A Framework for the Assessment of Knowledge Transfer in Software Development Organizations

by

John P. Woods

B.A. Computer Science, State University of New York, College at Oswego, 1980

Submitted to the System Design and Management Program
in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Engineering and Management
at the
Massachusetts Institute of Technology

May 2001

Signature of Author [illegible]
John P. Woods
System Design and Management Fellow

Certified by [illegible]
Paul R. Carlile
Assistant Professor of Management, Sloan School of Management
Thesis Supervisor

Accepted by [illegible]
Stephen C. Graves
Abraham Siegel Professor of Management
LFM/SDM Co-Director

Accepted by [illegible]
Paul A. Lagace
Professor of Aeronautics & Astronautics and Engineering Systems
LFM/SDM Co-Director

Abstract

Knowledge Transfer is a generic term that is applicable to many domains. This thesis will analyze the specific issues required to develop a comprehensive framework for the assessment of knowledge transfer in the software development environment. Knowledge transfer is an essential component of all business processes in software development. The framework in this thesis was developed at a level of sufficient abstraction to be applicable to all software development activities. However, the research interviews focused on two scenarios in software development recognized as requiring effective knowledge transfer in order to achieve successful results:

- **Porting**: A Porting project is an effort to enable a program to run on a different hardware or software platform. To port an application, you need to rewrite sections that are machine dependent, and then recompile the program on the new computer. The resultant code is then tested, often by some type of compliance suite, to insure that the ported product is operationally equivalent to the original product.

  When there is a market or business need, software products may be "ported" to new hardware/software platforms. A new team, with expertise in the new platform architecture, is usually called upon to do the porting. In order to optimize the port, knowledge from the original development team must be effectively transferred to the new organization.

Service Transfer (maintenance) – Similar to the above, an software development organization will often transfer or assign responsibility for product service/maintenance to a separate service/support team. The ability to insure high-quality customer support requires a successful transfer of knowledge from the development group to the service/support group.

This thesis provides a comprehensive framework for assessing a software development organization's knowledge transfer efforts and requirements. The framework presents a balanced approach introducing three columns of support for knowledge management activities in a software development organization. The three columns of support for knowledge transfer are:

- Organizational Structures
- Operational Process and Procedures
- Technical Expertise and Infrastructure

Using the three columns of software support as an analytical tool provides a holistic, end to end view of an organization's support for knowledge transfer.

.

Acknowledgements

I would like to thank my thesis advisor, Paul Carlile, for his support and valuable insights into the complex world of knowledge management in product development settings. Professor Carlile is a gifted teacher. Some of my most valued learning experiences during my studies at MIT have resulted from his perspectives and insights on knowledge management and organizational behavior. He has been an essential contributor to the overall academic enrichment of many SDM fellows. The SDM program as a whole has benefited tremendously from his teaching. I have returned to Paul's office many times for a "drink from the fire hose".

I would also like to thank my classmates in the SDM program for their friendship, perspectives, insights and their ability to laugh in the face adversity. The interaction with all my SDM classmates has been critical in forming the basis of my educational experience at MIT.

I would particularly like to thank Benjamin Koo and Pete Panetta for their friendship, teamwork and pursuit of academic excellence throughout the program. We did some great work together and it was always a pleasure to work with you both. In addition, my thanks to Ray Ro and Tim Flores for their friendship, support and willingness to share an idea or a laugh, regardless of the hour.

I am extremely grateful to the IBM Corporation for sponsoring my participation in the program. The company has provided me a unique opportunity for personal and intellectual growth at this mid-point in my career.

Most importantly, I would like to thank my family for their patience and support throughout this intense and demanding program.

Dedication

I dedicate this thesis to my wife and daughters....

To Shannon and Cailyn

You go girls!!!
May you always follow your dreams...

To Cindy Lynn

If the sky that we look upon
Should tumble and fall,
Or the mountains should
crumble to the sea,

I won't cry, I won't cry,
No, I won't shed a tear,
Just as long as you stand,
Stand by Me... [1]

Table of Contents

List of Figures

Introduction

Problem Statement/Motivation

Knowledge Management (KM) has been one of the most important topics of research for decades in the management literature. Beginning in the mid-1990's Knowledge Management (KM) came to the forefront of software development and became one of the hottest topics in information technology. The business press began heralding Knowledge Management as the next Silver Bullet[2] of the software development industry. However, like all silver bullets that proceed it, the term Knowledge Management is beginning to evoke more skepticism than excitement in the minds of technical and business leaders. Much of the disappointment can be attributed to the fact that the predominant approach to knowledge management has been information technically based. The common view being marketed in the business work is that with the installation of a corporate portal, search engine, data repository and collaboration tools, knowledge will begin flowing in the organization. Larry Prusak, executive director of IBM's Institute for Knowledge Management, states that he has observed hundreds KM implementations, a majority of which are sub-optimal. Prusak states that the primary reason for the problem is that it is easier and faster to just buy the technology than to think through the strategic issues.[3] It is with the problem in mind that this research began with the goal of developing a framework for the analysis of knowledge management in the software development organization.

A comprehensive framework is required in order to understand the essential inhibitors, enablers and components of a successful knowledge transfer in software development. Software development is a highly complex and technically challenging undertaking. This complexity extends well beyond the actual source code of a single application. Most assessments in the field today focus primarily on the technical expertise associated with product development. However, this approach fails to capture the full extent of project complexity.

The technological component itself requires understanding in terms that extend well beyond the subject software. One must also understand the associated operating systems, companion products, and system standards. There is also a strong set of technological functionality that supports the organizational structure and operational process of the two groups, which must be understood as well.

In order to perform a realistic assessment of knowledge management in a software development organization, a framework that allows holistic assessment of organizational, operations and technological perspectives is required. This thesis defines such a framework.

Thesis Goals

The goal of this thesis is to develop a framework that will enable managers and technical leaders in a software development organization to analyze the current state and needs for knowledge management in the organization. The framework defines a pattern inquiry to provide a comprehensive analysis of the knowledge transfer requirements and practices of the organization. The output of the analysis is descriptive in nature. This descriptive analysis can be used for two purposes. First, it allows management to judge the overall readiness of the groups to successfully transfer of knowledge across the organizational boundaries. Second, it provides managers and technical leaders within an organization with a foundation on which to build prescriptive knowledge management plans in order to take appropriate actions to bridge the gaps in knowledge transfer activities.

Thesis Structure

This remainder of this thesis is divided into eight sections:

The ***Overview of Research Approach*** section describes the foundation for the inquiry approach taken during the interviews of technologists, managers, technical leaders and individual contributors for this research effort. The research approach addresses the essential elements of knowledge transfer in software development from the organizational, technical and operational perspectives.

The ***Porting Project Background*** section provides basic background and context for the porting projects used in this research effort. The section which follows, ***Porting Project Research Summary***, provides a description of the interview results and highlights the key insights from each of the projects

Similar to the above two sections, the ***Service/Support Project Background*** section provides basic background and context to the service transfer project research interviews for this thesis. The ***Service/Support Research Summary*** provides a description of the interview results and highlights the key insights from each of the projects

The next chapter, ***Conclusion and Next Steps***, compares and contrasts the insights from the two major thrusts of the thesis and provides an overall summary of the key insights of this research.

This thesis concludes with two additional chapters. The first, ***Beyond the Code,*** provides a boundary for the overall scope of this research and frames a discussion for additional research in the future. The final chapter, ***Software Development Knowledge Transfer Assessment Framework***, lays the groundwork for a new framework, which could be used to address issues regarding knowledge management in software development organizations that extend "Beyond the Code".

Overview of Research Approach

Overview

It is the premise of this thesis that a comprehensive framework is required to address knowledge transfer in a software development organization. Three columns of inquiry; organizational, operational and technological provide the support of the framework. Each column offers a different perspective on knowledge transfer in software development. However, like the strands of a rope, the three columns are tightly intertwined in the development organization. This results in some overlap in the specific elements of the columns. This overlap is positive and warranted as it allows the analyst to view the essential elements of knowledge transfer activities from a number of valuable perspectives. Typically, management and technical leaders have a bias to one of these perspectives. The use of this framework allows for a more holistic approach to the analysis.

The purpose of this section is to outline the research approach used to structure interviews with management, technical leaders and individual contributors across several software development and service teams. It is best to view this section as a starting point of the analysis used to gain insights into the scope and essential characteristics of effective knowledge transfer in software development. The interviews attempt to uncover the organizational structure and operational environment of the software development/support groups involved in the transfer. It also assesses the requirements for technical expertise and the technological infrastructure in place to support the transfer. The focus of the research interviews is not on the solely the structure of each of the organizations. The research also attempts to discover the essential flows knowledge to and from the team, and to identify critical gaps in the transfer of knowledge.

As stated above, this is a starting point of inquiry for this thesis. The author recognizes that the research approach outlined below is expansive and complex.

However, utilizing this approach as a starting point for interviews with executives, technical leaders and individual contributors, the goal is to develop a much more concise, yet comprehensive framework for the assessment of knowledge management in the software development organization.

Organizational

The first column of support in the analysis framework is organizational. This research uses the three-lens approach as a starting point to organizational analysis in order to maximize the overall understanding of the organizational structures involved. A brief overview of the three-lens approach is provided here. However, as we move into the analysis and conclusion sections of the thesis, the organizational framework used for analysis moves towards a more domain specific taxonomy that is more useful in the context of software development.

Strategic Lens:
The strategic lens focuses on several essential organizational factors in the software development team and the teams with which they interact:

- Strategic Intent embodies the overall strategy and goal of the organization. For software organizations involved in knowledge transfer, this question addresses topics such as:
  - What is the purpose of the organization?
    - Is this a development group, service group or both?
    - Is this a revenue generation group or cost center?
  - How does the organization's goals/focus balance the development of new and leading edge technology with insuring enterprise level of software reliability?

- Strategic Grouping – How activities are distributed into jobs, roles, departments, etc... At the department level, this includes software development, quality assurance, software service and support, etc.... As the focus moves downward, to an individual team, you begin to see grouping in terms of generalists vs. specialists, operating system affinity, major software component focus, etc... How a team cleaves responsibility among individual members provides interesting insights into the success of knowledge transfer. Some typical groupings in software development teams include:

- Grouping by product function (client, server, etc...)
- Grouping by component function – Database, network, languages, etc..
- Business Process – development, support, test

In most development organizations, there is a combination or hierarchy of various groupings. When analyzing the strategic grouping of a software development team it is useful to understand to understand the grouping in terms of size, level of specialization and degree of interaction between the teams.

- Strategic Linking addresses how groups interface across the boundaries within their own organizational structure and/or other partner organizations. During this analysis, the strategic linking can be viewed at the micro level for linking within the organization. More important is the strategic linking at the macro level between organizations or team involved in the knowledge transfer. Analysis of both the existence and robustness of the interfaces is required. Common examples of strategic linking mechanisms include:

  - Formal hierarchy of reporting structure and information flow. What is the hierarchical structure of the owning and receiving organizations? At what level do the groups meet in the hierarchy?
  - Liaison Roles - Are there formal boundary managers within each organization? Across the organizations? How effective are these boundary mangers in facilitating the transfer of knowledge?

- Cross-Unit groups – To what extent are cross-unit groups used between the organizations. What is the purpose, time frame and level of commitment for these groups? (e.g. Problem focused v. function focused, permanent v. temporary, full-time v. part-time)
- Information Technology Systems – To what extend are Information Technology systems used to support knowledge transfer efforts between the organizations? Are they effective? Are the systems customizable?
- Planning process – What are the mechanics of the planning process between the organization. Is there cross boundary representation and/or participation? Do all organizational entities have input into the process? In terms of involvement in the planning process, many times membership is more important than overall effect on the process output.

- Strategic Alignment – What are the alignment elements within the groups (e.g. incentives). Does alignment exist across the organizations? Some typical examples of strategic alignment elements include:
    - Rewards and Incentives
    - Team and Organizational Metrics
    - Resource allocation/staffing
    - Resource development and training
    - Political and Cultural Influences

James Thompson's typology of task interdependence[4] is a critical component of strategic analysis in the software development field. Task interdependence can take the form of:

- Sequential Interdependence – In software development the movement between major milestone builds or the relationship between one release and the next typify sequential task interdependence. However, sequential interdependence is applicable when viewing task interdependence from the highest level of abstraction. Certainly the code of a of a prior release of software forms the foundation for the current release, however, the challenges of knowledge transfer in software development rarely follows such a linear transformation.

- Pooled interdependence – In software development different groups and individuals typically contribute to the final product. This is the general model of the product build cycle. Each component team contributes code changes that are combined by the release engineering team during the product build cycle.

