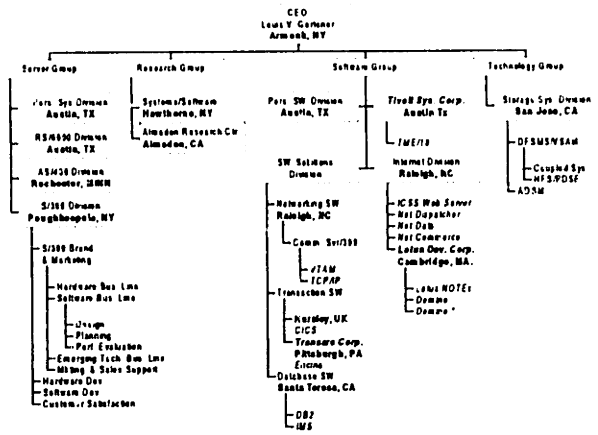


Figure 7. The IBM System/390 division contains many formal and informal linkages within the larger IBM structure.

Originally developed in the 1960s, the System/390 architecture has evolved and been extended over time in support of multiple product families and information technology paradigms. The latest technology paradigm, "network computing", calls for seamless worldwide interoperability among heterogeneous computing systems. Driven by widespread acceptance of the internet and world wide web, this movement represents a convergence of disjoint technologies, development cultures, and consumer markets.

The following sections provide case data on the System/390 business. The focus is on the evolution of the product architecture, organizational structure, and development process in relation to both internal and external events. In order to illustrate this evolution, the data is presented in chronological sequence. Interviews with engineers, architects, project managers, and functional managers along with secondary data have prompted the data to be organized into four sections representing significant eras within the evolution of the subject entities. Relevant observations from interviewees are annotated within the discussion. Figure 8 depicts the eras to be described including the key characteristics of each.

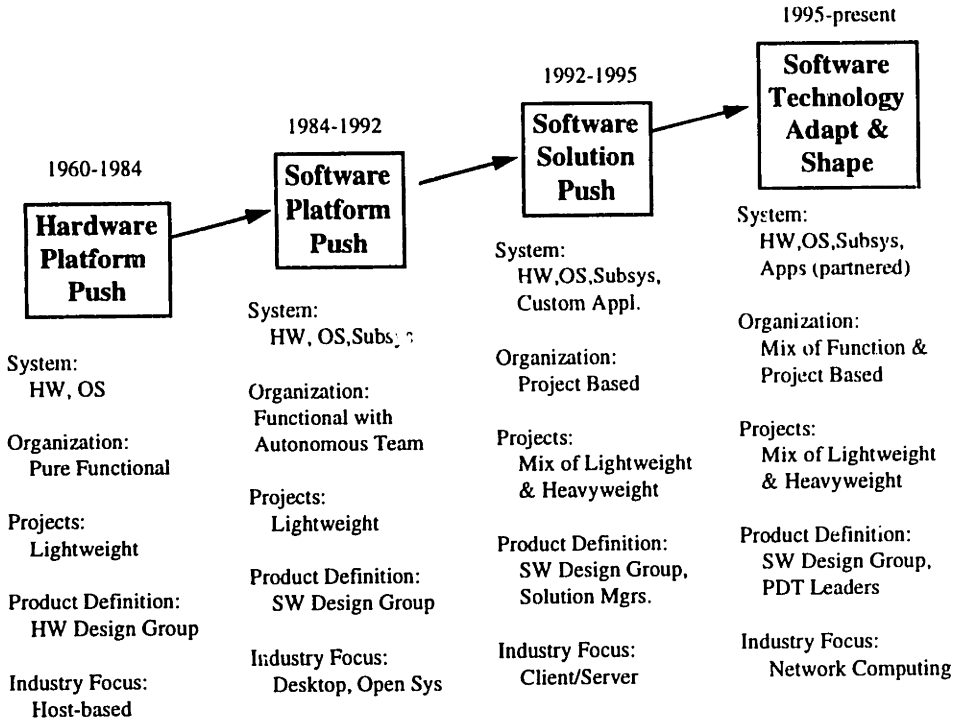


Figure 8. The System/390 business unit has evolved through four key periods.

Wheelwright and Clark have stressed the utility of functional maps and aggregate project plans in helping firms to analyze their development environment. While companies typically have processes, architectures, and organization structure thoroughly documented within their records, these typically exist in disjoint sources making it difficult to discern their relationship. This analysis proposes a model for describing these in a collected fashion through the creation of an "Aggregate Product Development Map (APDM)". The APDM uses the product architecture as a base for describing process flow and sharing of functional activities. Key within the APDM is its notion of explicit notation of boundaries not only within the product but also within the functional process flow. The criteria for determining these boundaries and the feasibility of using the architecture as a base is discussed in the subsequent analysis. Figure 9 provides an example of the Aggregate Product Development Map framework.

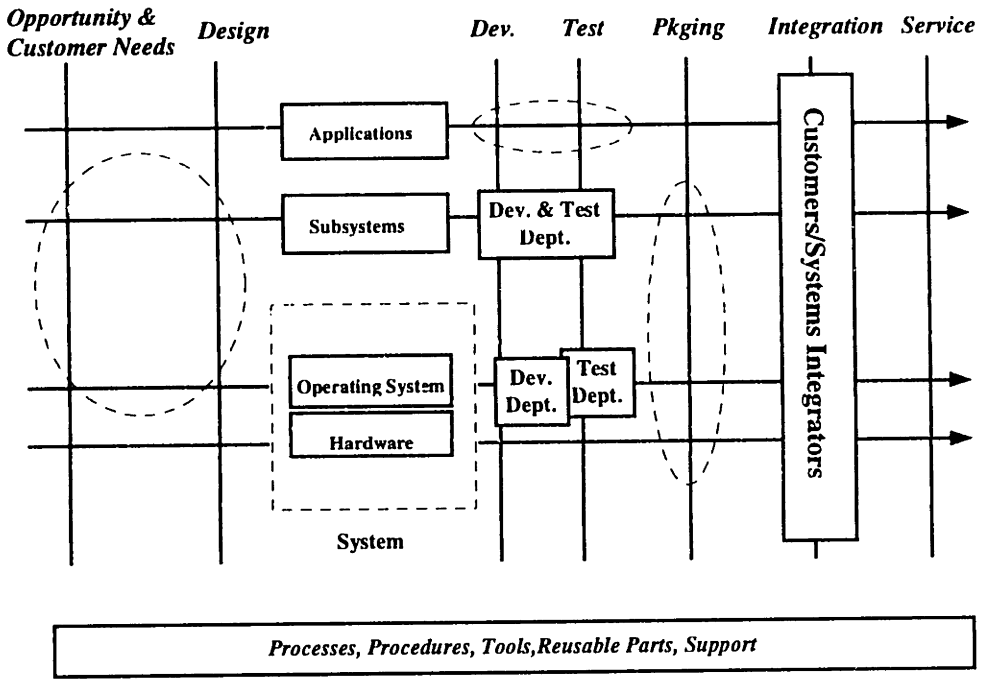


Figure 9. Aggregate Product Development Models provide a mechanism of describing product architectures in relation to their development processes.

Hardware Platform Push Era

Hardware Platform Push describes the predominant mode of operation of IBM's System/390 business line from its inception in the late-1950s until the mid-1980s. The System/390 architecture has remained the dominant design within the host-based computing segment throughout this time period. Host-based computing, itself, was in the primary focus of the industry throughout much of the period. As such, IBM (and specifically, its hardware group) has played the role of "shaper" within this arena. The System/390 architecture is diligently described in a publicly available document known as "Principles of Operation".

As previously stated, a computer architecture dictates the set of instructions to which a piece of compatible computing machinery must adhere (i.e. it describes the interface to the machine). IBM has used its System/390 architecture to support a large range of machines with various

performance capabilities. These machines were originally built on a leading edge technology known as "bipolar". Bipolar technology continually supplies power to densely packed circuits to speed the execution of logical operations. While bipolar represents a high performance technology, it also introduces constraints due to the amount of heat that is generated within the underlying circuitry. Moreover, each increase in computing power introduces corresponding increases in generated heat. This has prompted chilled water cooling systems to be installed on the largest computers significantly increasing operational and facilities costs.

IBM has provided four primary operating systems in support of the System/390 architecture. These operating systems provide the end user and programming interfaces to the hardware. Each of these has unique characteristics which make them better suited for particular customer segments. Of these, the most prevalent is the MVS (Multiple Virtual Storage) operating system. It is MVS that supports the mission critical workloads of large corporations. As such, MVS serves as the focus of the System/390 software business and as the subject of this case study.

The primary competitive pattern throughout the hardware push period was one of incremental, competence sustaining changes to the architecture with larger performance strides being made in the implementation of the machines supporting the architecture. In retrospect, the changes are viewed as competence sustaining in the sense that they generally occurred in the form of extensions to the existing instruction set. A fundamental principle that remains today within this architecture is the notion of upward compatibility. That is, programs that are written and deployed on a processor supporting a certain version of the architecture are guaranteed to execute unchanged on a processor supporting a subsequent version of the architecture. The original System/360 architecture has undergone four major innovations bringing it to its current form known as System/390.

Product Architecture

The terminology used to describe a System/390 configuration reflects the view of the system boundaries during this period*. In particular, the combination of the hardware and operating system

* It should be noted that this terminology continues to exist today.

were referred to as "the system". Subsystems and applications were viewed as extensions to the base and therefore not of primary concern to the developers of the system.

The parts of the operating system are referred to as "components". Each component reflects a functional area of the system and typically has a three letter acronym describing that function. For example, "VSM" is the Virtual Storage Manager component while "GRS" is the Global Resource Serialization component. Components are similarly decomposed into subcomponents which themselves consist of individual programs or "modules".

Modules serve as the "physical" elements of the MVS system. That is, they are the entities that are created by programmers and assembled into higher level abstractions. These programs have historically been written in assembler language or an IBM internal development language known as PLAS*. Construction of programs is facilitated by libraries of common parts known as macros. These libraries exist at the component level (i.e. routines that provide function specific to that component) and at the system level (generalized functions that are applicable throughout the system). The population of reuse libraries has typically occurred in an ad hoc fashion through developers generalizing common functionality that is developed during product development. In effect, common functions are "harvested" from development projects.

MVS employs a strict convention for creating and naming modules within the system. In particular, modules have eight character names. The first three characters denote the component, the next two characters designate the subcomponent within that component with the last three characters uniquely describing the module within the specified scope. Consistent naming of componentry is necessary when building large systems. It is especially necessary within software systems - again, due to the intangible nature of the componentry within an assembled software system. This consistency has allowed sophisticated problem determination and diagnostic procedures to be constructed in support of the MVS operating system. The naming convention has also served a second and possibly more important function in that it has created a common terminology for system description that is not only shared internally within the design and development organization but also externally with customers. This is especially useful for customer service personnel. The MVS decomposition model is shown in Figure 11.

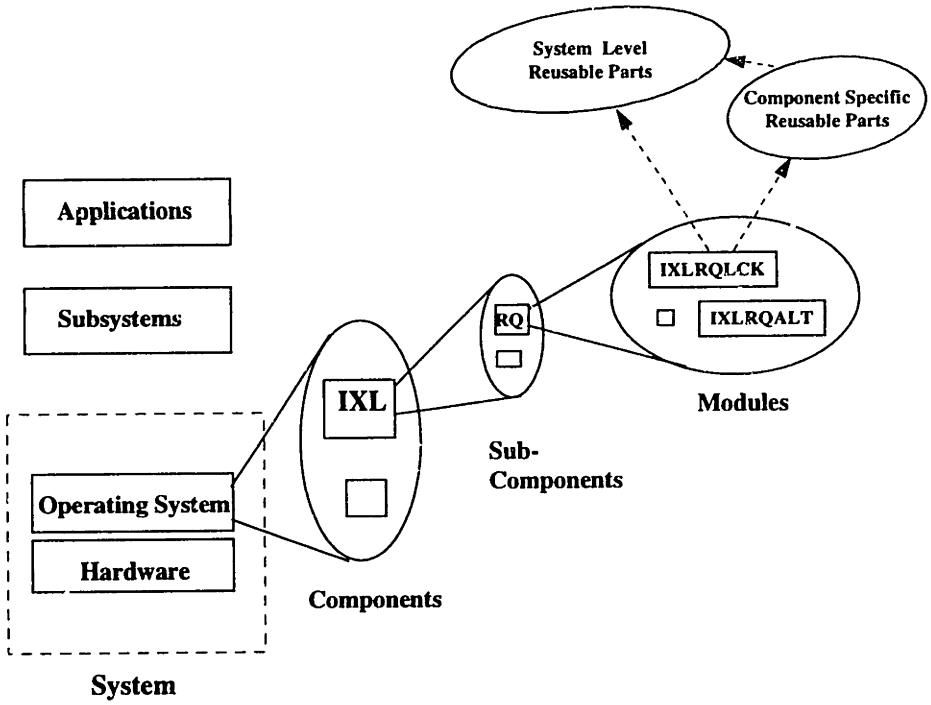


Figure 10. Consistent terminology facilitates the communication of information regarding the system componentry.

Organizational Structure

Initially the design, development, and marketing of System/390 products was performed by individuals existing within a pure functional organizational structure. Within a functional area, departments were organized in a manner which reflected the system componentry (i.e., a department responsible for function test of component A, a department responsible for the development of component Z, etc.). With the exception of the Marketing and Research functions, which existed within their own divisions, all base development activities were performed by employees within the System/390 division. Over time, the organization evolved to a lightweight matrix model with release managers serving as coordinators for the development of MVS Releases. Base development was almost exclusively performed at a single site within the Mid-Hudson Valley

* PLAS stands for Programming Language for Advance Systems. The language is an optimized variant of the formerly popular PL/1 programming language.

region of New York state*. Subsystem support was provided by IBM groups existing in other divisions and other development labs worldwide.

The following describes the key functional areas and their role within the development of the MVS Operating system:

Project Office

This group contains individuals who serve as lightweight project managers for releases of the MVS operating system. These individuals are primarily responsible for tracking project progress and insuring that commitments made by individual functional areas are met.

Software Design

The members of the software design team serve as the proprietors of the overall software system architecture. In this role, they are responsible for determining the content of MVS releases and for producing the necessary system and component level design.

The team has historically been populated with many component experts and very few generalists. These experts have typically emerged from within the development community. Typically, individuals spend 7-10 years within the development of a particular component (or a small set of components) prior to joining the design organization.

Software Development

Development groups have traditionally served as the owners of the implementation of the system componentry. For example, there exists a VSM group that is responsible for development of the Virtual Storage Manager component of the system. Development entails the design, creation, unit test†, and maintenance of the subcomponents and modules which comprise the component. Development teams typically consist of an overall development leader and several subcomponent teams. Similarly, subcomponent teams

* The Mid-Hudson valley includes the cities of Poughkeepsie, Kingston, and Fishkill, NY. At its peak period in the mid-1980s, approximately 30,000 IBMers were employed in the region.

† Unit test describes the process of a developer testing an individual unit, typically a module, in an isolated environment in preparation for introducing it into the larger arena.

consist of a team leader and several team members. All members are responsible for the creation of code. Growth patterns within a development group generally consist of ascending from being a development team member to leading a subcomponent team, and eventually to becoming a component leader.

Function Component Test

Function Component Testers (FCT) are responsible for the functional verification of the individual components within the system. Function test of components is performed on simulated test systems that are able to emulate various hardware functions and also have special features which facilitate the process of debugging errors. Similar to development, component test groups consistently "own" the verification of particular components across releases.

Software System Test

Similar to Function Component Test, System Test is responsible for performing functional verification activities. Unlike function test which is performed on simulated systems, these tests are executed on the actual hardware. This phase of testing often surfaces errors due to timing conditions and system utilization levels that are not able to be produced within the simulated environments.

Performance Test

Performance test, like System test, provides an integrated hardware/software test. However, their focus is not on functional verification but rather on exposing bottlenecks within the system which inhibit throughput or result in excess use of system resource. This is accomplished through exercising industry standard benchmark workloads and specialized stress tests.

Packaging

The packaging, or "build", group is responsible for building the product onto a tape for delivery to the customer. Building the product entails integrating the various software programs into a system image. The build group is also responsible for packaging the product for use in internal testing during the development cycle.

Customer Service

The customer service teams serve as the preliminary entry point for reporting

of customer problems (aka field bugs). Specifically, customers report problems through opening a Problem Management Record (PMR). Service personnel investigate the problem which often includes interacting with development. If a problem is determined to be valid, an Authorized Program Analysis Report (APAR) is opened. APARs describe changes required to a particular components and/or modules and include a severity indicating the urgency with which a fix is needed. The fixes themselves were most often provided by the customer service representative.

Tools Support

Each function within the product development cycle is supported by tools. For instance, developers make use of language compilers and library systems which are used to store the product code. Similarly, the build process is very automated. It is the tools group that is responsible for maintaining the tools set for use by the various functional areas.

Marketing

During this period, IBM was known for its outstanding marketing and sales channels (aka the "Blue Suit" channel). This group exemplified customer commitment and as such interacted very closely with the customer in installing, maintaining, and planning upgrades to the system. While very customer focused, this channel has traditionally operated in an outbound manner. That is, their primary focus has been on delivery to the customer as opposed to seeking of latent needs and requirements. During the hardware push era, the marketing group existed as its own division.

Research

The Research group also existed within its own division operating from laboratories in Hawthorne, New York and Almaden, California. The primary focus of this group during the hardware push era was basic research, as opposed to applied research and technology integration.

Process

New releases of the MVS operating system were generally delivered in 2 year intervals. Releases consisted of a series of functional deliverables known as "line items". Originally, line items were confined to a single component*. Release content was largely arrived at through individual designers defining items based on the needs of the components for which they were responsible (which again directly correlated to their competency). Subsequent management of the release process was performed by the Project office. One former release manager indicated that during this period resource was abundant and release cycles were long enough that release planning was generally not a problem.

While user groups, customer requirements channels, and direct interaction provided the basis for some line items, the overall software objectives were largely influenced by the hardware design organization. A senior software designer states, "Support back in those days usually consisted of the hardware guys informing us what new functions we needed to provide software support for - whether it was a large change such as the extended addressing support added by the 370-XA architecture or a minor item such as an enhancement in support of a single new instruction." In effect, the software organization's mission was to provide abstractions which would allow subsystems and customer written application software to leverage new hardware capabilities.

The process of performing detailed design for a line item was shared between the design and development organization. Designers would create an Initial Programming Functional Specification (IPFS) describing the objectives and proposed content for a line item. IPFSs provided the basis for the creation of more detailed design documents known as Final Programming Functional Specifications (FPFSs). FPFSs generally described the architecture of the affected component(s) (i.e. the subcomponents and often module structure). Creation of the FPFS was generally performed jointly by a designer and key members of the development teams. FPFSs were largely text based and often consisted of over a thousand pages of documentation. While the degree of overlap within the decomposition process between design and development varied based on factors

* It should be noted that over time, line item definition has evolved to be less component based and more functional in nature. IBM's chief architect attributes much of this to the fact that the increased focus on software has lead to more complex solutions (as opposed to direct support for discreet hardware functions).

such as personal relationships and experience levels, it most typically took the from depicted in Figure 11.

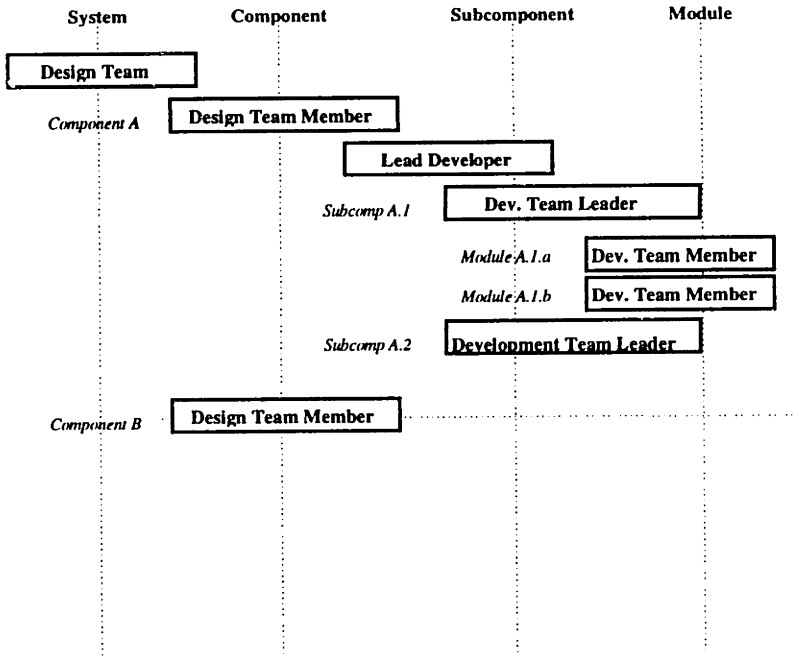


Figure 11. The decomposition process is jointly performed by Design and Develop team members.

In addition to the specification documents that were maintained in a library under the control of the project office, development teams often maintain component notebooks which are updated over time. Key within these notebooks is the inclusion of design rational sections. Design rational captures the thought processes and tradeoffs that are made as the component evolves over time. Component notebooks are generally kept in files on the main development system. The files were managed by members of the development team and not shared among components. Similar to FPFSs, component notebooks were largely text based due to the lack of graphics packages available during this period. It should be noted that there are no specific requirements for component notebooks. As such, the degree to which the notebook is maintained is often reflective of the team. One team produced a notebook containing 500 pages of documentation while another indicated that it relied mostly on the FPFS, the modules, and the naming conventions for understanding intra-component relationships.

The final and most accurate source of information regarding a component is in the modules that comprise this entity. Just as there are strict conventions for naming modules to be included in the system, there are also strict rules for documentation. For instance, each module contains a standardized comment section, known as the "prolog", containing information about the module such as its function, the services it uses, and requirements of modules that are to use the program. It is a requirement that all modules adhere to the documentation standard. It should be noted that while comments and source code document a module and its environment, it is often hard to detect the subtleties of its interaction with other modules within the system. This implies that updating a module within the system often requires implicit knowledge of its environment outside of what can be shown in the documentation.

The development process employed by the System/390 organization during this period could be characterized as adhering to the "waterfall" model. In this model, completion of each step in the process is required before proceeding to a subsequent step. For instance, coding does not begin until after the design has been thoroughly reviewed. Similarly, Function Component test would not begin verifying a line item until coding was complete.

While there was originally very little overlap across functions (i.e. development, test), development itself proceeded in an iterative fashion. Specifically, each release contained a series of development "drivers". Drivers provided a mechanism for integrating new and changed modules into the product. As developers completed their coding, they would transmit their modules into the driver. On a periodic basis (typically, every 4-8 weeks), the build group would create a version of the product containing the new code. When all the code for a particular line item had been integrated into the driver, Function Component test would begin testing. As more line items became available, system test would engage in the driver verification process. Over time, the waterfall process has evolved to include more overlap among functions.

While thorough and rigorous review of the design early in the process is certainly desirable, it does not prevent changes from being made throughout the development cycle. Inevitably, modifications to the design were required based on discoveries that occurred as part of the development process. Design changes were requested through submission of a Design Change Request (DCR) form and were approved by a board consisting of members of the design team as well as project management

personnel supporting the affected release. Similarly, problems found within the test phases were reported via opening a Program Trouble Memorandum (PTM). PTMs were assigned a severity indicating the urgency for which a fix was required. High severity problems forced components to be immediately returned to development for modification.

A final activity within the process included service transfer education. Service transfer education consisted of the development organization providing formal training to the customer service teams describing the infrastructure of the new and changed components within the system as a result of the new line items. The education was generally provided on a component by component basis during a two to three week session near the end of the product cycle. Figure 12 depicts the role of departments and functions in relation to the product architecture and process flow.

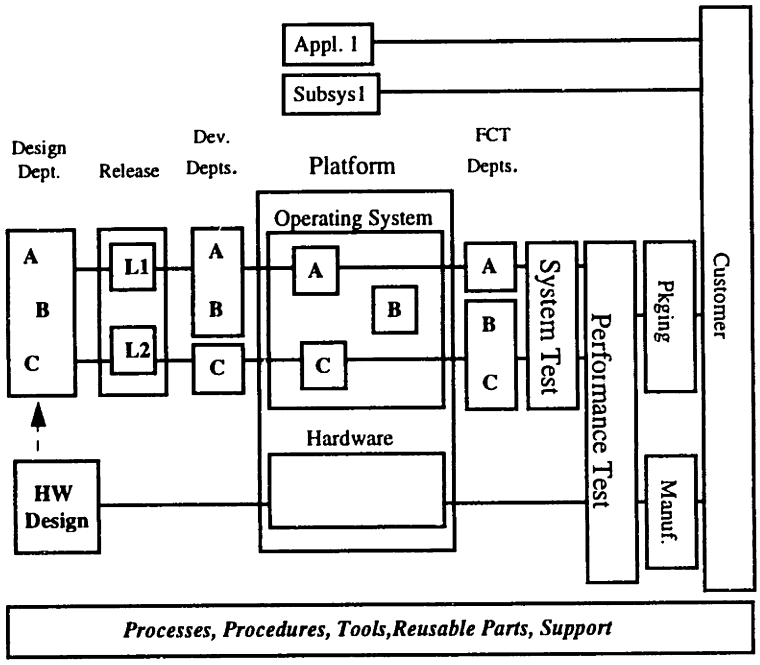


Figure 12. The hardware push era consisted of a waterfall development process with minimal overlap in activity.

Over time, organizational and process changes were instituted in an effort to improve efficiency. In the late 1980s, the Component test function was combined with the development organization. Departments within the organization continued to reflect product component structure within a specified function. While testers were merged into development departments, they continued to maintain their functional roles as testers. That is, development departments now consisted of

coders and testers. While some persons move between the roles, the majority of persons to this day maintain their original discipline.

While customer service personnel maintained responsibilities for interacting with customers and performing initial diagnosis of problems, the responsibility of fixing those problems was moved back to the development organization. The primary motivation of this change was to ensure that one group (i.e. department) maintained responsibility for all changes to the modules within a component. This allowed for consistency within the code and logic while reducing the amount of work required to merge changes from multiple sources into the development stream.

Software Platform Push

The Software Platform Push era describes a period when the focus of the System/390 business transitioned from being hardware-driven to being software-focused. While hardware sales continued to provide a majority of the revenue, software (and more specifically - application software) was beginning to play an increasingly prevalent role in the purchase decision. The industry focus during this period was on the emerging desktop and open systems market. These occurrences were not originally viewed as relevant to the System/390 business.

Many factors provided the impetus for this movement. The most ironic of these was that IBM, with the world's largest commercial computers, was rapidly approaching the limits of its bipolar technology. In particular, IBM engineers realized in 1984 that customer workloads would be able to exceed the capacity of its largest single machine in the near future. This would present a significant challenge to large customers as they would no longer be able to address capacity issues by incrementally upgrading their mainframe to include more processing power. Specifically, customers exceeding the limit of the existing machine would not only be forced to buy a second machine (representing a significant capital expenditure along with significant increases in facilities costs) but would also be forced to segregate their existing workload across the two images. The latter fact was perhaps the most alarming as it would represent significant labor costs and disruption to existing assets in the form of business logic (i.e. customized programs developed by the installation staff).