- Reciprocal interdependence – This is the most complex form of task interdependence and quite common in software development organizations. Reciprocal interdependence is characterized by complex, dense interactions between tasks across team and organizational boundaries. In software, development it is quite common that a code change made by one developer has dependencies or effects on code changes made by other developers. Reciprocal interdependence is one of the primary drivers of complexity in software development today. In fact, much of the research in software design and modularity is based on ways to reduce reciprocal interdependence by creating clear defined interfaces between components.

This typology of task interdependence is critical in the analysis of the teams and organizations involved in the software development knowledge transfer. In many ways, interdependence is the foundation for the need of knowledge transfer activities between organizations.

Political Lens:

The core of the political lens on an organization begins with a "stakeholder" analysis. A sample stakeholder map of a software development organization is shown in Figure 1: Typical Stakeholder Map of a Software Development

Organization. Depending on the specific team under analysis (e.g. development, porting team, service team, etc...) the orientation and central focus of the map will change.

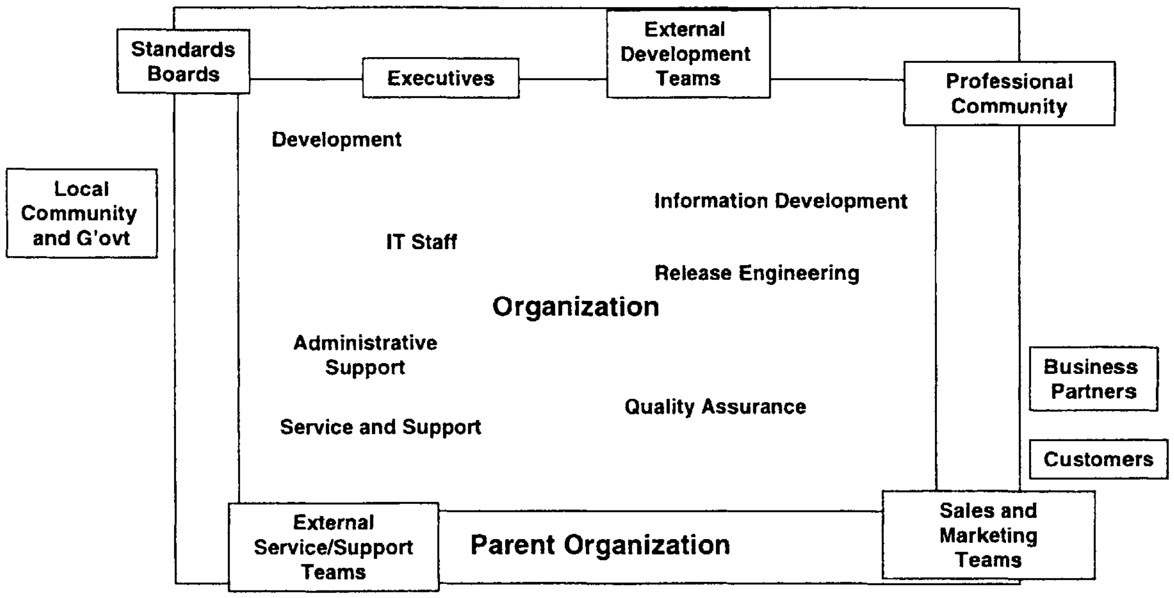


Figure 1: Typical Stakeholder Map of a Software Development Organization

The following groups categorize Stakeholders:

- Groups and/or individuals that contribute to the creation and development of the organization's output. (e.g. product or support) In a typical software development organization, this would primarily include the core development team, the service/support team, quality assurance, release engineering, information development, IT Staff and any other groups would make a direct contribution to the product.

- Groups and/or individuals that provide input to the function or organization's development process. The groups that provide input to the product development process can be viewed as the broad set of upstream influences on product development as illustrated in Figure 2: Upstream Influences on Software Product Development.

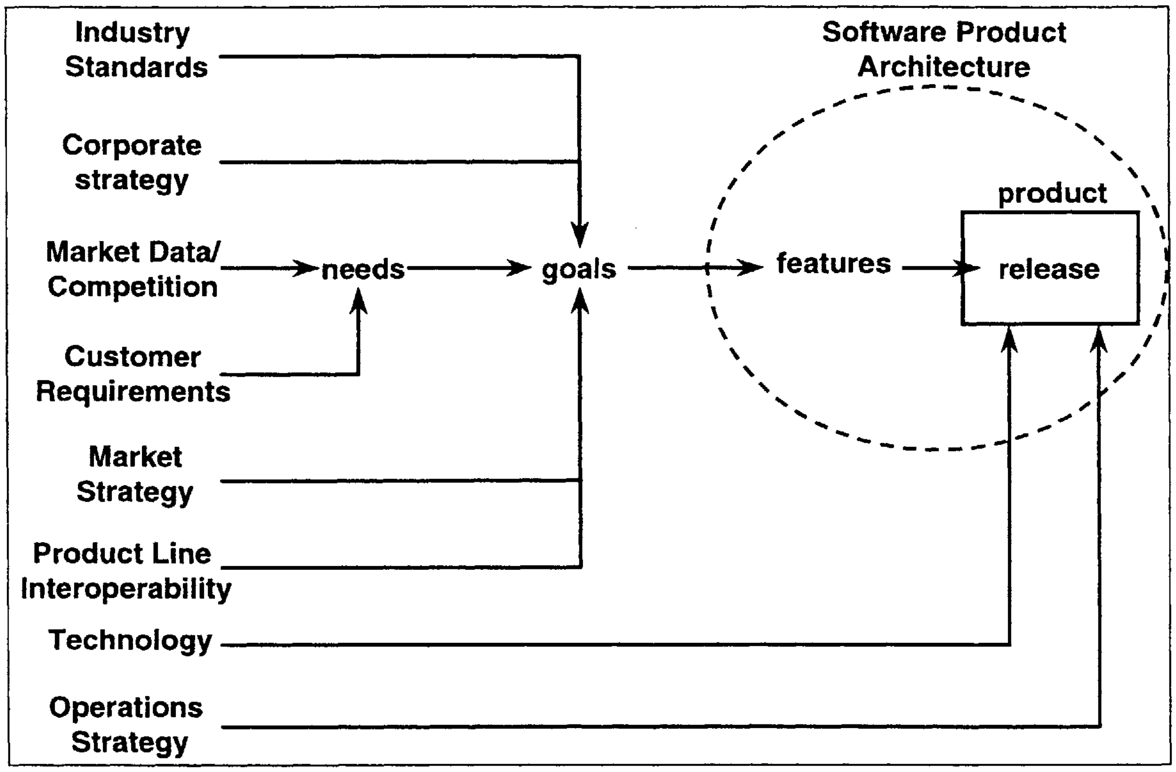


Figure 2: Upstream Influences on Software Product Development

- Groups and/or individuals who have dependencies on the output of the organization. In software development, this may include the customers, the sales and marketing teams, service and support, manufacturing, distribution, competitors as illustrated in Figure 3: Downstream Influences of Software Product Development.

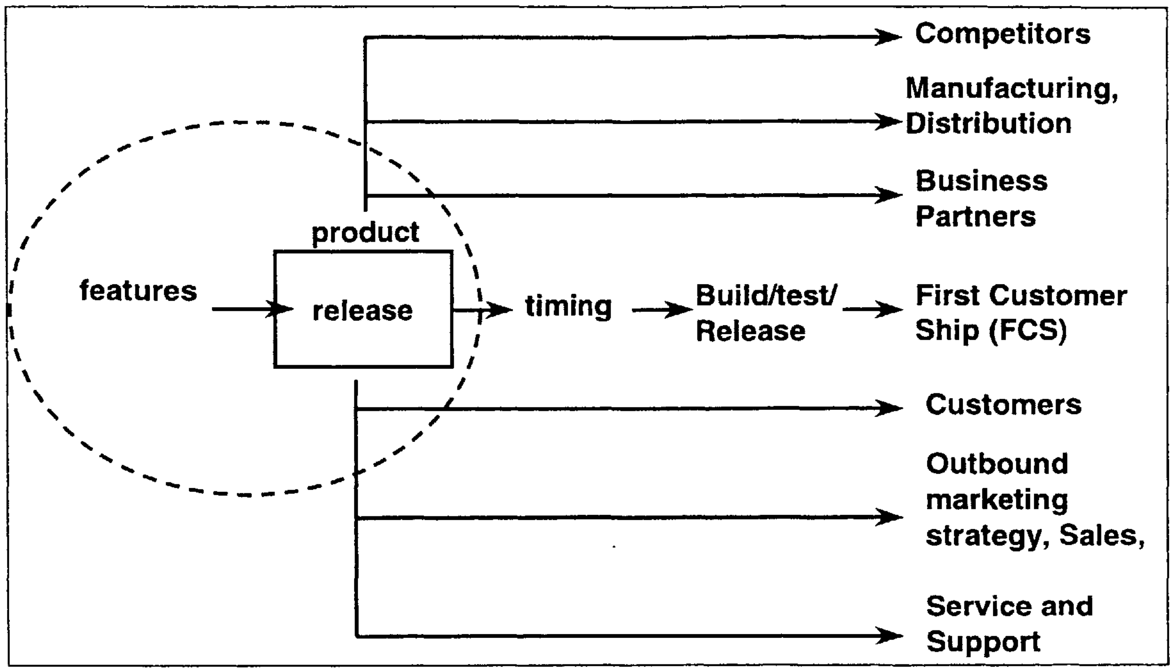


Figure 3: Downstream Influences of Software Product Development

The key analytical building blocks of a political perspective are quite simple. To use a political perspective you must:

1. Identify and map the relationships among the different *stakeholders* involved.
2. Uncover the most salient *interests and goals* the different stakeholders bring to the interaction and the extent to which the conflict or are congruent.
3. Assess the amount and sources of ***power*** of the different stakeholders[5].

The key to the political lens begins with the recognition and acknowledgement differences between organizations. The differences may be expressed in terms of motivation, interests and goals. It is critical that these differences be acknowledged in a non-judgmental fashion as valid and legitimate perspectives within the organization.

It is equally important to understand individuals and groups can have multiple interests and differences. The nature of the interests and differences is also quite dynamic, changing over time and circumstances. [6]

The categorization of organizational influences as upstream and downstream is a useful abstraction during organizational analysis. However, it is critical to recognize the temporal nature of upstream and downstream influences as they related to the system as a whole. The grouping of upstream and downstream influences typically represent the snapshot of the state of an organization at a specific instance in time. We must be aware that many of the downstream influences can and do provide feedback within the system to the upstream influences in another period of time. As Peter Senge points out, "Out there" and "in here" are usually part of a single system.[7] When doing a boundary analysis of knowledge transfer, it is important to recognize the system as a holistic entity. The downstream influences on the process today typically have a direct impact on upstream influences as the system moves through time.

The political lens is particularly useful in the analysis of the knowledge transfer in software development. In software development, knowledge such as software engineering, operating system, tools, domain specific, etc... has traditionally been a leading source of power. It is also critical to understanding the power and influences from up and down stream. The use of the political lens aids the ability to identify the sources of power and a starting point on how they influence effective knowledge transfer.

Cultural Lens:

In its broadest anthropological sense, culture refers to the way of life shared by members of a given society and includes knowledge, belief, art, morals, law, customs, an any other abilities and habits acquired by members of that society. [8] However, from an organizational perspective, Schein summed up culture with the statement that "Culture is the residue of success"[9]. As shown in Figure 4: Level of Culture, Schein states there are three levels of culture within organizations. An in depth cultural assessment of an organization is a complex and time-consuming endeavor that is well beyond the scope of this framework. However, a basic awareness of the cultural perspective of the organizational model has proved to be useful tool during the interview process.