The "Tivoli"* task force was formed in 1984 with a mission to create a scaleable system that was compatible with existing customer workloads. The task force was comprised of engineers from hardware, software, and research. The group selected a concept consisting of a new hardware facility, a Shared Expanded Storage (SES) Facility, which would serve as the mechanism for coupling multiple mainframe computers together to be operated as a single image. The ability to seamlessly support existing workloads would be possible through upgrading the IBM subsystems and the MVS Operating System software to exploit the new capability transparent to the users of their interfaces. Effectively, usage of the new facility would be performed behind the abstraction barriers that were used by most customer written application software.

In 1986, a breakthrough within the bipolar technology allowed hardware engineers to overcome the previously identified constraints causing the technology to not be viewed as an inhibitor for the foreseeable future. As a result, the Tivoli task force was disbanded. Convinced of the value of the concept, however, the software engineers continued with the design. In particular, this concept was championed by a chief architect within the Software Design organization.

The concept derived from the Tivoli task force emerged to become the world's first commercial clustering technology known as "Parallel Sysplex". The technology served as the basis for a new generation of the MVS software platform. In addition to the sysplex technology, several interesting events took place as a direct result of the sysplex project. These included:

- Formation of Design Councils

  Parallel Sysplex was recognized early as a major undertaking and one that would serve to extend the logical system boundaries. IBM subsystems which were previously managed as derivative products outside the base were now viewed as key in enabling this technology. Recognizing the need for increased and focused communication, a Software Design Council (SDC) was formed. Consisting of approximately twenty five key designers and research employees from across the System/390 division, the council convened on a periodic basis for a

* Please note, that Tivoli represented an internal IBM code name and is no way associated with Tivoli Systems, Corp. which has recently been acquired by IBM.

week long series of focused design sessions. Generally, these were held on a 6-10 week basis. The location of the council rotated among the IBM sites of the council members.

While the original mission of the SDC was to provide a vehicle for design of the new generation platform, it has since grown into the body directly responsible for establishing the technical direction of the platform. Today the SDC includes approximately 40 members and meets on a six week basis.

A similar hardware council was also formed consisting of technologists from both the hardware and software development labs. The hardware council was originally driven by the momentum and change created by the new platform movement.

Finally, a customer design council was formed. The council allowed key decision makers and IT personnel from large corporations to participate in the evolution of the parallel sysplex design. Effectively, this group represented the lead users of the new technology. The Customer Design Council continues to exist today. Similar to the internal design councils, the agenda consists of a week of focused sessions to disclose and receive feedback on current platform plans. The group meets twice annually. In the interim periods, customers agree to provide input within a specified response time.

- Joint Customer Study prototype

In order to prove feasibility of the cluster system design, a joint customer study (JCS) was performed with the AllState Insurance Corporation. The JCS consisted of creating a model of the AllState production workload at the IBM research facility in Hawthorne, NY. Once created, this environment served as the integration point for much of the early sysplex testing. It should be noted that a significant amount of the sysplex project was prototyped by the IBM research group. The JCS experience was valuable not only in the fact that it exposed flaws within several of the original sysplex components but also in the fact that it served as a shared learning vehicle for customers and IBMers.

The JCS illustrated the value of early customer input and resulted in the formation of formalized beta programs for MVS customers as part of the normal development process. Unlike Microsoft, whose beta program consists of thousands of customers, these programs are more controlled and involve a much smaller set of customers and IBM internal sites.

- Enhanced Integration Process

The sysplex project resulted in an additional change within the System/390 organization and development process. Specifically, the extension of the system boundaries to include subsystems resulted in the formation of a "solution" test organization. Solution test was responsible for performing an integrated test of the base system componentry and key IBM subsystems. This testing is performed at the Poughkeepsie, NY site which has evolved to the role of "System House" for the development of System/390 offerings. It should be noted that while solution test provided a vehicle for integrated product testing, these entities continued to be marketed as separate products causing customers to perform a similar integration at their site.

- Transition to CMOS technology

During this era, bipolar hardware technology was replaced in favor of new Complimentary Metal Oxide Semiconductor (CMOS) technology. While CMOS did not perform as well as the existing bipolar technology, the ability to cluster multiple systems together allowed this to be a feasible alternative. CMOS offered significant development and operational cost advantages, as well as, providing a new technology curve which would allow the bipolar capability to be matched in the near future. It should be noted that the new CMOS machines were completely compatible with existing systems and software due to strict adherence to the System/390 architecture.

It bears mentioning that the transition to CMOS technology also represented a movement of the majority of the hardware development mission from the Poughkeepsie, NY development lab to the IBM lab in Boeblingen, Germany. This resulted in major resource reductions within the Poughkeepsie site.

Figure 13 depicts the APDM as a result of the parallel sysplex project. The framework is able to highlight the extension of the logical system boundaries along with the corresponding extensions to the System Level Design process (i.e. via the Software Design Council, Hardware Design Council, and Customer Design Council) and the integration process (via expanded scope of back-end test groups).

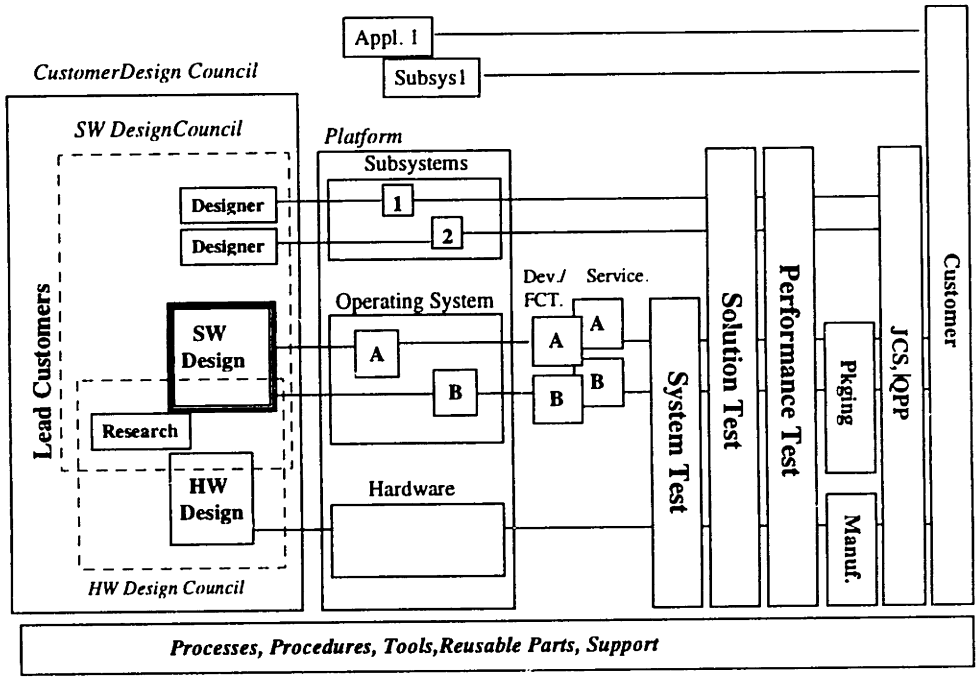


Figure 13. Parallel Sysplex introduced process and organizational changes, as well as, a new version of the software platform.

In addition to the previously described benefits of the sysplex project, the most prevalent is the fact that is served as a vehicle for organizational "bonding" across the divisions. Prior to this event, each of the product areas had operated primarily as independent entities. While gaining initial consensus across the groups took a significant amount of initial "selling" by the chief architects, the project proved very synergistic once it had begun. One architect noted that several new ideas and differentiating enhancements spawned from the focused work of this group.

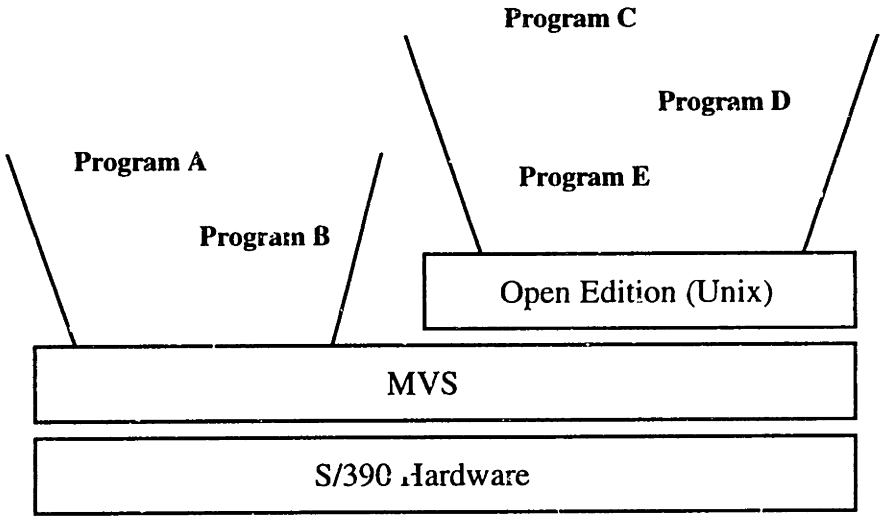
If sysplex provides a good example of planned renovation of a product platform, the second key event within this era provides evidence of System/390's poor linkage to the industry as a whole.

While the IT industry in the late 1980s was already experiencing the impact of the powerful desktop PCs and workstations, it was also hit from a different direction by the growing acceptance of Unix and open systems. As a result, many new key business applications were being developed using the Unix APIs.

"In bidding to retain its position as a contractor for the multi-billion dollar National Aeronautics and Space Administration (NASA) space station contract, IBM had to incorporate support into MVS for popular UNIX interfaces to comply with requirements handed out in Federal Information Processing (FIPS) document 151. Failure to comply with FIPS meant NASA would have thrown out MVS along with millions of dollars of IBM hardware that it ran on"[33]. The result of this was the OpenEdition project.

OpenEdition represents yet another abstraction upon the base system. This abstraction, however, encompasses support for the entire set of interfaces for a Unix operating system. Effectively, programs written on Unix systems could now be "ported"* onto the MVS system and executed using the OpenEdition APIs. Through supporting the Unix programming interfaces and user facilities, MVS had effectively expanded to become two systems in one. Figure 14 illustrates this concept.

* Porting refers to the process of taking the source code for an application (for instance that is written in the C programming language) to another platform and compiling it using a similar language compiler (in this example compiling via the C Compiler). The output of the compiler is able to be executed on the target system.



b

Figure 14. OpenEdition added a second personality to the MVS system through addition of UNIX APIs.

An initial assessment of the OpenEdition effort using the current software development process and resulting productivity numbers as a base indicated that it would take 40 programmers 8 years to complete the task. Neither the time nor development resource was available to facilitate this size effort. As a result, OpenEdition was allowed to operate outside the normal process through the use of an autonomous team (aka "Tiger" team) model.

Team membership included 10 developers with MVS skills, as well as, approximately 20 persons with no previous background in MVS programming. The team was located at a facility in Kingston, NY (approximately 20 miles from the main Poughkeepsie development site).
The choice of the location was jointly influenced by a desire for isolation as well as the fact that the 20 non-MVS programmers were already located at the Kingston site. It was also mutually agreed that speed was the primary objective of the project. As such, the OpenEdition team communicated minimally with the Design organization. Key within the formation of the team was the choice of its leader - a senior developer with years of experience within the development of the base operating system.

Many challenges were faced by the team. One such task was to quickly gain an understanding of Unix standards and what it would take to implement them. Because Unix systems were developed in a culture very different from that of System/390 commercial systems, many of the concepts and resulting usage patterns appeared foreign to the team members. One team member commented that "Unix just smelled different". The team leader indicated that this was overcome by not only reading a lot of books and specifications but also by applying learned concepts immediately (i.e. "learning by doing"). Additionally, two Unix consultants worked with the team during the ramp-up period. One consultant was from the IBM Research division while the other was an independent contractor.

Noted as a major source of efficiency within the development of the project was the fact that the team was forced to spend very little time understanding and formulating requirements because they were already defined by the Unix X/Open standard*. Given that, the underlying subcomponent structure was arranged to map as closely to similar models within MVS. The direct mapping approach was desirable in that it allowed for maximum reuse of base system componentry performing similar functions.

Five development teams were formed - each responsible for producing one or more of the defined subcomponents. In addition, a test team was formed. Teams consisting of functions which were more closely related to the base system were lead by persons with a background in that area (i.e. the MVS programmers). Teams implementing functions less related to the base system were more heavily populated with non-MVS programmers. The team leader was responsible for the overall development and as such worked across all teams in a consultant fashion. This included providing just-in-time education as well as reviewing designs and code.

With a few minor exceptions, all teams were able to make use of the development tools that existed for use by the main Poughkeepsie lab. Teams also operated in an iterative fashion as opposed to a waterfall method. An example of this was the "signal processor" implementation which went through six iterations throughout its development.

---

*A consortium known as the Open Group owns the Unix Specification. Their X/Open Specification 1170 (aka Spec1170) was the defining Unix standard during this period.

OpenEdition was delivered as a non-priced feature of MVS in 1991. A developer noted that the follow-on release, two years later, "really marked the first release in which you were able to do anything useful". Finally, in 1995, OpenEdition achieved full Unix branding. Figure 15 depicts the OpenEdition component in relation to the existing system componentry. Note that the model is able to reflect Spec1170 as the driving design principle. It should be further noted that Open Edition was not viewed as part of the base at this time by the architecture team.

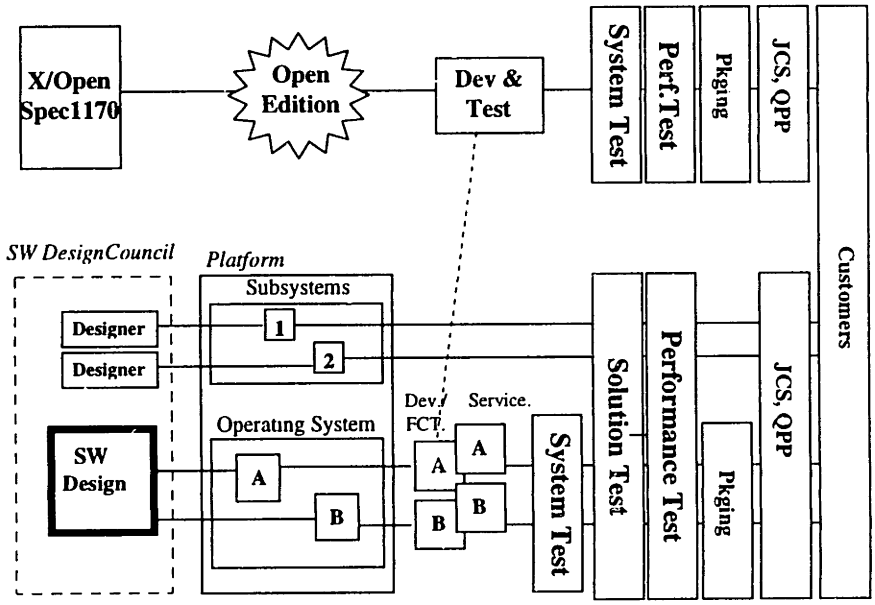


Figure 15. The OpenEdition component was developed within an autonomous team structure in Kingston, NY.

It should be noted that the OpenEdition group was eventually merged back into the normal development structure in 1995. This presented several problems as the team had adopted a culture and development model that was very different from that subscribed to within the base group. In particular, several conflicts occurred as a result of the OpenEdition group "breaking the driver". Where the iterative and fast mode of development had been acceptable within an isolated environment, this caused problems when the consequences of errors affected other development groups.

Software Solution Push

The Software Solution Push era describes a period fueled by an intense pursuit of growth. While the advent of Parallel Sysplex had driven new demand for the System/390 hardware, it had not done so in a manner that was able to offset declining hardware prices. Growth was thought to be achievable through the creation of new applications or solutions.

Several organizational changes were instituted to facilitate this effort. First, a new role was introduced within the System/390 organization -- the solution manager. Solution managers existed within their own department and served as heavyweight project leaders who were responsible for all phases of a solution including concept definition. Concepts varied - ranging from development of new products predicated on the base system infrastructure to porting of products and technologies from other IBM platforms into the OpenEdition environment. Solution managers had incentives that were linked to market performance of their product.

The second organizational change consisted of creating a Solution Development group that would act as a job shop from which solution managers could recruit resources. The Solution Development group was staffed from various sources including the base system development group. Due to the movement of much of the hardware development function outside of the Poughkeepsie area, many engineers were recruited from the hardware area and trained to be programmers.

Concerned with the possibility of not having sufficient resource remaining within the base organization to maintain component ownership responsibilities and perform new development, the base group was divided among two teams. The first team was responsible for the development of the current release while the second team was responsible for performing service on existing releases and beginning the design of changes in support of new releases*.

An investment review board consisting of top managers was established to allocate funds to solutions. The urgency with which this effort was undertaken resulted in an assortment of proposals being submitted within a relatively short (1 year) period. Effectively, the solution movement took the form of a competition with the top projects receiving funding. Figure 16 provides an APDM reflecting the results of the solution movement.

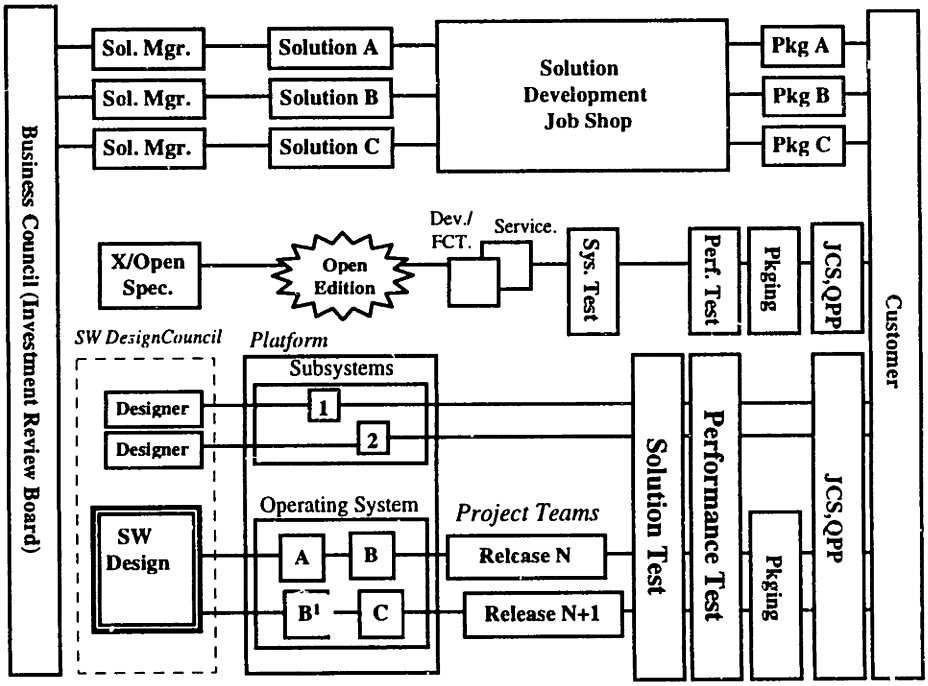


Figure 16. The solution movement introduced project based teams to the organization.

Overall the solution movement did not achieve the desired results. In particular, it was noted that this movement was predicated on market opportunity as opposed to "realizable" opportunity. Effectively, solution management teams identified market opportunity and were often able to propose a product to fill the perceived need. However, the product concepts were still very much representative of a "push" mentality with very little actual customer validation. As a result, products that were able to be created often were found not to meet the expected customer wants - or at least not to the degree that would command the projected price.

Many of the solution projects never left the prototype stage. Others did, but were found to perform insufficiently due to the immaturity of the OpenEdition environment and the primitives within it. Additionally, products that successfully exited the development phase often found that the existing channels were not sufficient for delivering the product and the required marketing messages.

---

* Effectively, the group was segregated into a Release N and a Release N+1 team. The Release N+1 had the added responsibility of providing service for existing components.

While the intent of the release teams for the base components were to allow for efficiency through focus, this was never achieved. The twin team philosophy was abandoned after one release. A tester within one of the key system components indicates that this caused problems because while new and updated components came "home", often the persons who had developed and tested them went to other areas. This left base groups with ownership of componentry with which it had no familiarity. In effect this was a worse situation (as it included re-merging an entire development stream worth of changes rather than a fix for a single bug) than the one that had originally provided the impetus for moving customer service responsibilities into a single unit.

Also exposed during this period was a deficiency within the development tools and support processes for these new environments. A notable example comes from a project that was forced to invent its own methodology for building its product because the existing tools and processes did not support modules created in programming languages other than PLAS or Assembler. The process created by the group resulted in taking two weeks to build the product. The same task is able to be performed in a matter of hours within the base environment. A member of the research division who is responsible for facilitating technology integration into the product areas indicates that tools and environment support may be the biggest inhibitor to initial technology integration.

The solution era came to a close with the elimination of the Solution Manager structure. Several artifacts of this era remained, however, in the form of products and projects that had been previously established. While the organization was returned to its previous component based structure, the system now consisted of more components (in the form of the products and technologies that had been introduced in the solution era). Additionally, a new design department was formed consisting of the lead technical persons from the various solution teams. Much of the responsibility for integrating the various products was returned to the software design council.

While the design council had been successful in the past at establishing technical direction which served to drive the business, similar success was not immediately realized with regard to stabilizing the solution situation. Integrating the solution members into the Software Design council resulted in an initial mismatch in backgrounds and competencies. In an effort to not only share information within the group but also to help educate the business council members, the council decided to create a series of strategy white papers. While the papers were able to communicate high level

product objectives and potential value of each technology, they did little to initiate the work required to adjust the base system to most efficiently support those technologies.

Just as the organization was becoming overwhelmed with the number of new technologies and products, so to were the customers. While the organization had successfully created in-house structures such as design councils and solution test organizations to support the decomposition and integration process in response to extending system boundaries, it had done little to insure customers were similarly sensitized to these effects. Installation of an MVS system required customers to install as many as 37 separate products. While integrated solution tests help to insure that the configuration was viable, it did little to prevent customers from encountering errors in the integration of this into their installations. As a result of these problems, MVS and its subsystems were re-packaged into a product known as OS/390. OS/390 allowed customers to receive a pre-built package of the integrated subsystems. This served to formally recognize the system boundaries not only internally but externally. In addition, an OS/390 release would be delivered every six months to allow customers to more effectively plan their migration scenarios. Figure 17 provides an APDM describing the system and organizational linkages during this period.

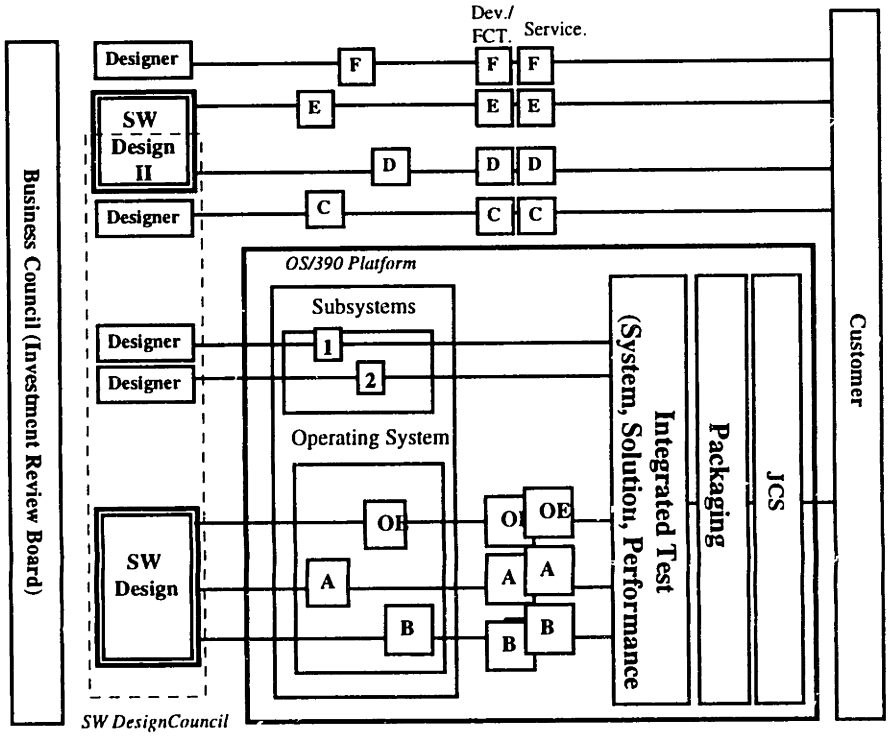


Figure 17. Technologies and products from the solution era were merged back into the functional organization.

Software Adapt, Shape, and Partner

In an effort to become more focused and avoid the mistakes of the past, the System/390 organization has moved to an Integrated Product Development (IPD) Model. Similar to the Investment Review structure that previously existed, the IPD model calls for an Integrated Portfolio Management Team (IPMT) which has responsibility for insuring funds are being directed in the appropriate direction to meet business objectives. Differing from the former model is the fact that the IPMT is linked to consultants existing within thirteen Industry Solution Units (ISUs). Industry Solution Units represent divisions within IBM containing consultants to industries such as Finance, Petroleum, Insurance, etc. In providing solutions, ISUs are not restricted to using IBM technologies and products.

In response to direction from the ISUs and larger corporate strategies, the IPMT commissions individual Product Development Teams (PDTs) to investigate concepts. PDTs are cross-functional teams that are lead by a heavyweight project manager -- the PDT leader. PDTs are responsible for the entire lifecycle of a product including initial concept. The PDT leader forms a core group of team members that will remain with the product throughout its lifecycle. Additionally, the PDT acquires the help of functional areas on as needed basis (for example, additional marketing persons are added to the team during the go-to-market stage). Development resources are maintained in a pool controlled by a program manager. These resources become "contracted" out to PDTs. Once formed, PDTs progress through a series of checkpoints toward product launch. Unlike the previous era, checkpoints provide a mechanism for continual review of investments allowing projects that are not meeting their goals to be terminated.

It should be noted that Design organization members also work informally with PDTs. In that role, they are not required to perform development but rather act as "consultants" in determining opportunities for subsequent harvesting and integration of technologies from new products into the base system. Figure 18 depicts the PDT structure during this period.