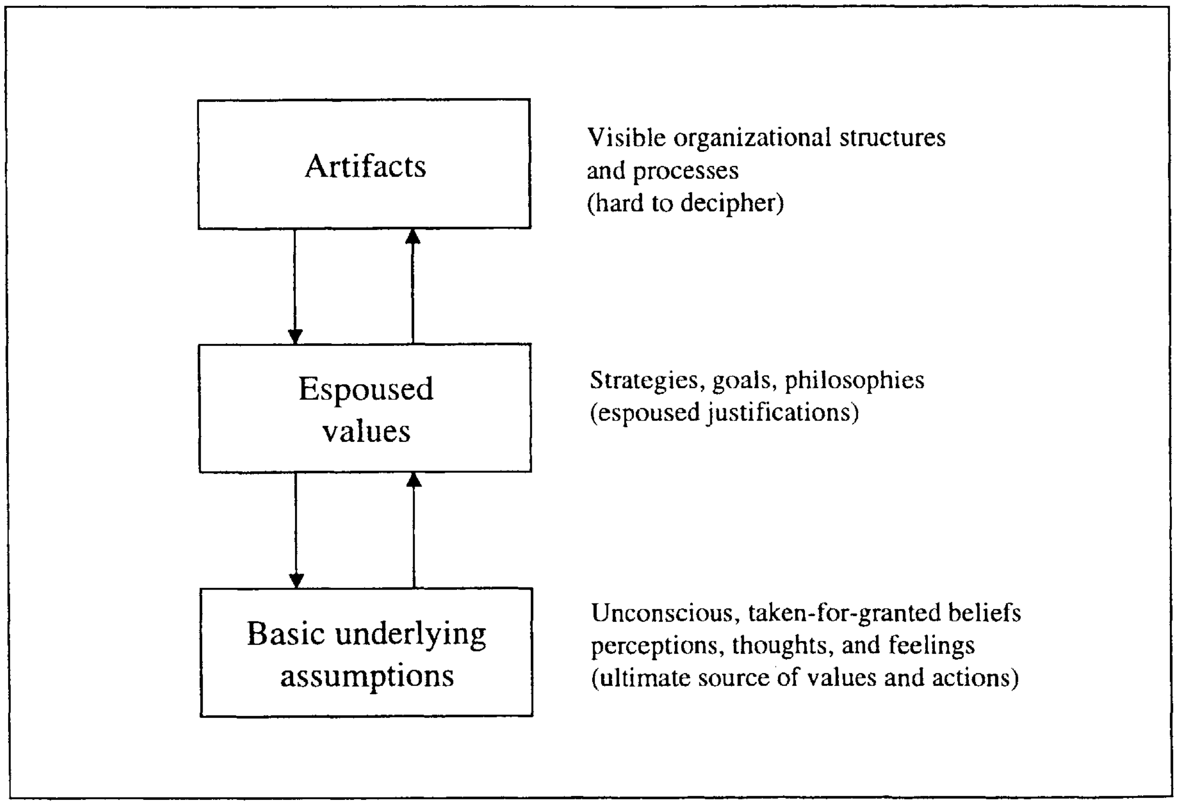

Artifacts
Visible organizational structures and processes (hard to decipher)
Espoused values
Strategies, goals, philosophies (espoused justifications)
Basic underlying assumptions
Unconscious, taken-for-granted beliefs perceptions, thoughts, and feelings (ultimate source of values and actions)

Figure 4: Level of Culture[10]

The highest level of organization culture is the ***Artifacts***. This is the easiest level to observe when doing a cultural analysis of an organization. The artifact level is comprised of the environmental factors that you encounter when you enter the organizational setting. These are the things you see, hear and feel as you interact within the work area. This includes items such as.

- Type of business attire
- Closed offices vs. Open landscape
- Pace of activity and employee behaviors

This level provides an immediate emotional response and/or reaction. However, there is little understanding of organizational culture at this level.

Level two, ***Espoused Values***, begins to get below the surface of the organizational culture. Analysis at this level usually requires inquiry with veterans of the organization as informants. The espoused values may include items such as:

- Corporate Values (Quality, Teamwork, Customer Service, Respect for the individual, Integrity, etc...)
- Corporate Principles
- Goals and Vision Statements.

Level three, ***Basic Underlying Assumptions***, is where the real grist of corporate culture has its roots. It is at this level that you begin to uncover the shared tacit assumptions that underpin the corporate culture. Many times these assumptions have their roots in the minds, actions and beliefs of the corporate founders. The founders are likely to impose their values and beliefs on those that they initially hire. Over time, these values and beliefs are ingrained as tacit knowledge that is taken for granted by employees of the firm. [11]

Operational

The second column of support in the framework is operational. One of the essential elements of organizational analysis is the operational environment in place to support the software development activities of the organization. What are the operational tools in place to support the development activities? This includes change management tools, problem management tools, knowledge capture and dissemination tools. What operational process and procedures are in place to address the boundaries of knowledge transfer? In addition, the tacit operational characteristics and interactions of an organization that are not explicitly captured in a particular process must be recognized and understood.

The purpose of the operational analysis is not merely to ascertain the existence of the operational tools and processes. More important to understanding the flow of knowledge transfer is to gain understanding of the differences or gaps between the operational environment of the two organizations and what processes are in place to work across that boundary.

Goals

Understanding the business and operational goals of group is essential in any organizational assessment. However, in this framework, the goals of the organization must also be viewed in terms of the requirements they drive on knowledge transfer activities. The assessment also interprets how are the development goals of an organization supported via the operational infrastructure?

Process

How well is the formal processes (documentation, change management, problem reports, requirements, specifications, etc) defined and understood by the partner organizations? What processes are in place to manage the interface between the two organizations? The operational characteristics of the informal processes must also be examined. Some examples of analysis in this area include:

- ♦ Are specific boundary managers defined to work between organizations?
  - ♦ What is the basis and support for these individuals?
- ♦ How do the processes of the organizations relate?
  - ♦ Are they shared across boundaries?
  - ♦ Are the processes duplicated and run in parallel? If so, is the interface well defined and robust?
  - ♦ Are they disparate processes? If so, is the bridging mechanism defined and robust?

Metrics

Metrics are important tools in the software development arena. However, as James Bach points out "In order to use metrics wisely ... you need the added ingredients of humility and *inquiry*. If you have an inquiring attitude, then metrics join all your other observations to help make sense of your situation." [12]

What metrics are used by the organization to measure success? How do these metrics support the development organizational goals? Are metrics in place to support the knowledge transfer activities between organizations? Are the metrics and the intent of the metrics understood? Are they closed loop systems or open loop systems?

Rewards and Incentives

Discussion of rewards and incentives are typically considered an organizational issue. However, I have also grouped them under operations in this framework. The focus here is on whether behavior that supports the operational characteristics of knowledge transfer is valued and rewarded by the organization.

Technological

The third and most fundamental column of support in the framework is technological perspective. Technology is at the heart of the software product development and software development knowledge transfer activity. After all, it is the technological expertise that must be successfully transferred in the subject development and support efforts. However, the technological characteristics of an organization also provide a broad layer of information that spans and supports the organizational structure and the operational goals the development team. Technology must be viewed from the following perspectives in order to adequately assess an organization's knowledge management activities in the software development environment.

Core Knowledge/Expertise

The technological thrust begins at the individual/team level of core expertise in the product domain. These are the core skills that allow an individual to work productively in the development environment. These skills include:

- Operating System
    - As an execution environment
    - Compiler and Linker
    - Debugger
    - Make Facility
    - Editors
- Programming Languages
- Standards and Protocols

Domain Knowledge/Expertise

The next level of technical knowledge is the application specific, domain layer. Given the prerequisite core knowledge as a foundation, this layer addresses the knowledge surrounding the specific application under development or support. These skills include:

- Code Access and Understanding
    - Understanding of the component definitions and interfaces
    - Module/Data Flows
    - Coding Standards
- Development Intent – Specifications, history, change control
- Debugging Aids
- Development Hints and Tips
- Ancillary Products

Process Support Tools and Control Mechanisms

The next layer of technology concerns the process support environment for development activities. Technical analysis at this level addresses process support tools and procedures including:

- Development change management process/tools
- problem reporting process/tools (internal, field, escalation, ...)
- build tools and procedures
- required hardware/software/machine time

It is also helpful to ascertain the level of understanding of the written and non-written development processes. During analysis, special attention should be given to the technological gaps/differences between the two parties of the knowledge flow. Are the processes and tools shared, duplicated, bridged or disjoint?

Collaboration

From the technological perspective, the focus on collaboration is aimed at the technological infrastructure to support collaborative efforts. This includes:

- Formal Databases, tools, web sites
- Communications vehicles
- Peer-To-Peer technical relationships
  - Across the organizational boundaries
  - To operating system, companion product expertise

Complexity

Complexity is at the core of all significant product development efforts. Joel Moses[13], stated that "Complexity is neither 'good' nor 'bad', but it is something that one 'spends' to achieve the end goal." With that context in mind, this framework includes an analysis to understand the important drivers of technical complexity in areas such as:

- Product Architecture
- Requisite Operating system(s)
- Development Environment
- Process
- Tools

- Flexible interfaces within and between the partner organization
- Linking vs. Duplication

It is a cross-domain architectural axiom that systems typically fail at the interfaces. Therefore, the analysis of complexity is particularly applicable to the knowledge transfer boundaries between the teams and organizations. The study of complexity naturally contains a strong bias towards definition and robustness of interfaces. Analysis should also distinguish between essential and gratuitous complexity. Essential complexity is driven by the core requirements of robust functionality.[14] Gratuitous complexity typically arises as a result of poor architectural constructs and/or poorly defined interfaces. The analyst should be constantly aware of the sources of complexity within the environment.

Porting Project Background

Overview:

In the most generic terms, porting is an operation to convert software to run in a different computer environment. The new computer environment can be a new hardware platform and/or a new operating system. The phrase "to port the program to UNIX" means to make the necessary changes in the application to enable it to run under a specific UNIX operating system.

In general, I have encountered two types of ports during my research. With the first type a development group obtains source code for an application in order to be "ported" to a new operating system or hardware platform. After the initial port, the new product than takes on a life cycle of its own, independent of the original version. There are business circumstances when this is an appropriate approach as a means of jump starting a development effort. However, these efforts are not successful "ports" of a product because the bridge to the original team (and code base) is broken.

With the second type of "port" the development group obtains source code for an application in order to be "ported" to a new operating system or hardware platform. When the changes are made, they are then merged back into the original source code base of the owning development team. This second type of port, when done well, allows for single source stream to be "portable" across several operating systems and hardware platforms. The product can then maintain consistency in schedules, feature sets, service, support and general development. This second, narrower definition of porting, is what this thesis defines as a successful porting strategy. This second approach is obviously more interesting from a knowledge transfer and operational logistics perspective as well.

Porting Project Research Summary

Overview

Interviews for this research where conducted with several different porting teams over a range of products as outlined below:

- Large Scale Software Server Project
  I conducted several interviews with a porting team responsible for the port of a large scale, extremely sophisticated enterprise server product. The product architecture was developed with portability as a primary design goal. The teams were responsible for porting the product to a new variation of UNIX. The size of the project was extremely large with the source base exceeding a million lines of code.

- Mid-Size Software Server Project
  The Mid-Sized software server project was an effort to port and external company's product to a UNIX environment to enable printer and file sharing between platforms. The code base was approximately 1000 source parts and was developed for a non-UNIX operating system without clear architectural attention to portability.

- PC Based Project
  The third project analyzed was a PC based project which ported a mid-sized, single-user, business application program for the Windows operating system being ported to the OS/2 platform.

Large Scale Software Server Project

Organizational Data

This is a long-term project, which has been in operation for approximately five years. In that period, the team has ported four major feature releases and over two dozen maintenance releases. As expected, there have been numerous transitions within the project over time moving the product from to a "yes it runs" version in the initial release to achieving dominance in industry standard benchmarks in later releases of the product. In fact, the porting team's performance maps very well to Tuckman's[15] model of small-group development following the path through Forming, Storming, Norming and Performing.

The initial porting team, responsible for the proof of concept port, was a small, highly motivated team of 8-10 developers given the task of porting the large scale (>1 million lines of code) server software product. The team was located in a different state than the core development team. However, the team was co-located with the target platform operating system development team.

There was strong management support for the initial team, both in terms of developing an independent, skunk-works mentally and in terms of providing strong financial support for capital equipment purchases. Initially, there was very little interaction with the core development team other than periodic snapshots of new code and 2-3 project reviews with the core team per year.

As the project matured and moved into a steady-state mode, the size of the porting team grew to approximately 25 developers and 25 additional people in quality assurance engineers and technical support roles. This drove a greater need for interaction with the core development team. Consequently, a small subset of the team was co-located with the core development team.

After the sub-team was co-located with the core development team, the frequency with which code drops moved between the core and porting team increased significantly. This allowed the porting team to keep in closer synch with the core team development resulting in a decrease in lag time for product availability from +/- 250 days down to 40 days. The co-location of the sub-team dramatically increased the level of informal knowledge transfer. The co-located team members gained a great deal of information via informal hallway conversations and had full access and participation in the core development team meetings. The co-located members were also given access to all core development resources (e.g. discussion databases, source trees, specifications, team meetings, etc...). This level of access was not granted to the remote team members, therefore, the co-located members of the team became gatekeepers of the knowledge transfer flow.

Finally, the core development team as owners and developers of the product source code held the ultimate power in the relationship. Source code is the physical representation of the product, and the source/result of knowledge in the software development community. .

Operational Data

The target operating system for the port offered an extremely immature development environment. The team was forced to make a significant investment in the development of homegrown tools to simulate commercial shared source code development environments. They also needed to create custom tools to migrate source code to and from the development platform of the core team.

The process for problem reporting was informal or non-existent in the early stages of the project. However, over time, a formal method of routing base product defects to the core team was established. Even at this point, there was no formal metric to measure the success/response to the routed problem.

The porting team had little input into the base product requirement process. Efforts by the porting team to implements specific features outside the process resulted in wasted effort and obsolete code due to parallel updates made by the core team.

There was a rigorous code review process in place to inspect all changes made by the porting team before integrating the changes back into the core source team. This process provided an important opportunity for knowledge transfer. It allowed members of the core development team to gain an understanding of the type and nature of changes required by the porting team. It also allowed the porting team to gain a greater understanding of the core platform architecture and coding standards. These meetings where viewed as very positive learning experiences by the porting team.

In terms of motivation, there were no explicit rewards and/or incentives in place to encourage the core development team to transfer knowledge to the porting team.

Technical Data

This product has an extremely strong and robust cross-platform architecture that defines portability as a fundamental design goal. This goal is evident in the low-level architecture of the product and the internally developed make/build tools used to compile and link the product.

From a technical perspective, the team possessed a strong set of language and operating system skills. Early in the project, the porting team discovered a key architectural interface of the product which was incompatible with the target operating system. A major portion of the initial porting work was devoted to understanding this technical boundary and creating an architecturally sound solution. It was a 2-3 month effort to understand the nature of the issue. An effective solution was developed that minimized the code updates to the base stream yet provided a clean interface to the operating system.

There was a heavy reliance on knowledge from operating system, file system and compiler/debugger development teams for the target platform, especially in the early stages of the port. This was attributed to the fact that this was the by far the largest porting project undertaken by the target platform. Consequently, many previously undiscovered boundaries and limitations where encountered. The porting team worked directly with the operating system and compiler/debugger team to resolve these problems.

As knowledge of the code and development environment increased over time, the team was able to automate much of the code rebase process. This reduced the time required to migrate new source trees from the core team to the porting team. ("rebase time)" The rebase time was reduced from 4-weeks to 2-3 days over the course of the project.

Another key technical insight recognized by the porting team was the importance of minimizing code changes. The team adopted a philosophy that measured success based on minimizing the number and scope of code changes required for the port. Open team inspections of all source code changes supported this philosophy. The goal was to limit changes to architecturally acceptable areas whenever possible. Also, when changes where required outside these areas, there was an attempt to make changes in such a way that a single set of source code would operate on all platforms.

Key Insights

Overall, both the development team responsible for the port and the management teams from both organizations viewed this effort as a success.

The resulting product drove significant revenue for the corporation and was recognized in the target market as a superior-performing product.

The key enablers of knowledge transfer for this port where:

- The co-location of the porting team with the operating system team in early stages of the port. This allowed frequent face to face communication and assistance concerning the development environment of the platform. This was especially important due to the immaturity of the platform development environment.
- Task interdependence between the two teams driven by periodic merges of the source code base. The code review/merge process was a major source of insights for the porting team.
- Strong knowledge of the operating system and development environment of the target platform of the port and close ties with key operating system personnel. These ties where supported by both personal relationships and formal management direction.
- The enhancement or created of development tools in order to increase programmer efficiency.
- The long-term duration and multiple release effort of the project allowed the team to achieve higher level of expertise and resulted in better team performance.
- Co-location of the sub-team with the core porting team dramatically increased the level of knowledge flow due to face to face interactions and full access to development data. These team members acted as gatekeepers or boundary managers for the remainder of the remote porting team.
- The focus of experienced porting team members on making cross platform, compatible changes or localizing and minimizing the required code changes when ever possible.
- Occasional face-to-face interaction by key technical members of the porting team with their counterparts in the core development team. This interaction increased the technical creditability of the porting team, greatly improving the

relationship, and the willingness of the core team to engage in additional interactions.

The key barriers of knowledge transfer for this port where:

- Immaturity of the operating system development environment.
- Lack of access to full development resources of the core development team by the remote porting team members.
- Little initial motivation for the core team to transfer knowledge to the porting team. There was some improvement seen in this regard as the changes made by the porting team appeared in the core development tree after code merges.
- Lack on input on the planning process for maintenance and feature releases cause porting team frustration and wasted effort resulting from attempts to implement core features outside of the process.
- Finally, the core development team, as owners of the product and the source code held the ultimate power in the relationship. Ownership of source code is the most fundamental source of power in the software development community.

Mid-Size Software Server Project

Organizational Data

This was a mid-size software porting project to move an external company's product to a UNIX environment enabling printer and file sharing between platforms. In the judgement of the 8-10 person team and cursory analysis of business results, this project ended in failure. A separate company located in another state owned the original source code base used in this porting effort. The owning company contracted with a firm in India to produce a portable version of the product. The team responsible for this porting effort received their code base from the India group. The porting team and the owning company had little direct interaction. There was a single interface into the owning company who was generally characterized as non-responsive.

Although there was little management support available from the India group, there was executive support from the owning product group and strong executive and local management support for the porting team. The porting team's home organization had a strong desire for a functional version of the product on their operating system platform. The owning product viewed the expansion of the platform base as a competitive asset.

The porting team consisted of highly skilled engineers with expertise on the target platform. This team had strong personal relationships with operating system personnel, and other experts from the community of practice in the porting to the target platform.

Operational Data

The target operating system for the port offered an extremely immature development environment. The team was forced to use homegrown tools, obtained from other porting projects, that simulated commercial shared source code development environments. They also needed to generate custom tools to migrate source code to and from the development platform of the core team.

The porting team had no structured process for problem reporting with the product owners and no input to the product requirements process. The porting team largely maintained the ported source code as a separate source tree with no formal review by the product owners. There was very infrequent transfer of source code changes back to the product owners. This changes were not feed back to the porting team in future code drops.

There was no sharing of development data repositories between the owning group and the porting group. The only shared communication vehicles between the teams were electronic mail and occasional conference calls.

Technical Data

From a technical perspective, this team possessed a strong set of language and operating system skills. The team had several members with extensive experience in the application porting domain.

The code base was approximately 1000 source parts developed for a non-UNIX operating system. The architecture and design of the product was not effective in terms of supporting cross platform development. There were large portions of the code which used #ifdef pre-processor statements to accommodate various platforms. Although, these pre-processor statements are found in all products, a well designed cross-platform product attempts to localize and/or minimize this type of coding approach. A more robust approach is to design and structure the source code so that a single set of source code statements will perform the required function for all platforms.

Due to many of the organizational and operational issues mentioned above, the source code of the porting team diverged significantly from the source code of the original product. As the product was put into production, significant scaling and performance problems where uncovered. Due to the diverged source base and architectural constraints of the product, there was no cost-effective technical solution. Consequently, the ported version of the product was put into "end-of-life" mode meaning development, enhancements and future porting activities were terminated.

Key Insights

Overall, this project was viewed as a failure by the technical team members and management involved in the project. Although there were several enablers to knowledge transfer, the barriers to the knowledge transfer effort prevented the ultimate success of the project. Some of the key factors which contributed to this result are as follows:

The key enablers to knowledge transfer were:

- ♦ Strong knowledge of operating system and development environment of the target platform of the port.
- ♦ Strong technical knowledge of porting project discipline

The key barriers to knowledge transfer where numerous, including:

- ♦ Lack of cross platform architecture of the product
- ♦ Lack of technical commitment from the product owners to the porting effort
- ♦ No sharing of design, specifications or other internal development data
- ♦ Little support for problem reporting/resolution from the product owners
- ♦ No merging of changed code back to the owning product team resulting in major source code divergence.
- ♦ Lack of input to owning product planning/requirements process.
- ♦ Immaturity of operating system development environment.
- ♦ Very little task interdependence between the owning product group and the porting team. Again, the core team, as owners of the product source code, held the strong source of power leverage.

PC Based Project

Organizational Data

The team charged with the port of this program was a small, 3-4 person team of highly skilled contractors engaged specifically to work on the project. The execution of the project took slightly over two years. The project was moderate in size consisting of approximately 1500 source parts. However, it was an aggressive effort for such a small team. The technical lead of the contract team had previously worked in the organization. He was well know and highly respected for his technical acumen. There was strong political pressure for a timely completion of the project. Consequently, there was strong management support for the contract team. The contract team was co-located with the core development team.

From a political perspective, there was little initial support for collaboration with the core team. As the original developers of the product, the core team was heavily invested in the code base as a fundamental source of power. There was little motivation to help the contract team port the code to a new platform. The typical attitude was to characterize the new platform as a distraction and that "contractors are getting the big bucks, so let them figure it out...."

There was strong task interdependence between code changes made by the two teams due to a common, shared source code base. However, as the "new guys", the contract team was largely responsible for management of the interdependence. The core team changes frequently "broke" to contract team build. The contract team was then responsible to correct the problems. However, it was viewed as unacceptable for the contract team to break the core team build. On the rare occasions on which breakage to the core occurred, the contract team was then responsible to correct the problems.

As the teams were co-located, there was a strong bias to face-to-face interactions as opposed to structured communications channels or reliance on management hierarchy.

Training was very much a hands-on, on the job effort. There was very little internal documentation or specifications available. Interview subjects attributed this to that fact that development team was rewarded on the output working code. The production of specifications where not part of the reward system.

Operational Data

Operationally, as mentioned above, there was a shared code base for development of the OS/2 and Windows version of the product. There was a shared problem reporting system as well. The porting team had full access to all repositories within the development organization. Upon completion of the 2 year porting effort, the source code ownership transferred to the core development team for future enhancement and support.

The goal of the project was to have full compatibility of results at the application level between the Windows and OS/2 versions of the product. The metrics used running side-by-side tests and insuring identical operational results between the core Windows product and the OS/2 port. In other words, the Windows version of the product provided the reference mode upon with which to measure the success of the OS/2 port.

Technical Data

From the technical perspective, the key technical expertise required was a strong knowledge of the OS/2 development environment. The basis of many of the porting problems was the coding standards used in the base product. The OS/2 compiler was much stricter in enforcement of ANSI standards, resulting in many coding constructs, which worked well on Windows, not compiling in the OS/2 environment.

The OS/2 porting effort also helped to undercover many base product defects. Correcting these bugs in the source tree was a strong factor in building creditability to of the porting team in the eyes of the base development team.

Key Insights

Overall, this effort was viewed as a success by both the development team responsible for the port and the management who engaged contractors. The resulting product drove significant revenue for the company.

The key enablers of knowledge transfer for this port where:

- Co-location of the development and porting team allowing for frequent face to face communication
- Shared source code base driving strong task interdependence between the two teams. Although the porting team bore much of the responsibility for conflicts that arose at the source code level, the shared tree did provide fundamental interlock between the teams.
- Strong knowledge of the operating system and development environment of the target platform of the port.
- Two year duration of the co-located team project allowed the team to reach a productive level of performance.
- Strong skills and overall creditability of the porting team members in the eyes of the development team.

The key barriers of knowledge transfer for this port where:

- Lack of formal specifications and documentation
- Little motivation for the core team to transfer knowledge to the porting team. The core team had little stake in the OS/2 version. Their focus was solely on the Windows product.
- Lack of architectural attention to portability in the base code tree caused an increase in the number of code changes required thereby increasing the overall complexity of the porting effort.
- Again the issue of code ownership as a power source of the core development team appears as a barrier to knowledge transfer which the porting team was required to overcome.
- Successful completion of the porting effort satisfied the porting contract resulting in the transfer ownership of the OS/2 of source code to the core development team. This eliminated the need for further collaboration.

Overall Porting Summary:
Overall, across both successful and failing software porting efforts, there are some key themes that where uncovered which effect knowledge transfer activities and the success of the software porting effort as a whole.

- **Strong affinity to Operating System specific skills** – across all the projects interviewed, the most important skill highlighted was operating system skills on the target operating system of the port. This includes knowledge of the runtime application programming interface (API), Graphical User Interface (GUI) API's, memory management, multi-tasking support, etc...
- **Strong affinity to development environment and language expertise** – Similar to the above, there is also a strong requirement for base skills in the programming language(s) used in the source code. More importantly, knowledge concerning the development environment on the target platform is key to the success of the team. A strong understanding and access to expertise concerning compilers, debuggers, make facilities, library management systems and other development environment tools is required. In general, the teams had this knowledge on entry to the project and/or relied heavily on others in the community of practice who had this knowledge.
- **Interactions with the Core Development Team** - Personal interactions with members of the core development team where highlighted as key to knowledge transfer throughout the interviews. In the case of the failing porting project, lack of direct interaction with the core development team was highlighted as an important barrier to knowledge transfer.
- **Task Interdependence** – The type and level of task interdependence between the core and porting team was key. Teams that had reciprocal interdependence via shared source trees or formal merges of source code changes where significantly more successful. The source code merge processes and/or a shared source code base were viewed as significant opportunities for learning and knowledge transfer. The knowledge transfer in

the case of shared source code was a two-way transfer. The both teams learned about the important differences and similarities in the coding practices and requirements of each other's platforms.