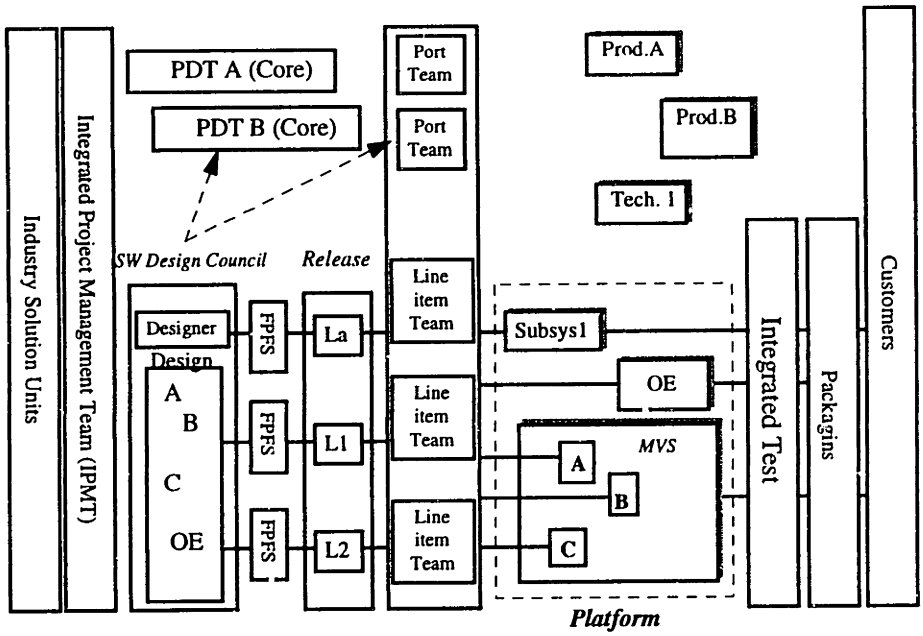


Figure 18. Design team members serve as consultants to PDTs.

Summary of Case Data

The case data has shown that the System/390 business unit experienced great changes during the 1990s. During that time, the division emerged from a position of competing for its existence (i.e. battling the "mainframe is dead" proclamations within trade press) to a position of being a viable and increasingly preferred platform choice for the deployment of network computing applications. Two items were key to this resurgence:

1. Adherence and attention to base architectural principles and beliefs
2. Extending the base system to support a more pervasive programming model

To its credit, the System/390 division continued to execute its strategy of creating a new platform generation throughout the troubled period of the early 1990s in which the industry focus was far removed from host-based technology. While the underlying technologies (i.e. Bipolar vs. CMOS, etc.) changed throughout the emergence, the base architectural principles remained constant. This strategy and adherence to purpose is now paying dividends as the new parallel sysplex architecture

has provided a scaleable, clustered system platform structure that serves as a potential base for next generation computing systems.

The foresight and understanding of large scale system requirements by IBM's engineers is evident in the fact that, 7 years after the creation of parallel sysplex, clustered systems models are becoming a focus within the industry*. Most notably, system providers like Microsoft and Hewlett Packard are producing clustered systems in an effort to evolve to support enterprise computing environments. If System/390 had not adhered to its strategy and strengths during this period, the results may have been devastating.

Also noted in the case is the fact that the system (almost entirely by accident) was extended to provide support for Unix applications through the creation of its OpenEdition component. While originally viewed as an ancillary effort to satisfy requirements for a single government contract, OpenEdition has since evolved to be an entry point for bringing new technologies and products onto the System/390 platform. One senior manager recently stated, "OpenEdition has proved to be a success technically. But, perhaps its biggest contribution to System/390 is that it has helped overcome some of the stigmas associated with mainframes." In short, OpenEdition has helped to prevent System/390 computing systems from being disqualified from consideration for application development and deployment before decision makers reach the point of realizing classic strengths provided by the platform. It was also noted that the development of the OpenEdition component was performed by an isolated autonomous team having little interaction with the base system architecture team.

While the increased demand for Unix style applications executing on MVS has to be viewed as a positive, it also brings with it a new set of challenges. Specifically, customers have begun to not only desire, but expect, applications executing within the OpenEdition environment to seamlessly inherit all of the classic strengths of mainframes (i.e. reliability, scalability, etc.). While it is true that there are inherent advantages in executing within this environment, new programs are not currently as well positioned to exploit base technologies in the same manner as the applications for

* While the lead engineers were key in providing the vision and leadership, all involved with the effort should be commended as this represented a time of great turmoil within IBM and the global economy. The effort required to complete this effort is especially notable considering the environment it was performed in. This is a tribute to those individuals as well as an indication of the deep culture that existed within the "former" IBM.

whom they were originally developed. Evidence from the solution era indicates that not only are there deficiencies within "new style" products, but that the pressure to proliferate these products and grow the business can lead to loss of focus within the organization.

Given this, the challenge for System/390 (and particularly its architects) is not only to find a method to incorporate new technologies into a stable architecture but also to insure that where appropriate those technologies are synergistic with the existing technical and organizational assets

CHAPTER 5. A MODEL FOR EVOLVING MATURE PRODUCT PLATFORMS

Utterback states that a strong technological base is as critical to the prosperous survival of a firm as a good understanding of markets and strong financial position and that meeting the challenges of discontinuous change requires the firm not only to be aware of its own vulnerability but also to make organizational adjustments that facilitate successful competition with an invading technology.

Implicit in the above statements is the need for firms to be flexible. Flexibility entails many things. First, it assumes that the platform and/or the derivative product architectures are able to be enhanced in response to market events. Encompassed in that statement is a requirement that the organization itself not only be positioned to quickly execute development projects, but that it should also contain the necessary communication channels which allow it to evaluate and anticipate change in relation to external events. Additionally, the structure should facilitate the ability for the organization to learn and evolve with its environment.

In pursuit of these objectives, a model is proposed for the System/390 organization that will allow the firm to maintain competency in existing areas while increasing skills in the delivery of new technologies. It should be noted that the structure is predicated not only on the product architecture but also with regard to existing (and necessary) project flows, and cultural considerations. The structure is driven by evidence from the case data along with lessons from the existing literature. Much of the model reflects actual events that are currently in place as System/390 continues to evolve and adjust to this new environment.

Key within the model is a new position within the organization known as the "Level II Architect". The term "Level II" is derived from a concept formed from this analysis known as "leveling". Leveling describes the pattern of innovation that is currently being used by the System/390 organization in the adoption of new technologies. While the concept of leveling is fairly primitive, the analysis will show that there are subtleties within this process that can inhibit the fusion of base and emerging technologies. It is the intent of the Level II architect to lessen the impact of leveling.

The basis for the model was conceived from asking the following questions:

- What is the firm's current process for creating shaping technologies ?
- What is the firm's current process for adopting technologies from other sources ?
- Where are these processes and technologies complementary and/or in conflict ?

The questions are formulated based on observations from the case data. Specifically, the case study was able to highlight the fact that the patterns for creating shaping technologies and for adopting technologies from other sources were radically different, hence providing the impetus for questions 1 and 2. Question 3 is included to prompt a discussion regarding the inhibitors preventing the fusion of new technologies from outside sources with the technologies created by the firm.

As a prelude to discussing these issues, it is first necessary to understand the options for platform innovations that are available to providers. The next section provides a set of terminology and a model for describing these innovations. The model and its resultant terminology will be used in the subsequent discussions.

Platform Enhancement Models

As a first step in creating the model, terminology is formulated to describe the components of an existing product platform in relation to new technologies. In describing technologies, Adler and Shenhar suggest that a dimension of aggregation that has proven particularly useful is the distinction between base, key, pacing, and emerging technologies. Base technologies are those that are common to everyone in the industry. Key technologies are those that provide competitive advantage at a particular point in time in the industry. Pacing technologies are those that while not currently deployed in the industry, can reasonably be assumed to have the potential to displace one of the key or base technologies. Emerging technologies are on the horizon. This model proposes the following variant of this set:

- **Base Technologies**

  Base Technologies reflect non-distinctive technologies which are provided by the base system configuration. In software platforms, these represent the APIs available to users of that system. For example, Base technologies within Windows95 systems would include the Windows development (Win32) APIs.

Base technologies exist within a Base platform. In the model, a Base platform containing specific technologies will be assumed to be the reference point for any innovations proposed by the platform provider.

- **Key Technologies**

Similar to the earlier definition, key technologies designate technologies that provide immediate competitive advantage for the platform. For example, the Parallel Sysplex technology provided by System/390 would be considered a key technology for that platform.

- **Derivative Technologies**

Derivative technologies are those technologies that were originated on the Base Platform. An example of a Derivative Technology would be the IBM subsystems that were developed first for the System/390 and continue to evolve with the platform.

- **Leveled Adapters**

Leveled adapters represent abstractions that allow base platforms to emulate the functions and facilities of a non-native technology. Many levels of adapters can be present upon a platform. OpenEdition is an example of a level -1 adapter on the MVS system. Higher level adapters are those that exist upon another adapter.

Figure 19 depicts the concept of base, derivative, and key technologies. The base platform provides a set of interfaces which allow derivative products to utilize services and technologies included in the base. For the purposes of this analysis, it will be assumed that the cost of development for a derivative program is roughly equivalent to the number of APIs used within this program. That is, if derivative D1 invoked one hundred APIs from the base and derivative D2 invoked 50 APIs, we would assume the initial cost of developing D1 to be double that of D2.

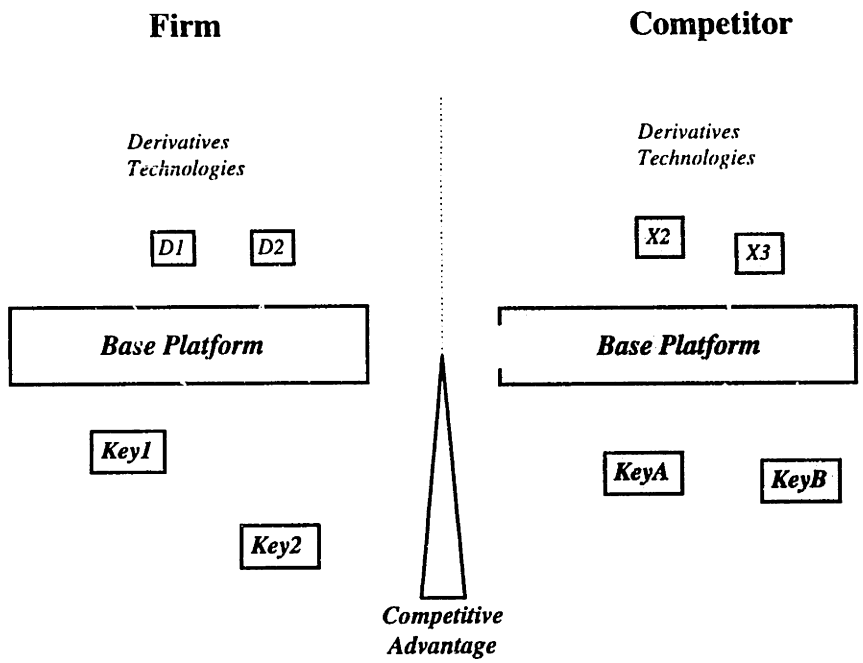


Figure 19. Derivative technologies make use of services provided by the base technology platform.

Some of the technologies provided by the platform may be key technologies. As such, they provide competitive advantage. Within this model, the notion of a technology meter is used to reflect the degree of advantage a particular technology provides. In the example provided in Figure 19, key technology, key2, provides more competitive advantage than key technology, key1. This is evident in the model through key2 existing "deeper" on the technology meter than key1. It should be noted that over time key technologies progress "up" the technology meter. That is, as others adopt or displace the technology it may become less of an advantage.

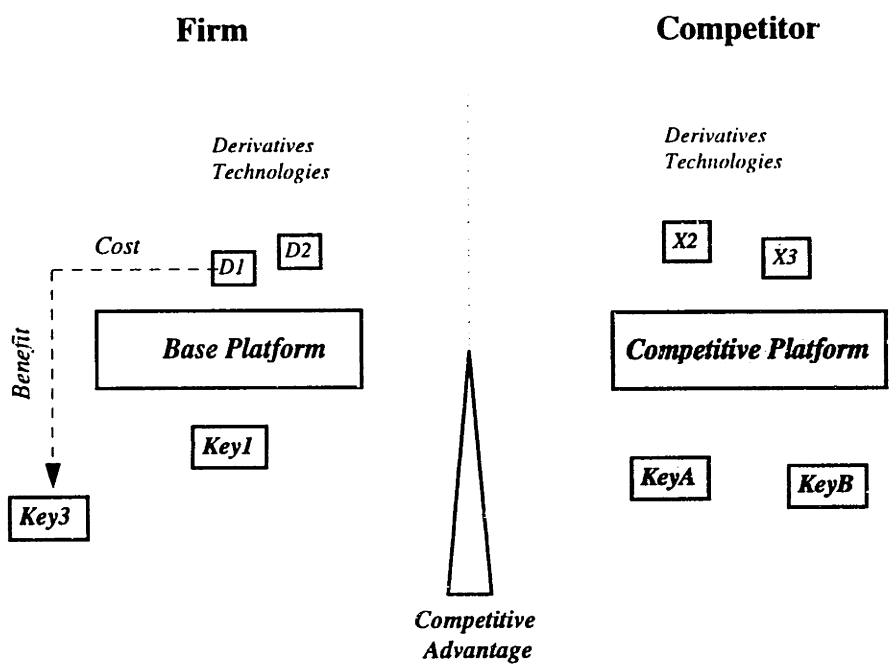


Figure 20. Gaining the advantage of new platform technology often requires changes to the derivative product.

Figure 20 describes the process of adding a new key technology, key3, to the base technology set. While key3 may actually be packaged, sold, and installable as part of the base technology set, the advantage of the functions provided are not realized by a derivative technology until that derivative is updated to "use" the new function. It could therefore be stated that the cost of an existing derivative program leveraging a new technology is equal to the incremental cost of enhancing that derivative to invoke the interfaces of the new technology. By the same token, if new value was provided behind an existing abstraction then the derivatives of that abstraction would receive that benefit at no cost.