When there was no sharing or merging of source code a divergence of the code bases occurred. This resulted in a major barrier to knowledge transfer between the teams. The core team felt less knowledgeable about the newly created code base and felt it was not in a position to help when issue arose. The porting team also experienced a decrease in the need and/or desire for interaction with the core team.

Another key element of task interdependence is the level of shared processes between the porting team and the core team. This includes a wide range of items such as make facility, build scripts, library systems, problem reporting and requirements process. There was a direct relationship between the mapping of these operational processes and the success of knowledge transfer efforts. In the some cases where these processes did not correlate well, additional tools/processes where created as bridges. This appeared to be a successful approach as well. Lack of mapping and/or bridging was found to result in code divergence as indicated above.

- **Code is power –** All projects recognized that the core development team, as owners of the product source code, where in the ultimate position of power in the relationship. The teams were required to recognize and accept this fact in order to proceed with the porting effort.

Service/Support Project Background

Overview

As a development organization develops new products and/or new releases of existing products, they are marketed, sold and deployed by customers in the field. In the enterprise-computing environment, a key to a successful product launch is the level customer support for the product. In a support organization, there are typically three levels of customer support.

Level 1 is the initial point of customer contact. Level 1 is responsibility for authenticating the customer and insuring entitlement for support structure access. After verifying customer entitlement, the Level 1 representative will do in Problem Determination (PD) and may begin the Problem Source Identification (PSI) process. A general definition of these terms may be helpful at this point:

- Problem Determination (PD) – begins with the assumption that what the program action expected by the customer differed from the results that where seen. Problem Determination continues with the following steps:

1. Identify the Environment – This includes the identification of both the hardware and software environment under which the problem occurs.
2. Identify the Problem Type – which may include installation, setup, operation, performance, etc...
3. Identify Problem Area – This steps attempts to identify the failing component. (E.g. Installation/Configuration, Communications, Program API or other specific component of the product.
4. Problem Recreation - As the final step of the problem determination process, there is typically an attempt to try to re-create the problem. The problem recreation procedure and the probability of the recurrence of the problem are key factors in isolating it. If the problem is intermittent, any factors that the users feels are related to it becomes important information

- Problem Source Identification (PSI) – The first stage of the process (Problem Determination) deals primarily with an external view of the problem. The second stage of the process (PSI) is an attempt to try to identify the real cause of the problem. This step is largely dependent of the specifics of the problem encountered.

  The PSI phase begins to dig deeper into the problem systems and my include viewing of log files, trace data sets, return code, network data, messages, system dumps, etc...

Level 1 representatives are typically generalists, covering a wide range of components and/or products. The primary roles of the Level 1 service representative is to do an initial pass at the PD/PSI process in order to search a database of know problems to see a fix currently exists for the problem. If a fix for the problem exists, it is provided to the customer and the call can be closed pending feedback. If not, all the PD/PSI data available is collected and the call is passed on to a Level 2 service representative.

Level 2 as the name indicates, is the second level of contact for the customer experiencing a problem. Level 2 will continue the PD/PSI process initiated by the Level 1 representative. The Level 2 representatives are typically more specialized in a particular product or component than their Level 1 counterparts. As a result, they have the technical ability to dig deeper into the PD/PSI process doing an in depth analysis on all information available concerning the program fault and providing direction for additional information gathering at the customer site. Depending on the organization, the Level 2 representatives may or may not have access to the source code of the product. This additional analysis may lead to the discovery of a fix or other resolution of the customer problem. However, it may also reveal that the customer has encountered a new, previously undiscovered problem. In the case of a new problem, the Level 2 representative to gathers all the information available to date and passes the problem on to the Level 3 representative.

Level 3 representatives ideally work on new, unique problems reported by the field. Depending on the organization, the level 3 team members either work closely with development or in some cases are actually part of the development team. In the case of code defects, the Level 3 representatives are ultimately responsible for providing the source code changes required to correct the customer's problem.

The level 3 team will also collect groups of fixes together into a periodic maintenance release (MR) for the product. Some customers then use these maintenance releases as a preventive action. Also, service/support teams may suggest the installation of an MR as a corrective measure for a specific customer problem.

Service/Support Research Summary

Overview

For the Service Transfer portion of this analysis I interviewed key management personnel, technical leaders and individual contributors involved in the service and support roles for their organizations. The greatest insights for the purposes of this framework were gained by analysis of the three different organizations involved in the service support a single product. The three organizations perform level 2 and level 3 support for same large-scale Enterprise Server product. The first organization, **Core Support Team,** is a level 3 team that is part of the core development organization and co-located in the same office space. The second, the **Brand Support Team,** is a level 2 team that resides outside the development organization and is located in a separate building within the same campus. The third organization, **Remote Operating System Specific Team,** is a level 2/3 team located in a different state and focuses on problems for a single operating system. Figure 5: High Level Customer Support Problem Flow shows the relationship between these organizations for the typical support problem flow.

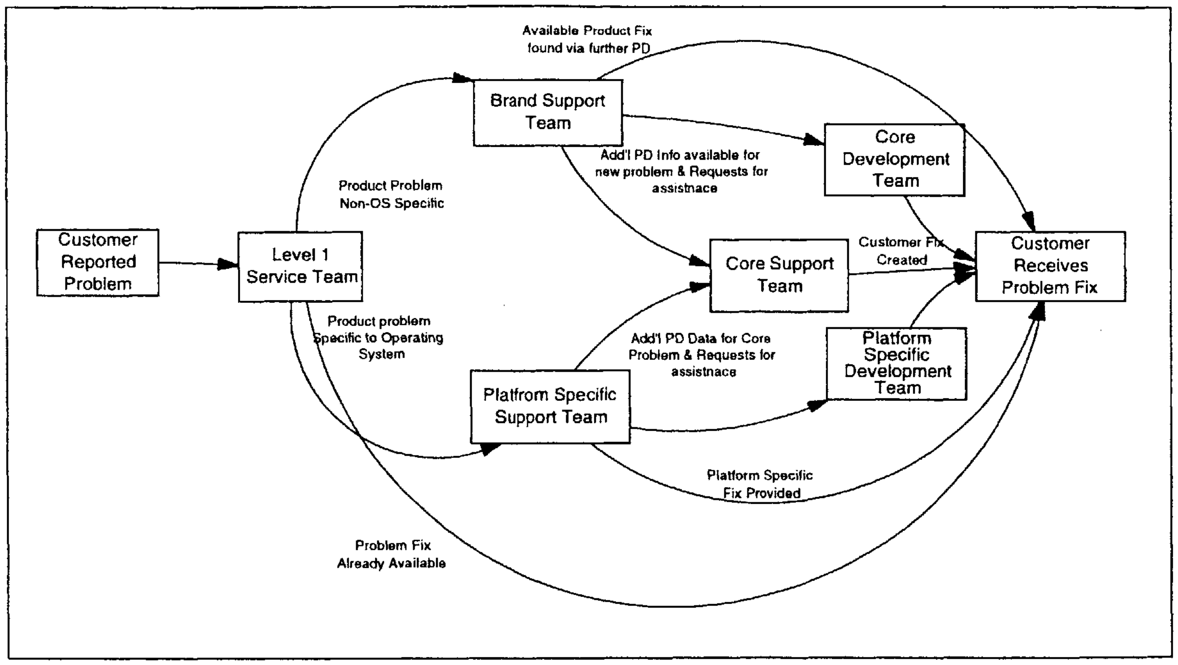

Available Product Fix found via further PD
Brand Support Team
Core Development Team
Add'l PD Info available for new problem & Requests for assistnace
Product Problem Non-OS Specific
Customer Reported Problem
Level 1 Service Team
Core Support Team
Customer Fix Created
Customer Receives Problem Fix
Product problem Specific to Operating System
Add'l PD Data for Core Problem & Requests for assistnace
Platform Specific Development Team
Platfrom Specific Support Team
Platform Specific Fix Provided
Problem Fix Already Available

Figure 5: High Level Customer Support Problem Flow

It is important to note that there is a significant difference, at the technical level, between the knowledge transferred in a porting environment compared to the knowledge transferred in a service/support environment. In the porting environment, the goal of the team is to replicate the behavior of the application. This can (and does) include the porting of software defects to the new platform. In the service support environment, the goal of the effort is to diagnosis failures found by customers in the field. Therefore, the knowledge transfer effort is focused on gaining insights into functional decomposition, operating system environment, diagnostic tools and techniques.

Core Support Team

Organizational Data

The **Core Support Team** is a 14-person team that is co-located with approximately 160 core product development personnel. The group typically handles 75% of the problems reported to the team. Requests are made to members of the product development team for assistance in addressing the remaining 25% of the calls. This is a technically diverse team that views themselves as generalists, although deeper analysis reveals that the support representatives do tend to a cleave themselves along operating system lines and/or functionally along major component lines. Generally, members of the team are capable of addressing a wide range of the problems reported.

The interviews conducted revealed a common theme of personality and learning traits displayed by individuals in the service/support role. Generally, members of the these teams are not as motivated by the monumental task of product development. These individuals are:

- Strong in diagnostic and debugging skills
- Strong in customer facing skills
- Interrupt driven
- Multi-tasking

Most of the team members where former members of other support teams outside of the core. These individuals bring their background in basic service/support skills and view the move into the Core Support Team as a step up given the close proximately of the development team.

Organizationally, there is a very loose hierarchy within the Core Development organization. Technical leaders within the Core Support Team consult with individual developers directly with management intervention used on an exceptional basis. There are no formal cross-unit work groups, however, informal networks and working relationships are strong in their usage. For example, one member of the support team is also a part-time member of a functional development team. In additional, there is job movement between the core development and support teams as well.

In terms of Information Technologies, the Core support team has total access to all development information technology resources. The team also has access to additional to sources of information externals to the development team. The access includes read/write access to source code within the development organization including the source pack for all Maintenance Releases (MR) of the product.

The strongest source of political influence is technical knowledge and expertise. This is the basis of individual credibility for team members. Overall, the relationship between the support and development teams is viewed as good. However, informants discussed the existence of an "air or superiority" among developers which is worthy of worthy of note. In the engineering pecking order, support personnel, responsible for service field releases, are often viewed as doing less creative or important work as compared to the development team, the creators of the product. This perception existed to some extent in all interviews conducted.

Culturally the service support team is a sub-culture within the main development organization. The core artifacts espoused values and underlying assumptions are well aligned due to the co-location of the support and development teams within a single organization. The customer focus (as opposed to development focus) of the support team forms the basis for the sub-culture in this environment.

An additional item of note is that although the interviews focus on knowledge transfer activities between the core development team and the core support team, the customer is a key stakeholder in the overall success of the support team. If knowledge transfer is viewed is a broader scope, the complexity of the analysis increases dramatically as the number of stakeholders increases.

Operational Data

As highlighted above, the operational alignment of the Core Support Team maps very closely to the Core development team. The support team has full access to all internal development education programs. All members of the Core Support Team have strong programming knowledge, technical capabilities and a solid understanding of the module level and data flows of the product. There is constant interaction and a strong awareness of the current operations within the development team.

In addition to participating in the internal development education, there is a heavy reliance on On-The-Job (OJT) training for support personnel. There is a strong, yet informal, mentoring program between senior and newer member of the Core Support Team.

Metrics used by the Core Support Team are currently under development. This is the result of senior management changes within the organization. At this point, analysis is beginning to look at:

- Total Calls Open by Platform
- Calls by Severity and Number of Days Open
- Total Calls by Severity

The current focus on metrics is on information gathering in order to understand the current state of the group in addition to what information is needed to run the business and make decisions.

From a change management perspective, the Core Support Team utilizes the same change management process as the development team. In addition, before submission, the development component owner reviews all code changes made by members of the service team.

From a requirements perspective, the Core Support Team ***owns*** the content of the Maintenance Releases (MR). However, the team has little influence of the content of new product releases. They have little input in terms of product quality and product RAS (Reliability, Availability and Serviceability) line items.

The goals of Maintenance Releases (MRs) is the timely injection of product fixes in order to improve the overall stability of the product operations in the field. The measure of success of the MR is judged by the results of customer installations in the field. When working on customer problems, there are typically many interactions between the customer and the support representative every day.

Technological Data

As addressed above, the Core Support Team has full access to all technologies used by the development team. However, one of the important differences uncovered between development and support is the environment for problem diagnosis. In the development environment, problem diagnosis allows for a strong hands-on approach including:

- Problem recreation
- Code stepping with debugger
- Re-compiled code with DEBUG flags turned on

On the other hand, debugging field problems does not afford such a pragmatic approach. Field debugging is typically done via post-failure data analysis. The customer's primary goal after a failure is restoring the system to a working state; thus allowing continued operations at the customer location. Problem diagnosis typically begins after restoring the customer's system to a working state. In the interest of high availability and continued operations, customers are reluctant to allow problem recreation and/or in depth analysis of the system in a failed state. Consequently, the support team must develop additional tools and techniques to analyze problems in this environment. The general feeling among team members was that the product was lacking in the tools required for the type of 'post-mortem' analysis.

As highlighted above, the core development and core support teams shared all development tools and process openly. This included:

- ♦ Code Repository Library
- ♦ Change Management System
- ♦ Development Tools
- ♦ Databases (discussion, specifications, documentation, etc...)

Key Insights

In summary, the core support team is extremely effective in terms of their knowledge transfer activities with the core development team. The key enablers of their knowledge transfer efforts are:

- ♦ Co-location within the core development team location and a shared cultural environment.
- ♦ Strong personal relationships between members of the core support and their development counterparts.
- ♦ Full access to all internal databases and repositories of the core development team.
- ♦ Write access to the source code providing 75% of the fixes in a maintenance release
- ♦ Strong knowledge of the product as well as specific operating systems.
- ♦ Core development team code reviews of all changes submitted in a MR. In addition to improving the quality of the release, this activity supports knowledge transfer as it provides a physical representation (i.e. source code) of knowledge as a discussion vehicle with the core development team.

The key barriers to knowledge transfer are:

- ♦ No input into the core development requirement process, which limits specific reliability, availability, serviceability (RAS) content of new product releases.

- Lack of service tools in the product.
- The difference in diagnostic technique used for remote diagnosis of customer problems, as compared to the hands-on methods used by development teams members, creates a barrier in terms of common ground and shared technical experiences.

Brand Support Team

Organizational Data

The Brand Support Team consists of 20+ people who reside in a different building within the same office complex as the Core Support and Development team. Approximately half the team is "code aware" meaning they have development level skills in programming in addition to some understanding of the product module and data flows. The other half of the team is comprised of senior support specialists with strong general analytical skills and good customer orientated communications skills. This team views themselves as generalists, however, there is some cleavage of the team along operating system lines and/or major product component area.

Interviews among this team again pointed to the specific personality and learning traits of many of the team members. Generally, the team members enjoy the challenge of multi-tasking and interacting with numerous customer IT representatives. It was highlighted that the time demands on support personnel are much more customer driven than their development counterparts. Where developers are faced with deadlines, the hours they commit to meeting those deadlines are usually within the control of the individual. However, in a support role, individuals must be available as the customer demand dictates, during the customers working hours. In fact, two members of the development team are on call (24x7) every five weeks to handle high-severity customer problems. The time demands required by the position are viewed as a barrier in attracting more highly qualified engineers to the job. There has been some success in attracting former developer to the position. The financial rewards and incentives of the support personnel are on par with those of their development counterparts.

Organizationally, the Brand Support Team is contained in a separate company than the Core Support Team/Core Development Team. As would be expected, the interviews pointed to a somewhat stronger reliance on formal management hierarchy in dealing with the Core Support team. Although formal management interfaces were categorized as sometimes helpful and sometimes not, the interfaces where generally viewed as positive paths for problem resolution and problem assistance. Management interfaces were viewed as improving and becoming more focused on the end-to-end product support environment.

The nature of the call flow, as shown in Figure 5: High Level Customer Support Problem Flow, requires communications and information flow between the Brand Support Team and the Core Support Teams. However, there are very few formal cross-unit work groups currently in place. The management and members of the Brand Support Team desire more formal and frequent interaction in order to build personnel networks with the development team. The boundary crossing relationships to the Core Support and Development teams are considered as one of the essential enablers toward the success of the Brand Support Team. The building of these relationships is strongly perched on the shoulders of the individual engineers, many of whom invest a lot of personal energy into the relationship. In the opinion of informants, the essential elements in building these relationships is individual technical credibility which is generated the following approaches:

- "Do your homework" by performing detailed problem analysis prior to contacting the core teams
- Be able to justify the importance of the problem in terms of business impact to the customer
- Have a positive attitude and confident approach when contacting development. View yourself as an equal.

- ♦ The more specific, code level analysis and discussion the better.

A second tier of boundary crossing individuals exists in the form of the Product Line Consultants. This is a formal group of individuals, which report in through the Brand Support Organization, and are responsible for interfacing with the development organization before new product releases. The goal of this group is to gain technical information about upcoming product content and features and work with Brand Support management to:

- ♦ Insure the required technical infrastructure (network, hardware, etc...) exists to provide diagnosis and support for new releases
- ♦ Define key technical development contacts for new features
- ♦ Define the formal problem escalation process for the new product

Currently, this information flows from development to the Brand Support team through the Product Line Consultants. The Brand Team has expresses a strong desire to influence the development direction based on support experience, but there is little evidence to support any occurrences of this type of influence. The Product Line Consultants are also working to create a consistent service strategy across releases and products supported by the Brand.

Effective negotiation skills where identified as an essential asset for members of the Brand Support Team. Negotiation skills are applied in the management of relationships with customers, field representatives, the Core Support Team and the Core Development organization. However, negotiation skills aside, technical expertise was consistently identified as the fundamental source of credibility when interacting with the development team.

Another key source of power within the Brand Support Team was unofficial contacts and personal relationships with the development team. The ability to quickly and informally contact development resources for assistance was viewed as one strongest assets of team members. Team members perceived a strong increase in overall effectiveness as the quantity and quality of the personal relationships with development increased.

Culturally, the gap between the Core team and the Brand team is narrowing. This is primarily attributed to a cultural change in the Core team due to recent explosion of growth in that organization. There has also been more movement of personnel into the core development organization from the brand organization. However, from the Brand organization's perspective, there is an essential difference in the espoused cultural values of the two organizations. The view is that the core team is driven by the development of cool, new technologies, while the Brand team espouses values of customer support and satisfaction. Although this is an obvious oversimplification, additional analysis on the cultural gaps is justified.

Operational Data

There is no formal training or education available for the Brand Support Team. Education from the Core Development team is considered as an after thought and is ad-hoc at best. Some training is available from development on a sub-component level, but this is not a formal requirement of service transfer. The Brand Support Team attempts to develop its own training programs, enlisting support from development where feasible. The support team is beginning to deploy training programs to assist the service representatives with their customer communication skills. This training includes telephone etiquette, verbal and written communications skills, reflecting skills, etc...

Operationally, there are several key metrics currently deployed by the Brand Support team including:

- Customer Satisfaction – goal of 85% of customers surveyed are satisfied
- Initial response time for call – Manually monitored by management
- Severity 1 time to resolution – goal of 7 days (resolution can be a Software Fix, work around or Hot Fix/Temporary Patch)

The support team does not have complete control of these metrics. Many exogenous factors affect time to resolution and customer satisfaction. Also, this metrics do not address the knowledge transfer between development and the service team or the quality of the work flowing across that boundary.

In terms of the product planning operations, the Brand Support team has little or no influence on the product plan or requirement process. There is a strong interest in obtaining membership in the product planning process in order to obtain information on future development and have influence on the content and direction of currently released and future products. The team would especially like to influence requirements in the area of reliability and serviceability. The Brand Support Team is also interested in details of future product direction in order to respond to customer inquiries and inform customer of committed features.

There are a number of process and procedures put in place to manage the relationships between the Brand Support Organization and the customers. The Critical Situation Team and Software Account Managers where viewed as having a positive effect on the both flow of information and the quality of the relationship between the support team and external customers.

The problem reporting process involved several disconnected systems. This anomaly required duplications of information when transferring data from the Brand Support Team to the Core Support Team. This disconnect was also attributed to an open loop problem systems which hindered tracking individual customer problems to a specific code change.

Technological Data

From the technical perspective, the Brand Support Team has a broad mix of skills on the team. As mentioned in the organizational data, approximately half of the team is code aware. The remaining team members draw from technical diagnostic skills and customer relationship skills. The team does have read-access to the code. The team uses the source code for informational purposes in order to gain insights into specific customer issues. This access was viewed as a major source of technical insights for the team. The support team also expressed a strong interested in abstractions of the code such a problem diagnosis approaches, dump reading maps, etc... Development provided little of this type of information and attempts where made to generate it in house.

The Brand Support Team does not make actual source code modifications. Therefore knowledge of the development environment among the team members was viewed as coincidental and not required.

The team relies heavily on diagnostic skills and operating system knowledge as discussed above.

Key Insights

The knowledge transfer activities in regards to the Brand Support team have a more complex nature due primarily to the increase in the number of stakeholders, the nature of the knowledge transfer requirements and the organizational boundaries involved.

Some key enablers effecting knowledge transfer in this environment include:

- Large investment in personal relationships with Core support team and Core development team, which form the foundation of knowledge flow between the groups.
- Product line consultants who "have a seat at the table" during the development of new product versions. This provides early insights to key training and hardware requirements.
- Strong diagnostic skills of support team members.
- Access to source code and some development data

Some key barriers to knowledge transfer include:

- Organizational and cultural barrier between the Brand support team and Core support team preventing a more free flow of knowledge.
- Limited understanding of product source code and lack of source code modifications by the brand team. In software development, the source code is the physical representation of the product. Without a complete understanding of the product source code, the representation of the knowledge during transfer becomes much more challenging. The knowledge sought is an abstraction of the source code into module flows, control block linkages, diagnostic techniques. In most cases, the knowledge has not been synthesized into this form.
- The metrics for success are not reflective of knowledge transfer activities and are heavily influenced by factors outside the control of the Brand Support team. For example, there are many exogenous influences on customer satisfaction and severity 1 time to resolution.

Remote Operating System Specific Support Team

Organizational Data

The Remote Operating System Specific Team is a separate service/support team that is located in a different state and different organization than the Core Service Team. However, the team is co-located with the development team responsible for the Operating Specific Port of the application. All members of the team have strong knowledge of the under-lying operating system environment. All members of the team have strong programming language skills and full access to the source code.

The relationship with the local development team is characterized as extremely good. Generally, local development doors are open to members of the support team. There is long-standing cultural support within the organization for this open door policy. Management principles within the organization dictate that if there is a direct conflict requiring a choice of supporting a customer in the field and working on new development, the customer will always receive the higher priority. In fact, members of the local development team are routinely called upon for assistance in customer problems. Members of the development team also take rotating assignments as members of the support team.

The relationship with the local team is quite helpful. However, when difficult problems arise, it is the Core Support Team that is approached for assistance. The Core Support Team has significantly more experience and technical knowledge about the product. The Core Support Team was also sought after for cross-unit work groups, primarily internships and temporary assignments.

From a product requirement perspective, the team has little input into the process. This is the source of much frustration for the team. There have been efforts to add reliability features to the product outside of the process. However, these efforts have been largely unsuccessful due to the dependence on the Core Source code base. Changes made outside of the process can become obsolete or be invalidated because of changes made to the source base within the owning development organization. The service team requirements primarily concern serviceability aids in the product.

The culture in the Remote Operating System Specific Team is significantly different from that of the Core Development Team. The Remote Operating System Specific Team is part of a large, hierarchical, process oriented organization. The Core Development Team is less formal and with a bias towards technological drivers over process.

Operational Data

One of the principle operational issues concerns the use of several different problem reporting systems by the various support teams. There were actually three different systems involved. There was not clear, automated, closed loop process to tie these systems together. This was viewed as a problem due to data and information loss when moving diagnostic data between the systems. Customer satisfaction was negatively affected by the discontinuity among problem reporting systems.

The support team did not have full access to the core development resources. This was seen as a major barrier to effective knowledge transfer. This was a particularly sensitive issue for the team, since support teams for most other products at this location did have full access to information from their development counterparts. In terms of access to information from the local, platform specific development team, full access was available.

The informants commented on the importance of on-the-job training. Temporary assignments and/or internships with the core support team were viewed as the most effective means of knowledge transfer. Again, the importance of the assignment as a vehicle for developing creditability and personal relationship between the core and remote support teams was viewed as a critical aspects of knowledge transfer. The lack of detailed documentation of module flows and control block structure increase the importance of these assignments.