The choice to describe key technologies as existing at "depths" which represent their competitive advantage provides a method for visualizing the value equation. Specifically, the value of a new technology to a derivative component could be stated as the cost of exploiting the new interfaces

versus the advantage provided by the technology (i.e. the depth of the technology within the technology meter).

The model depicts competing technologies as existing to the "right" of the current base platform. This implies that the introduction of non-standardized interfaces to a platform can be viewed as having "extended the platform to the left". For example, the application programming interfaces introduced by the sysplex project represent extending the MVS system to the "left" as those interfaces are unique to that system. Similarly, the addition of the OpenEdition interfaces could be viewed as having extended the system to the "right" as it moves the platform closer to the industry standards. The concept of movement helps clarify the actions that a platform provider might take in relation to existing componentry.

In general, it can be stated that software platform innovations can be most accurately described as being coupled along the following two dimensions:

- the degree of technology introduced to the system
- number/nature of interfaces required to exploit the technology

Describing changes along these dimensions yields four basic patterns of innovation. The actions and resulting models for platform evolution are depicted in Figure 21 and are described in the following:

- **Partnered Leverage**

  Partnered leverage describes the pattern where a provider is able to provide new technology for use by its derivative partners. However, in order to realize those benefits the derivatives are forced to use new platform specific APIs. The use of platform specific APIs implies a degree of lock-in with the platform by the derivative product.

- **Platform Leverage**

  Platform leverage describes the concept of providing increased technology advantage through increasingly standardized interfaces. While this strategy is at first reflective of an adaptive strategy, it is actually that of a shaper (or firm that is positioning to become a shaper). In this case, the platform does not achieve lock-in of derivative products through the introduction of

new platform specific APIs, but rather through providing pervasive functions in a manner that provides advantage for the stakeholders.

- **Platform Harvest**

  Platform Harvest describes the pattern of the platform providing increasingly standardized interfaces with no additional technology advantage being gained. This could be viewed as an adaptive move and one that is used to maintain position while searching for a new shaping technology.

- **Partner Reject**

  Partner Reject occurs as a platform owner provides additional specialized interfaces with no technology value. This mode is reflective of poor design and/or as a reaction to problems (i.e. bugs) within the platform. Firms should consider withdrawing from the partnership as a result of this occurrence.

Figure 21 illustrates these concepts.

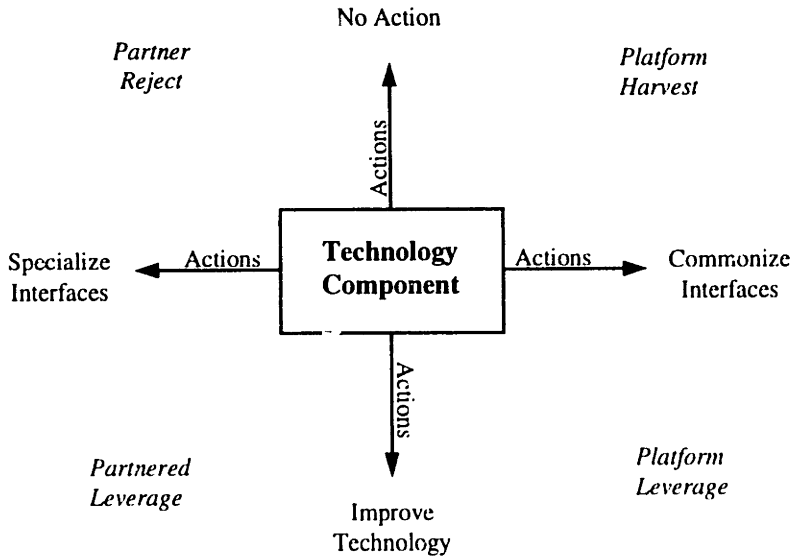


Figure 21. Platform enhancement patterns can be described along the dimensions of interfaces and technology advantage.

While the previous sections have shown strategies for platform enhancements in relation to existing componentry and derivatives, it is useful to extend the model to include the concept of adopting foreign technologies. Foreign technologies are technologies that originate on another platform.

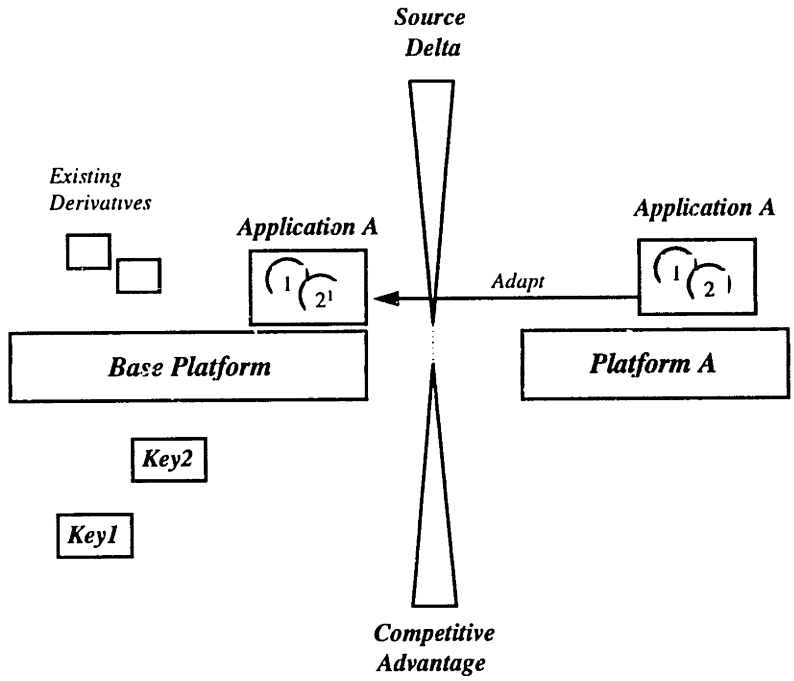


Figure 22. Adopting foreign technologies requires consideration of cost, benefit, and source platform degradation.

Figure 22 provides an example of adopting a foreign technology. In the example, Application A exists on Platform A. As such, the application makes use of the APIs provided by Platform A. In addition, the componentry and infrastructure of Application A has been designed around the interfaces and facilities provided by Platform A. The cost of moving Application A to Base Platform B could be described as the cost to modify Application A's source code and architecture to use the interfaces provided by Base Platform B. The benefits derived from this movement are equivalent to the technology benefit that is able to be realized on Platform B. Admittedly, the cost

of adopting a technology from a foreign platform is much higher than updating an existing derivative as described earlier.

The cost of moving applications between platforms also becomes less feasible when the notion of "source delta" is considered. As stated earlier, derivative technologies are designed to be optimized for their source platform. This implies that an application will typically pay a penalty in performance when it is moved to a new platform. Also, encompassed in source delta is the notion of the development environment. For example, Application A was developed using the tools, facilities, and processes by the Application Development Environment provided of Platform A. Moving the application to Platform B requires a developer skilled not only in the interfaces provided by Platform B but also in its development environment.

The above stated facts are what bring about the need for adapters. As stated, an adapter allows a base platform to emulate the facilities of a foreign environment. The degree to which an adapter is able to achieve equivalency with the native environment is reflected in the source delta. Source delta also has a secondary effect in that the applications executing on the adapter also experience the degradation of the adapter's source delta.

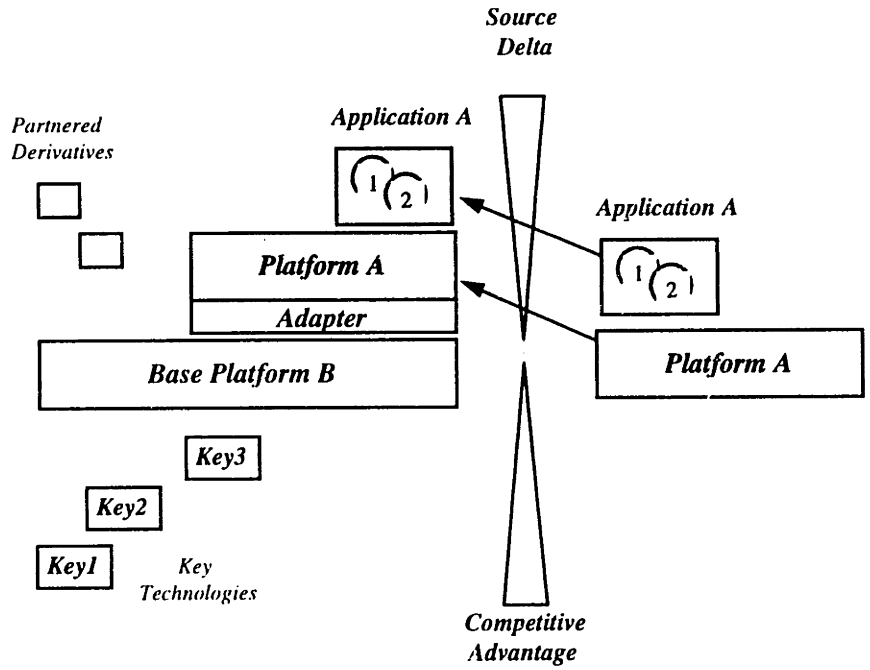


Figure 23. Applications moved between platforms experience a source delta.

Figure 23 provides an example of an adapter. In this case the adapter allows applications developed for Platform A to execute on Platform B without changes. This is certainly more attractive to Application A as it can maintain its current logic and component architecture. In addition, the adapter could also emulate the development environment of Platform A further reducing the source delta. Worth noting is the fact that even though the adapter provides interfaces and facilities mimicking those from other platforms, the adapter itself may be developed using the base platform facilities.

The example in Figure 23 was able to illustrate the process by which adapters can be used to capture applications from foreign sources. Also, shown (although not previously described) was the fact that the adapter leveraged very few of the key platform technologies. Effectively, in providing the adapter, Platform B introduced a Platform Harvest strategy. As stated earlier, a harvest strategy can be subsequently evolved to be one of platform leverage. Conversely the application could gain a technology advantage through explicit use of platform specific interfaces and facilities. This move would be reflective of a partnered leverage strategy. In practice leveraging key platform technologies within derivatives typically involves the participation of both parties. Figure 24 illustrates this concept.

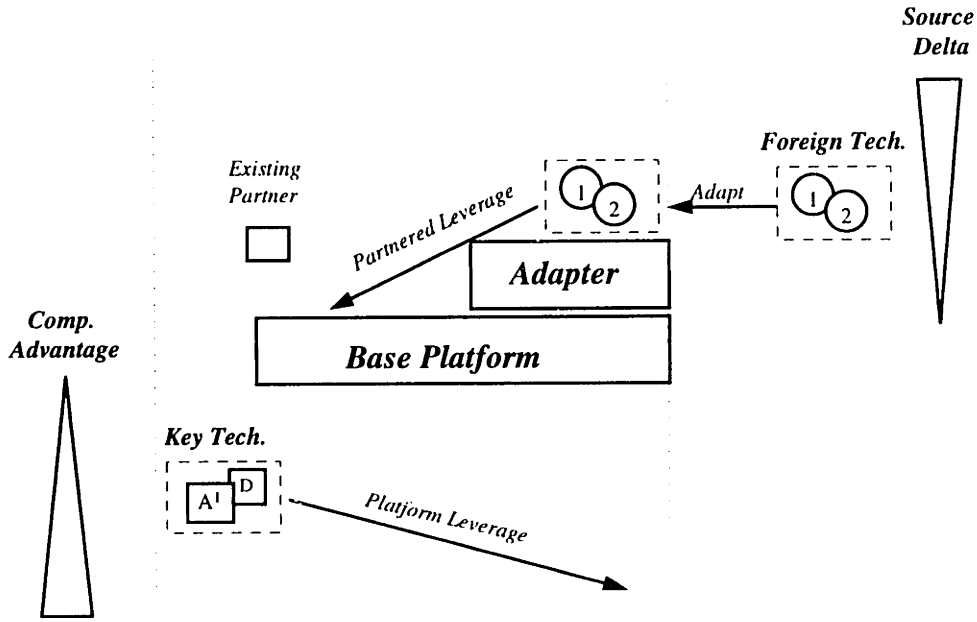


Figure 24. Leveraging of Key platform technologies is most often a partnered activity among derivatives and platform providers.

The previous section provided the basis for a model that describes platform actions. The model is now used as a vehicle for articulating the current method of operation for creating and adopting technologies within the System/390 division.

Inhibiting Technology Fusion -- the Effects of Leveling

System/390 is not currently achieving the leverage that is described in Figure 24. In fact, in some cases the exact opposite is occurring. In particular, the following efforts are underway:

Continued Advancement of Key Technologies

"On the left" remains the design and architecture group who continues to serve as the party responsible for the creation of key technologies. Key technologies possessed by the platform include Parallel Sysplex and its complementary products such as Workload Manager. Rather than adopting a "platform leverage" mode of operation, the key technologies by and large continue to be advanced in a "partnered leverage" manner. With each partnered leverage move made by the platform, the entry barriers for leveraging key technology becomes higher for non-partners.

Support for New Applications

New applications continue to emerge "from the right" in the form of Unix-sourced applications. Generally, the process consists of the initial port of the source code being performed by a porting team followed by a period of incremental performance improvements to the application in an attempt to reduce the source delta of the application. The incremental performance improvements occur through trial and error. In particular, ported applications are able to be executed within the traditional performance test environment which is able to detect "hot spots" within the programs. Where these are recognized, the porting team attempts to change the code.

It should be noted that there have been occurrences of new applications leveraging key platform technologies like sysplex for competitive advantage. These occurrences have all been in the form of leveraged partner moves by the new application. In these cases, personnel from research have been funded to perform this work.