Technological Data

A particularly striking statement from one of the team members was ***"Source code is knowledge***". This was an extremely insightful statement, since the product source code is the most fundamental, physical representation of the development teams' knowledge in software development. Manuals, specifications, module flows and other documentation are higher level abstractions of the source code.

The support personnel stressed the importance of platform specific knowledge as a key to addressing customer problems. Understanding the operating system and operating system specific diagnostic tools was viewed as a fundamental requirement followed closely by understanding of the application. There was further specialization of expertise below the operating system level. For example, there are recognized experts on dump reading, locking, memory management, communication and other platform specific diagnostic techniques.

As read only users of the source code, there was a stronger need expressed for module flows and control block documentation. These are the abstractions of the source code mentioned above. The need for these items was attributed to the breath of responsibility of support personnel. Development personnel typically work in depth on a single component. Support personnel are responsible for a much broader view of the product. Using the source code as the primary method of internal documentation for the product caused team members to have information overload. Although there were many repositories of information available, there was no clear map of the resource layout or understanding of where to look for specific information. This was a contributor to the information overload as well.

Another important technical issue uncovered was that much of the synthesized knowledge that was available was contained in private repositories such as an individual's e-mail folder. One informant suggested that having a person assigned to gather the data in a sharable spot would be of tremendous value. He viewed this person as a "knowledge manager". I suspect that this is a simplistic answer to the problem. The capture and sharing of technical knowledge is more an organizational construct than a task that can be assigned to an individual.

Key Insights

Overall, the Remote Support Team was effective in their knowledge transfer efforts by leveraging their relationship with the local development team and the Core support team well.

Some key enablers to knowledge transfer include:

- Positive and open relationship with the local development team, which had a strong code, level knowledge of the product internals.
- Temporary assignments and internships with the Core support team as essential element of hands-on knowledge transfer and relationship building across organizational boundaries.
- Access to source code. As one informant put it ***"Source code is knowledge"*** in terms of being the lowest level, physical representation of a developers knowledge.

Some key barriers to knowledge transfer include:

- Operationally disconnected problem-reporting systems, which lead to a loss of customer problem data and information.
- Lack of complete access to core development internal information. This was both a technical problem and a cultural issue since historically, the groups in the Remote Support Team had full and complete access to such information.
- Lack of synthesized information concerning product internals (e.g. module flows, control block structures, error codes, diagnostic procedures, etc...)
- Lack of a defined resource for collecting and disseminating synthesized knowledge to the team.

Overall Service/Support Summary:
Overall, this research uncovered a number of important themes that applied across the range of service support teams interviewed. These included:

- **Affinity to operating system knowledge** – similar to the porting projects, there was a strong prerequisite of operating specific knowledge on the part of service/support team members. The focus of the knowledge was on problem diagnosis techniques applicable to the platform, including dump reading skills, communication trace information, operating system memory management, multi-tasking management, etc... This affinity was most apparent in the tendency of teams to cleave specialization along operation system boundaries.

- **Personality and learning styles** - The interviews conducted revealed a common theme of personality and learning traits displayed by individuals in the service/support role. Generally, members of these teams do not have a strong attraction to the monumental task of product development. These individuals have a strong affinity towards diagnostic and debugging skills and strong customer communication skills. Support team members are generally interrupt driven and enjoy multi-tasking.

- **Interactions with core support and/or development** – Similar to the porting teams, service/support personnel noted these interactions as critical opportunities for knowledge transfer. These interactions were also viewed as a important opportunity to build credibility with core development personnel, resulting in a more positive working relationship and an increase in the frequency of interactions. Short term internships and temporary assignments were viewed as the most successful path for knowledge transfer interactions.

- **Source code abstractions** – Teams that did not make source code changes directly expressed a strong need for abstractions of the source code in terms of module flows, control block documentation, error flows, diagnostic "cookbooks", etc... This was the also case for groups that had read access to the code. This is attributed to the core knowledge transfer requirements of the service/support teams. For teams not making code updates, the knowledge sought was heavily weighted towards diagnostic as opposed to operational. Although the source code represents all the information required, and was helpful to the support teams, there was a generally feeling of information overload due to the wide breath of coverage expected by typical support representatives.

- **Stakeholders & Metrics** – Although the focus of this research was on knowledge transfer between the development and service teams, one of the key insights was the fact that there were other essential stakeholders involved. The most powerful stakeholder is the **customer.** The customer makes the ultimate determination of the success of the support team and therefore, indirectly, an assessment of the success the support team's knowledge transfer efforts.

  In a similar relationship, the metrics for success are not reflective of knowledge transfer activities. There are many exogenous influences outside the control of the service/support teams which effect measures such as customer satisfaction and overall response time.

Conclusion and Next Steps

Porting Projects:

The analysis of the porting projects focused on the assessment of knowledge transfer in regarding to the requisite knowledge required to move (port) the software code from one platform to another. Viewing the analysis of the three porting projects together, the following themes are paramount in the assessment of knowledge transfer efforts within the organizations.

Stakeholders

There two primary stakeholders in the porting projects in each of the successful porting projects; the core development team and the new platform porting team. As owners of the software product source code, the core development was consistently in the more powerful position.

It is useful to note that in the case of the failed porting effort; an intermediary group was placed between the core development team and the porting team. The porting team felt the intermediary group provided little value and attempted to bypass this group in favor of the core development team.

Goals

In a porting effort, the goals of the porting team are usually clear, and well aligned with the core development team. At the most basic level, the goal of the porting team is to duplicate the functionality of the product on a new target platform while minimizing the number of code changes.

Metrics

The metric used for success is duplication of operation between the core and ported project. In fact, the most common first step a developer on the porting team takes when debugging a problem is to attempt to duplicate the problem on the core platform. If the problem is reproduced on the core platform, it is not a porting problem, but a core defect. In most cases, ownership of the problem is passed onto the core development team for resolution.

Source Code as a Boundary Object

In porting projects, the source code is at the heart of the effort. As one porting team member put it, "***Code is knowledge***". Although this may be an overstatement, source code is an extremely effective boundary object as defined by Carlile. [16] As Carlile states, the critical elements of an effective boundary object are:

- *A shared method for individuals to represent their knowledge.* In software development projects, source code is the lowest level representation of the knowledge of a developer.

- *Provides a concrete means for individuals to specify their differences and dependencies.* Again, in software development, differences and dependencies are represented via source code updates. In fact, in software porting projects, differences and dependencies are explicitly enumerated via ***#ifdef*** pre-processor statements in the source code; or ideally the code is modified in to be appropriate for both platforms.

- *Facilitates a process for individuals to transform knowledge.* As highlighted in the interviews, the changed source code is typically input to the source code merge process. After the merge process completes, the core source code is modified (transformed) to include the changes of the porting team.

During the code merge, the reviewers and core developers gain insights into the nature of the new target platform. In the large-scale server port, evidence indicated that the initial merge of platform specific code into the core source tree gave core developers developed an awareness of the type and nature of issues encountered by the porting team. Consequently, some core developers attempted to avoid these issues in future versions of the product. This made future porting efforts more straightforward, since the knowledge of porting teams' prior work was now imbedded in the core development source code.

Viewing the results of projects without shared source code access provides additional support for the importance of the shared source code. In this environment, there was little knowledge transfer. Typically, without a shared code base, the source streams diverged resulting in project failure due to lack of compatibility or increased costs of development. Without the shared source code base, the best that can be said is that there is a one-time knowledge transfer which can be used to jump start a development team.

One final note on the product source code. Informants consistently pointed out that the core team draws significant power as the owner of the source tree. Even when the tree is shared, in cases of inconsistencies or conflicts concerning code modifications, the porting team bore the responsibility of resolving the conflict. The core team's response in conflict situations is the removal of the offending code. This allows the core team to continue as normal, but has a direct impact on the porting team.

Technical expertise on target operating system expertise
All porting teams interviewed stressed the importance of target operating system knowledge. This includes operating systems, compilers, debuggers, build environment, etc... In fact, knowledge of the target operating system was consistently viewed as more critical than knowledge of the product. Lack of maturity and compatibility of target platform development environment was also consistently identified as a key concern for the porting teams.

Cross-Boundary Personal Interactions
Personal interactions with members of the core development team where highlighted as key to the success of the porting efforts throughout the interviews. Members of the porting team viewed face to face communications as key to the knowledge transfer process. These interactions allowed team members to gain insights into product design and learn new features and/or changes planned early in the process. The interactions where also viewed as opportunities to build personal and team credibility with the core development team. Personal credibility was identified as reinforcing a feedback loop which increased the core development teams willingness to engage in additional personal interactions. As personal interactions increased, there was an increase knowledge transferred. As knowledge transfer increased, this had a positive effect on the individual's personal credibility continuing the reinforcing loop.

Software Support Project
The assessment of knowledge transfer between the core development team and the service/support teams interviewed proved to be significantly more complex than similar efforts between the core development team and the porting teams. The following sections summarize the important insights.

Stakeholders
In the software support projects, there were at least three primary stakeholders; the core development team, the software support team and the customer. The service team appeared to be the least powerful of the stakeholder given that the core development team position as the owners of the source code and customers roles as the arbiters of success for the service team.

Goals
The goals of the core development team and the service team are not well aligned. The goal of the service team is to provide customer support and/or correct defects in existing products. The goal of the core development team is

the creation of new software functionality. Of course, the development team recognizes the importance of customer support and correcting field defects, however it is not a primary goal of a development organization.

Metrics

No specific measures are in place to judge the success of knowledge transfer between the core development team and the service support team. The support teams' success is measured on their effectiveness on from the customer perspective. This significantly increases the complexity of the assessment.

Source Code is not an Effective Boundary Object

All of the service and support teams referenced the source code as an essential element in knowledge transfer. In the past, some of the team did not have access to the source code. The team escalated this issue through management channels to obtain access.

The Core Support team is the only support team with the ability to make updates to the source tree, the other support teams have read only access. The Core support team is viewed by the other support teams has an important knowledge source. In many instances they are sighted as equals to core development in terms of knowledge and assistance.

The teams with read access use the source code a vehicle for gaining knowledge and insights while debugging customer problems. However, the source code is no longer an effective boundary object in this situation. For the service/support teams, the code is a static representation of knowledge. The service support teams do not use the source code to represent their knowledge, specify their differences and no transformation occurs. This lack of a boundary object drives a strong need in the service/support teams for abstractions of the source code such as module flows, control block descriptions and diagnostic path maps.

Technical expertise on target operating system expertise
Although the members of the service and support teams viewed themselves as generalist, deeper inquiry consistently highlighted a bias to operating system specialization within members of the team. An understanding of the operating system environment and specific platform debugging approaches where essential elements of the overall team member's effectiveness.

Personal Characteristics and specialized skills
One of the key insights uncovered during the assessment was the identification of key differences in the motivation of service and support team members compared to their development counterparts. Informants consistently pointed out that strong diagnostic skills and customer facing skills are essential skills of service team members. The nature of diagnosis for customer related problems require a much heavier reliance on post failure data. Service and support personnel also appear to be more motivated by the frequent, short-term successes of resolving specific customer problems, where developers are more motivated by the longer term satisfaction of product feature generation.

Cross-Boundary Personal Interactions
Personal interactions with members of the core development team and/or core support team where highlighted as key to knowledge transfer throughout the interviews. Members of the service team who obtained internships or temporary assignments with the core support team viewed these as outstanding learning experiences. The benefits where expressed in terms of knowledge gained while on assignment and the development of a personal network of contacts for future assistance.

Similar to the porting team, the interactions where also viewed as opportunities to build personal and team credibility.

Beyond the Code

The research approach applied in this thesis has provided some useful and interesting insights concerning knowledge transfer in software development organizations. However, these insights are focused on the very narrow perspective of technical knowledge transfer between engineering professionals. During the interview process I could not ignore that fact that the knowledge transfer needs of these groups extend well beyond the technical exchanges outlined above. A short vignette, synthesized from discussions with several of the service/support managers may be helpful in illustrating this point.