Adapter Improvements

In addition to the continued creation of key technologies and increased proliferation of new applications, there also continues to be significant improvements being made in the OpenEdition Adapter to reduce source delta. These improvements occur primarily in the form of incremental reduction in the overhead of key functions. The guidance for which functions to optimize is driven by data collected from customers who are beginning to deploy production applications in this environment. It should be noted that there is very little focus on enhancing the other constituent part of the source delta equation -- the development environment.

Figure 25 depicts the current mode of operation within the System/390 organization.

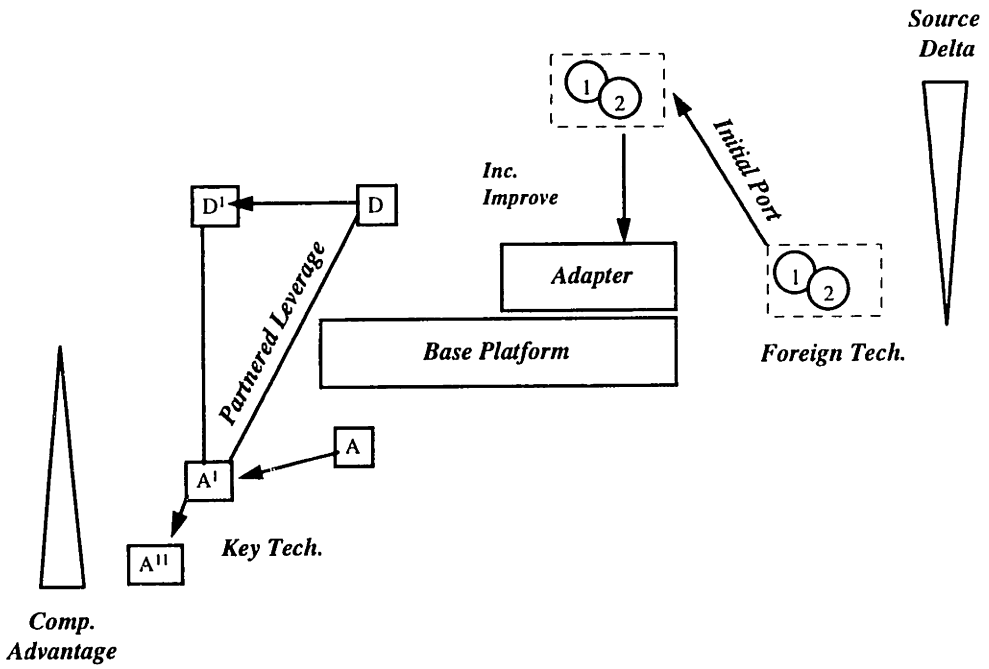


Figure 25. Platform leverage is not being fully realized within the System/390 environment.

Given the currently described state, one could ask "Should System/390 be concerned with its technology base ?" The answer to this question would be unequivocally, "Yes". Of course, all firms should remain focused on their technology base in this fast changing environment.

It is not the technology base however which should be of prime concern to System/390 at this point. Instead, it should be concerned with its problem solving structure.

The author submits that while the System/390 technology base is currently in a period of transition that this problem will inherently stabilize itself. In fact, it is in the process of being resolved through the actions that are currently in place. These actions are able to be relied upon because they represent problem solving strategies that have been proven to work in the past.

It is this last statement with which System/390 should be primarily concerned. While firms should value and leverage their problem solving assets in the creation of future technologies, they should also continue to evaluate these techniques for relevancy. In particular, firms should not only remain concerned with the immediate effectiveness of techniques but, more importantly, how effective their techniques will fair in avoiding future problems. Just as firms are able to recognize and replace their technology as its reaches its limits, similar efforts should be undertaken to update problem solving methodology. Admittedly, this is easier said than done.

One of the key inhibitors to updating a firm's operating procedures is the fact that much of the knowledge base and many of its problem solving techniques are implicitly embedded within its structure. Henderson and Clark have shown that over time the communication channels, filters, and shared problem solving techniques of an organization evolve around the interactions that are critical to its task.

Evidence of this has been demonstrated within the case data. In particular, the system level design process is heavily facilitated by the informal communication channels that have emerged between designers of key system componentry. In effect, these channels provide the vehicle for maintaining the system level architecture for the platform. These channels were shown to be able to evolve independent of geographical location and reporting structures and to be able to be optimized through formal processes such as design councils.

Also noted was the fact that a designer's competency regarding the internals of a component and its current architecture are able to be sustained and enhanced through working closely with the development team - especially in performing the decomposition process as part of the detailed

design phase. Communication channels were shown to exist at all layers within the nested system. In particular, development teams were shown to interact along the lines of the architecture of a particular component (i.e. along subcomponent boundaries).

Shared problem solving techniques also proved to be pervasive within the channels. Evidence of this within development teams exists in the manner in which they produce component specific reusable parts and component level documentation. Problem solving techniques within the designer/developer channel have also been shown to exist and to differ by channel - indicating the fact that these are entities that dynamically evolve over time. At the highest level, it has been shown that the architecture team has adopted a problem solving structure that results in a partnered leverage method of platform innovation.

One only needs to listen to a discussion among any of these entities to appreciate the filters and efficiencies that have been established over time within the channel. Each discussion is not only filled with acronyms but often absent of assumed context that would be required for an outsider to effectively participate. Filtering is also evident in the "subtraction by addition" mentality that has been adopted by many of the development groups. In particular, groups often do not desire the addition of new resource because of the high cost of educating new persons. This cost is proportional to the degree of filtering that takes place within the group. Worth noting is the fact that the design rational that is maintained within specifications attempts to address "over-filtering".

The case has shown System/390 to be skilled in adjusting the boundaries of the platform. Each extension was accompanied by similar changes in structure (either formally or informally) that served to complement the decomposition and integration processes in a manner that allowed the integrity of the platform to be preserved. It was also shown that the organization and technology are flexible enough to encompass an adaptive technology that emulates a competitive environment. Namely, the OpenEdition component and its team were able to be integrated into the base system componentry and the accompanying development processes.

The dynamics of channels, filters, and problem solving techniques in relation to organizations and architectures is embodied in Henderson and Clark's overarching theme of architectural innovation. They state that architectural innovation is a pattern of change in which the core concepts embodied

in a system's components remain unchanged but the interaction among them changes subtlety. As a result, the entities that evolve around these interactions are disrupted. This can cause the application of filters or advocating of a particular problem solving technique across component boundaries where this is not appropriate.

The concept of architectural innovation most prevalently occurs for System/390 across the boundaries of an adapter -- in particular through the OpenEdition Adapter. As stated, an adapter allows the system to provide an entirely different environment for development and deployment of non-native programs. Attached to the system via the adapter are user programs and the developers of those programs. While those programs and programmers are not as well positioned to take advantage of key platform technologies, they are nonetheless constantly pressured to do so by the architecture group. In effect, the architecture group imposes the partnered leverage problem solving technique that was appropriate for its existing interfaces across a new channel that has emerged within the system.

Similarly, information that may have had no need to have been communicated across the old interface becomes filtered even though it may be pertinent to the user behind the adapter. The case detailed the fact that the chief architects originally spent a lot of time attempting to achieve support from the IBM subsystems for the sysplex strategy. Once this buy-in was achieved and a working relationship established there was no need to re-visit this issue. In effect, the value was understood and filters were created to avoid duplication of this information. Those filters continue to exist today. Rarely, does a member of the architecture team attempt to sell the value of sysplex to new applications developed in-house. The team assumes that the value is understood. As a result, very few new projects are motivated to explore use of these facilities.

Perhaps more costly than the two groups rejecting each other's ideas initially is when they, in fact, do attempt to partner in the creation of applications using the new environment. Current projects provide evidence of designs (produced by designers with a background in traditional system development) which have been over-engineered. In these cases, the design has been formulated assuming the existence of not only primitives providing the same qualities of service as those within the traditional environment but also assuming a skill set and background within the

development team similar to that of the existing environment. In these cases, it has been late in the development cycle before this mismatch was realized.

In response to these realizations, the designers have typically re-engaged with the development team and proceeded to revert further to techniques from the existing environment in an attempt to "save the project". The result of this is a hybrid product that combines techniques from both arenas in a sub-optimal manner. This reactive architecting often surfaces all the way back to the user through changes in the interface and serves to propose a "partner reject" innovation pattern. Effectively, this lack of understanding of the subtleties of the new environment by the designer leads to a product that is ultimately not "manufacturable" to the desired specifications by the development team.

One might ask why the designers were not able to realize these differences beforehand. In particular, why would they not understand the capabilities of the tool set (i.e. the system primitives and the development team) as part of formulating the design ? The fact of the matter is that these things were most likely filtered and assumed away as part of the established problem solving methodology.

As stated, designers interact frequently with the development team in performing decomposition within the detail design phase. During this phase, filters are very pervasive. For example, rarely do the designer and development team discuss issues such as module documentation and the need for module level recovery. These things are assumed to be part of the development criteria. However, when an architectural innovation has occurred this assumption goes unrecognized by the development team. The absence of this function has been shown to go undetected until the latter stages of the project (typically until the stress test phase when the need for advanced development methodologies such as module recovery finally surface).

Just as the filters used by the designers of the new interface are active and damaging when interacting with developers from the new environment so to are the filters that have been established by the adaptive developers through their years of experience within their domain. In short, they assume the platform provider to possess the same competence level regarding the

platform as the native provider - which is undoubtedly not the case. These competencies are developed over time and through experiences in supporting users.

In retrospect, it can be asserted that the addition of adapters does not severely damage the component or architectural knowledge of the firm. That is, the adaptive componentry embodies similar core concepts and proposes similar abstractions as that of the base system. Additionally, these interact in a like manner. It is this notion that allowed the OpenEdition team to efficiently map these to existing base system interfaces in the first place. In gaining this efficiency, however, the firm was able to bypass the crucial learning phase which allows for the building of competency not only in initial creation of the technology but in understanding the needs of its users. In short, adapters allow for a fusing of not only technology but the users of that technology including their cultures, processes, and skill sets. In order to account for these occurrences requires platform providers to make special concessions.

Overcoming Inhibitors -- A Model for Platform Evolution

The previous sections have shown that System/390 finds itself in the unenviable position of supporting a technology for which it has minimal acquired domain knowledge. It is further hampered by the fact that the organizational structure and development process do not currently accommodate the necessary mechanisms for overcoming this deficiency. As a result, the platform is accumulating a portfolio of non-distinctive products along with a set of key technology assets that continue to evolve in a proprietary manner (i.e. they continue to diverge from the new applications). As a result, the platform is becoming "split". While this splitting does not defer from the immediate profits of System/390 (actually, demand continues to increase), it does threaten the long term viability of the platform. In particular, System/390 must reconcile its dual strategy prior to the base becoming irreparably segmented.

This section proposes a model for obtaining the necessary technology fusion. The model is not based in the creation of new technologies or innovative processes that speed time to market, but rather attempts to address the deeper and more subtle problem of how to quickly acquire competencies in the deployment of a new technology and in particular how to overcome the effects of architectural innovation that have been shown to exist by the previous analysis.

Key in this model is the notion of a new position within the organization known as the "Level II" architect. The term "Level II" is formed from the fact that these architects concentrate on the concerns of users who are attempting to deploy applications upon a Level-1 adapter environment such as Open Edition. In effect, these individuals serve as application architects within the adaptive environment as opposed to working "behind" the interface as a platform provider.

Level II architects are members of the system architecture team and report to the same line of management as the "normal" design team. These individuals have a strong background and knowledge of the base system componentry as well as the development process for creating it. This knowledge will have typically been gained through previous work within the base system development area. In short, Level II architects have a background similar to that of existing members of the architecture team. In fact, it would be preferred that they be chosen from this team.

Level II architects accumulate foreign technology knowledge through working directly with the development sub-teams of PDTs throughout the porting process. This differs significantly from the existing process where the design organization maintains an ancillary role as a consultant to the team. The initial rational for the consulting role was grounded in the notion that design resource was too valuable to "waste" porting code. However, this philosophy has resulted in the architecture team, like its base technology path, being removed from the flow of incoming technology. The author submits that when the domain knowledge of the entire architecture team is at stake there should be no higher priority. Integrating early with the PDT provides two advantages to the existing consultant model:

1. Architects are exposed to the subtleties and nuances of the new development environment.

   Engaging early with the development team provides a vehicle where designers can "learn by doing" to enhance their domain skills. Because they understand the filtering methods of the primary architecture team they can subsequently relay this information in an unambiguous fashion to the main group. Hence, accelerating the dissemination of domain knowledge.

2. Relationships with PDTs are enhanced

As indicated, the relationship among the design organization and developers outside the base system area has been marked by inefficiencies. This can be attributed to a lack of shared values. In particular, new environment developers have little appreciation for the core technologies provided within the system. As a result, the design organization has not achieved a stature within this group equivalent to that within the traditional segment. Formal partnering gives members of the architecture team a channel for establishing credibility within the new segment which could lead for enhanced integration in the future.

While the Level II architect works within the development team, his primary responsibility continues to lie in the evaluation of foreign technologies and processes with respect to existing platform componentry and processes*. In effect, the Level II architect serves as a "technology translator" for the remainder of the architecture team. Because they serve as translators to the main design organization, it is imperative that these individuals possess the qualities described above. In effect, they should be positioned to leverage the efficiencies of the existing channels in communicating information regarding new environments.

The concept of a Level II architect is best illustrated through application to the System/390 platform. It has been shown that the design organization has developed strong problem solving patterns over time and as a result have developed channels among base, key, and derivative partners. These channels over time have become optimized through the creation of filters. As stated, flow of information between this group and the groups producing new technologies has typically been inhibited by existing channel filters. These inhibitors however can be overcame not by re-positioning the architecture but rather by re-positioning individuals sharing a common protocol across an inhibited interface. In effect, placement of the Level II architect provides a bypass of the existing filters. This concept is depicted in Figure 26.

* This is also the rational for insuring that the Level II architect continues to report departmentally to the design organization. In particular, it is important that the L2 Architect remain within a reporting structure that is able to clearly identify his objectives and prevent him from being over-consumes with activities not in line with his primary mission.

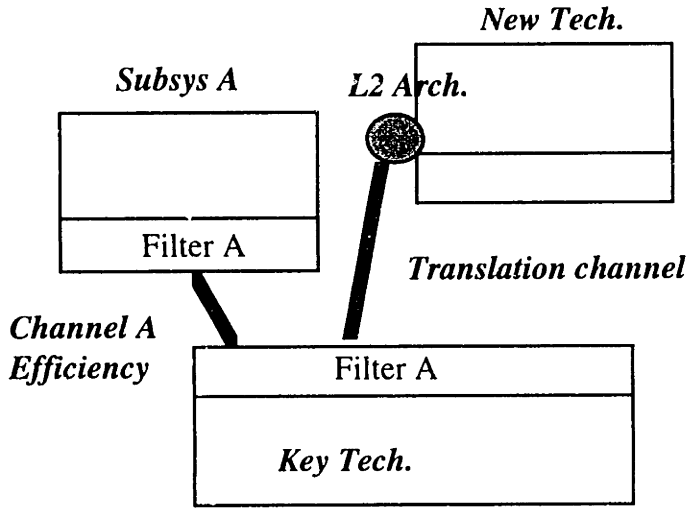


Figure 26. The reconfiguration of persons with channel decoding ability can bypass filters.

It is important that the Level II Architect remain in close touch with the Core design group to insure that his skills do not atrophy and that his translation ability does not become outdated. It is similarly important for the core group to exercise the channel in an outbound manner. To facilitate this communication, a set of deliverables is proposed for the level II architect. These deliverables insure that the organization knowledge base is able to be populated with information that is commonly understood. In effect, the Level II architect is able to construct a technology catalog from his work within the derivative areas. The deliverables required of a Level II architect follow:

- A clear documentation of the foreign product architecture including subcomponent structure and affinities such as data files.
- A porting efficiency report including modifications required to source during the initial adaptation phase.
- Preliminary designs and sizing for base system integration opportunities.
- Assessment of the development environment inhibitors including a prioritized list of plan candidates for addressing these.

- Advanced technology prototypes

Just as PDTs are required to periodically report status to the review board regarding product progress, Level II architects are similarly required to report on the status of the described deliverables. Completing a checkpoint requires approval from both the review board and the software design council.

CHAPTER 6. CONCLUSIONS AND AREAS FOR FURTHER RESEARCH

This thesis has demonstrated through example and reference to existing literature that software architectures are layered, flexible, and dynamic. It has shown that while these characteristics provide significant benefits in the construction of systems that they also pose challenges in managing and communicating the boundaries of not only the components within the system but of the system itself. Highlighted is the fact that management of these boundaries is a shared activity among stakeholders existing in multiple firms and that changes in the system architecture often result in secondary effects within the value chain.

Also illustrated is the fact that the IT industry has undergone unprecedented technical change that has not only affected the business models, technologies, and products of the competitors who compete in this arena but of society in general. It has been shown that from within the industry emerge pervasive products that create virtuous circles of growth with increasing returns for the stakeholders. Increasingly these products are built on open architectures which allow the component parts to be sourced by multiple vendors.

It was also shown that from within this partnered development environment has emerged a new form of architectural based competition known as a "web" with two potential competitive postures for firms to assume -- that of a shaper or that of an adapter. The role a firm assumes within this environment is subject to change over time as a result of the firm's actions.

A model was formulated to describe the choices available to platform providers within web based competitions. The model proposed that platform changes within these environments should be evaluated along the dimensions of interface characteristics and the degree of technology advantage provided by a particular interface. The model yielded a description of four patterns of platform innovation. The "Platform Leverage" pattern was shown to be most beneficial to both the consumer and supplier by mitigating risk through the provision of key differentiating technologies behind standardized interfaces. Platform Leverage also provides an additional benefit to providers in that it advocates fusion of existing platform technologies with new technologies thus helping to standardize the infrastructure.

A case study was performed of IBM's System/390 (aka Mainframe) Software business. System/390 was shown to be a robust, scaleable, and mature hardware and software platform which supports the mission critical business processes of large corporations and in doing so contributes significantly to not only IBM's revenue but also its profits. The platform was shown to have been successfully evolved throughout multiple information technology paradigms. In addition to external paradigms, the case was able to highlight that the impetus for architectural change and the associated decision making process has also transitioned over time from being primarily hardware controlled and driven to being focused on the provision of partnered software solutions.

Logical system boundaries were shown to have been extended over time with each extension being accompanied by an organizational or process change facilitating the need for extended decomposition and integration. Empirical evidence also suggests that an organizational model predicated on the product architecture allows for maximum preservation of component knowledge across changing paradigms. In particular, attempts to modify the development structure from being component-oriented to be project-based inevitably failed.

The analysis shows that while System/390 remains a shaper in the decreasing mainframe segment of the IT industry, its technology portfolio leaves it well positioned to assume a similar role within the emerging network computing arena. Currently, the platform deploys a dual strategy consisting of support for both traditional applications and "new style" applications that originate in Unix environments. Support for Unix environments has occurred as the result of being invaded by a disruptive technology during the late 1980s. Support for these applications is provided in the form

of system componentry that emulates the interfaces and characteristics of a Unix System. The analysis defines the term "adapter" for these entities.

The platform innovation model suggests that an optimum solution would have System/390 converge these disparate system areas into a solution that provides the best qualities of each solution (i.e. deploy a platform leverage strategy). Further application of the model surfaced the fact that not only was System/390 not converging on a common and leveraged technology base but that these technologies were on divergent paths.

It has been concluded that the primary inhibitor of achieving platform leverage in this environment is not based in technology but rather in organizational and process structure. In particular, it was shown that architectural innovation has occurred within the product platform. As a result, efficiency mechanisms that have evolved over time within communication channels between development groups has been shown to have a detrimental effect on information flow when the mechanisms are used across a different communication channel as a result of a subtle reconfiguration of the components within the architecture.

Filtering was shown to be most pervasive across the boundaries of an adapter. In particular, adapters were shown to experience a dual filtering effect. That is, the interface experienced consumer side filtering of information due to patterns that evolved through interactions with other platform providers. Similarly, supplier side filters were shown to exist due to the similarity (but not equality) of function provided.

The solution proposed for this problem included a new position within the organization of the platform provider known as the "Level II" architect. "Level II" architects work hands-on with development teams in the deployment of new applications within the adaptive environment. The primary objective of these individuals is to serve as a "technology" translator to the base architectural teams. Level II architects are chosen from individuals who share common characteristics with the architecture team. As a result, they not only avoid the effect of damaging filters within communication channels, but are actually able to accelerate the fusion process through transporting new information across already optimized channels.

This analysis has raised many issues that may serve as a basis for subsequent research. In particular is was highlighted that firms very rarely maintain the system level architecture in a formal manner. As partnered projects among various vendors becomes more prevalent, it will be interesting to explore the manner in which system level architectures are able to be communicated. As a base, one could choose to study the effectiveness of the various software framework components that are starting to appear on the worldwide web.

Although highlighted only tangentially in this study, it is becoming obvious that allocation of resources within firms is becoming an increasingly difficult task due to the number of new technologies in which a firm must try to support. A study of the "change-over" costs among individuals in a software development group would provide insight into the firm's ability to shift resources as part of managing the development pipeline for the creation of new technologies.

Similarly, it is becoming necessary that firms not only extend the bounds of technology systems but also look for mechanisms to make maintaining of the existing infrastructure more cost effective. In software systems, this most closely relates to the process of re-writing code for maintainability. It would be interesting to formulate a model to understand where this is an effective technique.

And lastly, this analysis proposes a structure which called for the bypassing of component filters through the "seeding" of new technology groups with members from the core architecture group. While the model proposed preventing atrophy in channel skills through frequent utilization, it would be interesting to understand the rate at which encoding/decoding skills do in fact deteriorate within product development firms and what are the factors which influence this.

BIBLIOGRAPHY

[1] Nevens, T., Summe, G., and Uttal B. (1990), "Commercializing Technology: What the Best Companies Do". *Harvard Business Review* (May-June 1990).

[2] Uzumeri, M., and Sanderson, S. (1995), "A framework for model and product family competition." *Research Policy,* 24,583-607.

[3] Wheelwright, S.C., and Clark, K.B. (1992), *Revolutionizing Product Development,* Mc-Graw Hill, Inc., New York, New York.

[4] Meyer, M., and Utterback J. (1993), "The Product Family and the Dynamics of Core Capability." *Sloan Management Review* (Spring), 29-47.

[5] Prahalad, C.K., and Hamel, G. (1990), "The Core Competence of the Corporation", *Harvard Business Review* (May-June 1990).

[6] Anderson, P., and Tushman, M.L. (1990), "Technological Discontinuities and Dominant Designs: A Cyclical Model of Technological Change", *Administrative Science Quarterly* (December 1990) 604-633.

[7] Campbell, D. (1969), "Variation and selective retention in socio-cultural evolution", *General Systems* (14) 69-85.

[8] Henderson, R.M., and Clark K.B. (1990), "Architectural Innovation: The Reconfiguration of Existing Product Technologies and the Failure of Established Firms", *Administrative Science Quarterly* (March 1990) 9-30.

[9] Bower, J.L., and Christensen, C.M. (1995). "Disruptive Technologies: Catching The Wave", *Harvard Business Review* (January-Feburary 1995) 43-53.

[10] Gulati, R., and Eppinger, S.D. (1996). "The Coupling of Product Architecture and Organizational Structure Decisions", *MIT International Center for Research on the Management of Technology.* Working Paper #151-96 (May 1996)

[11] Liskov, B., and Guttag, J. (1986), *Abstraction and Specification in Program Development,* The MIT Press, Cambridge, MA.

[12] Rechtin, E. (1991), *Systems Architecting: Creating and Building Complex Systems,* Prentice Hall, Englewood Cliffs, N.J.

[13] Booch, G. (1994), *Object-Oriented Analysis and Design with Applications,* The Benjamin/Cummings Publishing Company Inc., Redwood City, CA.

[14] Shaw, M. (1989), Larger Scale Systems Require Higher-Level Abstractions. *Proceedings of the Fifth International Workshop on Software Specification and Design.* IEEE Computer Society.

[15] Ulrich, K. and Eppinger, S.D. (1995), *Product Design and Development,* McGraw-Hill, Inc. New York, New York.

[16] Cusumano, M.A., and Selby, R.W. (1995), *Microsoft Secrets. How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People,* The Free Press, New York, New York.

[17] Cussamano and Selby, pg.

[18] Guttag, J. (1995) System Design and Management Proseminar Lecture, MIT, Fall 1995, Unpublished.
Guttag also warns that Rapid Application Development can be a trap if not managed properly. In particular, he hilights the possibility of a run away project if clear goals are not set for the output of each iteration.

[19] Cussamano and Selby, pg. 131.

[20] Krivda, C. (1995) Available at a Price; High Availability Systems can be achieved no matter what your budget may be, *MidRange Systems,* n8, v8, April 1995.

Note, the subject study lists the cost of an outage as being approximately $84,000/hour. Independent studies by Find/SVP (1992) and Oracle Corporation (1995) estimated this to be $1400/minute. All studies indicated that these were conservative figures, normalized across industries. It was noted that industries such as Financial services will experience much greater losses due to system failure.

[21] Young, J. (1996) *Exploring IBM's New Age Mainframes,* Maximum Press, Gulf Breeze, FL

[22] Morris, C. and Ferguson, C. (1993), "How Architecture Wins Technology Wars", *Harvard Business Review* (March-April 1993).

[23] This is supported by the research of Dr. Madassor Manoor, a leading consultant in Business Intelligence.

As an interesting note, Microsoft corporation predicts this number to be much higher. This information is available from the Microsoft worldwide web site, www.microsoft.com. See presentations on Cedar technology.

[24] Bauer, M. et al (1994), "A distributed system architecture for a distributed application environment", *IBM Systems Journal* (Vol. 33, No. 3, 1994).

[25] Cussamano and Selby, pg.

[26]Comment by System/390 Customer Design Council member, April 1997., Poughkeepsie, NY. Name and company are not able to be disclosed in this forum.

[27] Business Week article on McNeely

[28] Orfali, R., Harkey, D., Edwards, J. (1996) *John Wiley & Sons, Inc.,* New York, New York

[29] Morris, C. and Ferguson. C. (1993), "How Architecture Wins Technology Wars", *Harvard Business Review* (March-April 1993).

[30] Hagel III, J. (1996), "Spider versus Spider", *The McKinsey Quarterly* (1996, Number 1).

[31] Achi, Z., Doman, A., Sibony O., Sinha J., Witt S. (1995), "The Paradox of Fast Growth Tigers", The McKinsey Quarterly (1995, Number 3).

[32] Adler P. and Shenhar A. (1990), "Adapting Your Technological Base: The Organizational Challenge", *Sloan Management Review* (Fall 1990).

[33] Carico, Bill, and Van Der Zel, W. (1996), "OpenEdition MVS: The System, the Strategy, the Significance: Part I", *Technical Support* (May 1996)