> Jesse and I discussed the organizational, operational and technical aspects required to for his team to adequately address customer-reported problems in the code. Jesse said, “You know, we control the content of the maintenance releases. We have many good fixes available and we put out a new release every quarter. The thing is, the customers don’t install the maintenance releases as they become available.” Although, this was not specifically relevant to the discussion of knowledge transfer between the development and service teams, curiosity got the better of me and I asked, “What do you mean?” Jesse replied, “Well the customers treat each maintenance release like a major product release. They run it on a test installation first, and then it takes up to 6-months before it is propagated throughout their system. Meanwhile, they are calling us to report problems on production systems that are already fixed on the maintenance release they are testing in house.” Having gained an understanding of the volume of fixes (sometimes several hundred) that are included in a maintenance release, I asked if it was due to the magnitude of change the maintenance introduces. Jesse responded, “To some extent that might be the case, but it is mainly that some customers have been burned in the past and they just don’t trust the quality of the maintenance. They are also concerned about how the new version of our product will interact with other products they run in production. They purchase these products from our company and our competitors. It is a tough perception to overcome. The customers just want the individual fixes for the problems they encounter, but they don’t seem to understand the complexity of managing a system in that fashion. We can use testing to gauge the stability of a maintenance release, but the complexity generated by long term support of individual fixes becomes astronomical just due to the shear number of combinations. When you throw the other products into the mix, things really get crazy… ”

This short story raises some interesting issues. The customer IT personal, field support teams, business partners, competitors are now very powerful stakeholders. In addition, the operational and technical processes that generate the maintenance need to be understood in terms of how they support the goals of the stakeholders. When looking at the upstream and downstream influences on the process, it becomes apparent that there are some powerful feedback loops that effect the issues of trust and the customers' perception of product quality. There are essential needs for knowledge transfer within and outside the organizational boundary to support these efforts.

In general, as these teams are viewed in their overall business contexts, the number of stakeholders increases and the requirements of knowledge transfer explodes. A valuable insight uncovered during this research is the recognition that of several layers of knowledge transfer occur within the successful software development organization. Knowledge transfer requirements for the software development team exist between marketing, sales, competitive organizations, corporate strategy organizations, customer enterprises and many other stakeholders.

The first layer knowledge transfer is the fundamental transfer of core technical knowledge. This is the basic information required to perform quality development, service, support and/or porting of a software product and was the primary focus of this thesis.

The second layer, also addressed in this thesis, addresses the knowledge transfer interaction at the product and process level. This information flow contains domain specific information regarding product functionality (e.g. source code, specifications, architectural information, standards, etc...). In addition, this knowledge flow contains processed-based information regarding build procedures, internal problem reporting and requirements process, internal development information, training options, etc...

The third and most complex layer addresses the knowledge flow at the business process level. This includes such items as customer problem reporting and requirements processes, marketing and sales support, competitive influences, corporate strategy, etc...

Depending on the view within the organization, the key stakeholders change. However, a requirement for effective knowledge transfer between these stakeholders consistently exists. It is with this larger organizational context in mind that I offer the Software Development Knowledge Transfer Assessment Framework in the following chapter as a starting point for this analysis.

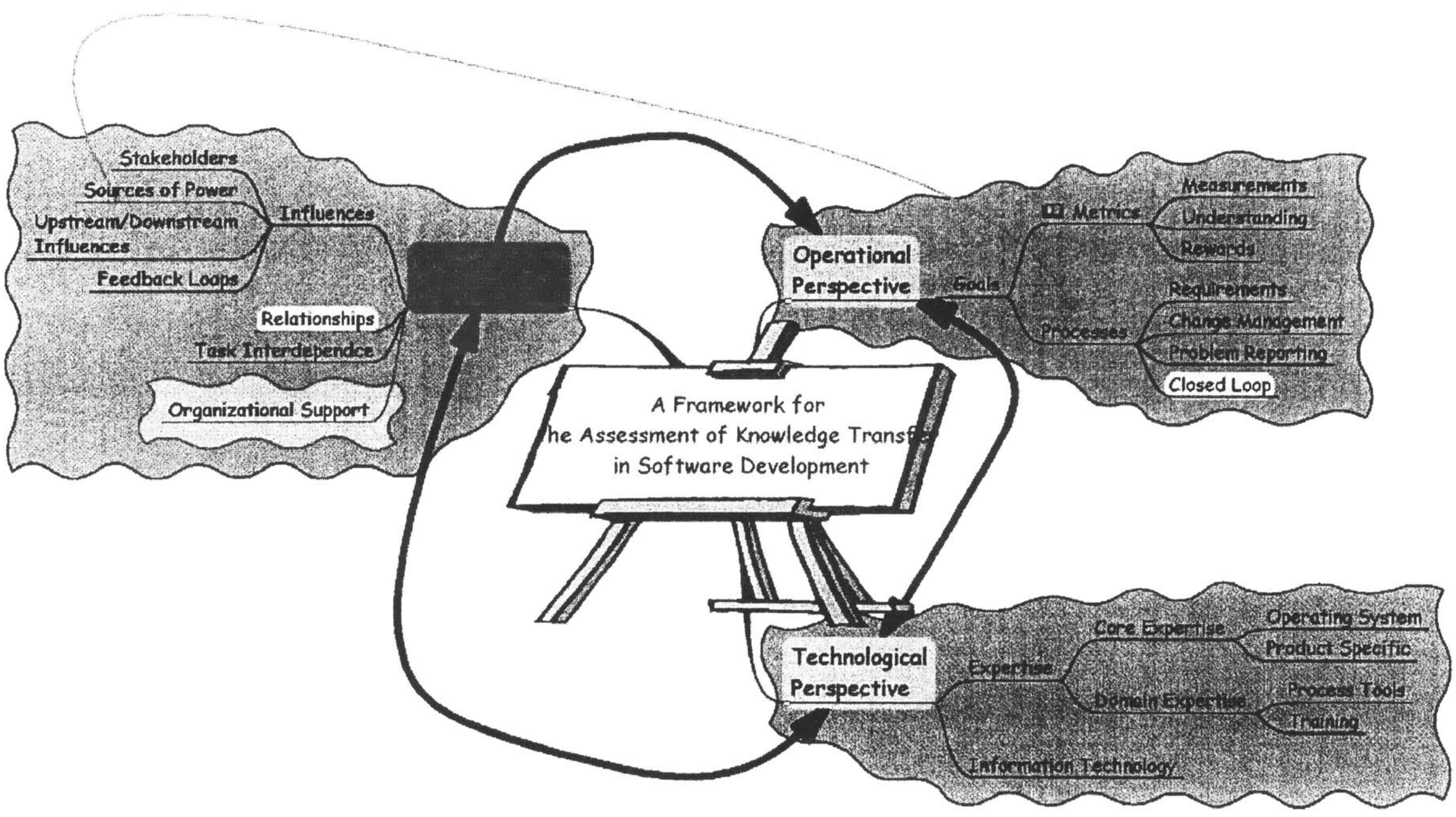
Software Development Knowledge Transfer Assessment Framework

Overview

The standard view is that knowledge transfer in software development is that it is fundamentally a measure of technical knowledge transfer. However, by applying the analysis framework within this thesis, it becomes apparent that a more holistic approach to viewing knowledge transfer is required to measure the true level success of knowledge transfer efforts within a software development environment. During the development of the research approach for this framework, it was simple (albeit artificial) to segregate the organizational, operational and technological trusts of the research methodology. This proved to be a very useful and productive path of inquiry and understanding. However, during the analysis of the data it becomes apparent that the three columns of this framework are tightly interwoven entities that form the basis of a holistic assessment for knowledge transfer in software development.

The results of interview experiences, applying the research approach outlined in Chapter 1, have generated a much more concise, yet still comprehensive framework. The three columns of support for the framework remain in tact. However, the breath of inquiry within each column has been greatly simplified. In addition to the simplification, a suggested sequence of inquiry is suggested.

Figure 6 : A Framework for the Assessment of Knowledge Transfer in Software Development



Knowledge Transfer Assessment Process

Organizational:

The inquiry begins with the organizational perspective. The first step in the process the development of a stakeholder map. Although stakeholders have a varied set of goals and interests, the focus of this assessment is on nature of the knowledge transfer flow of between the stakeholder and the organization under assessment. An individual stakeholder can play multiple roles over time. For critical knowledge flows, the stakeholder typically has upstream, downstream and feedback influences on the product or process. These influences must be specified. In addition to the nature of the knowledge transfer flow, the source and level of power for each stakeholder and the importance of the flow to the business operations must be captured. The result of this effort is a list of essential input, output and feedback knowledge flows for the specific product or process. Individual analysis of each knowledge flow occurs at the assessment proceeds.

Operational:

Using the output of the organization perspective, the focus of inquiry changes to the operational column of the framework. For each of the essential knowledge transfer flows, state the intent of the flow in terms of goals of the transfer. Next, identify the supporting processes and/or procedures for the flow and any metrics that can be used to gauge its status or success. When analyzing the supporting processes, it is important to map the processes to the upstream/downstream influence and feedback loops isolated in the organization phase of the assessment. Are all segments of the flow addressed? Are the metrics sufficient to gain an understanding of effectiveness of the flow?

Technical:
The Technical column of the framework has two major components. The first component is technical expertise. This includes the core skills in software product development and domain specific knowledge about the product under development. The second component of the technical perspective is information technology in place to support the knowledge transfer flow. The includes shared product build procedures, shared code and data repositories, collaboration tools, etc...

Overall Analysis:
At this point in the analysis, having drilled down on an individual knowledge transfer flow, it is useful to step back and view the components of organizational support for the specific flow. What strengths and weaknesses have been identified? In the broad sense, what organizational mechanisms are in place to support the transfer? These include:

- Management support
- Informal support networks
- Technical interchange and training
- Business intelligence data
- Boundary objects

The framework described above outlines three columns of support for knowledge transfer efforts in a software development organization. I believe that this framework is flexible and comprehensive in nature and applicable at all layers of knowledge transfer within an organization. As the number of stakeholders increases, so does the complexity of the knowledge transfer efforts. As the complexity and number of knowledge flows increase, a structured approach for inquiry becomes critical in order to gain a comprehensive understanding of the knowledge transfer issues facing the organization. As a result, this structured framework is proposed as a starting point for this effort.

Appendix A: Executive Summary

A. Problem Statement

Knowledge Management (KM) has been one of the most important topics of research for decades in management literature. Beginning in the mid-1990's Knowledge Management came to the forefront of software development and became one of the hottest topics in information technology. However, like all silver bullets that proceed it, the term Knowledge Management is beginning to evoke more skepticism than excitement in the minds of technical and business leaders. Larry Prusak, executive director of IBM's Institute for Knowledge Management, states that the primary reason for KM failures is that it is easier and faster to just buy the technology than to think through the strategic issues. It is the author's belief that understanding the strategic issues begins with knowledge transfer assessment of the organization.

In order to perform a realistic assessment of knowledge transfer in a software development organization, a framework that allows a holistic assessment of organizational, operations and technological perspectives is required. This thesis defines such a framework and applies it to two specific scenarios in software development; porting projects and software service/support operations.

B. Originality Requirement

This thesis develops a unique framework of inquiry for the analysis of knowledge transfer in software development organizations. This work leveraged nineteen years of personal experience as a technical leader in the software industry combined with interviews of executives, managers, technical leaders and individual contributors in the software development community. The result of the research is the creation a holistic methodology for the assessment of knowledge transfer in software development organizations. The assessment framework is comprehensive and applicable at multiple levels within the organization.

C. Content and Conclusions

In software development organizations, individuals at various levels tend to view knowledge transfer activities in a very narrow perspective based on position within the firm or personal bias. For example, engineers may view knowledge transfer primarily from the source code level. IT personnel view success based on the installation of Knowledge Management portals or other KM information technology tools. Managers may view the transfer in terms of organizational interfaces and/or formal escalation paths. Operationally focused individuals may base their opinions on the state of the process artifacts and the success or failures of ISO9000 audits. Each of these perspectives is valuable. However, none can present a complete picture of the state of knowledge transfer activities of the organization. It is necessary to perform a comprehensive inquiry that includes all of these perspectives in order to have a system level view of knowledge transfer within the organization. The result of failing to view these efforts at the system level range from sub-optimal knowledge transfer practices to dysfunctional software projects.

The common theme discovered during the interview process was the lack of a comprehensive approach to organizational analysis. The primary barrier preventing leaders from performing a comprehensive assessment of the organization's knowledge transfer efforts is the lack of structured inquiry tools. The development of a comprehensive framework of inquiry tools that provides a system assessment of the knowledge transfer in the organization was the goal of this thesis.

The framework for assessment of knowledge transfer begins with the organizational perspective using a stakeholder analysis to ascertain the upstream/downstream influences and feedback loops for essential knowledge transfer activities. The operational perspective views the intent, goals and metrics that support the identified knowledge transfer flows. The technical perspective in the framework assesses both the information technology infrastructure supporting knowledge transfer and the fundamental technical expertise required by team members to achieve adequate performance.

At the lowest technical level of the assessment, there a number of common issues that where uncovered including:

- complexity of knowledge transfer based on the number of stakehoders
- the alignment of goals and metrics within the organization
- the use of software source code as an effective boundary object
- an affinity for operating system expertise as a core requirement of effective knowledge transfer
- the value of personal interactions in knowledge transfer efforts and the feedback loop supported by these interactions.

Using interview data from executives, managers, technical leaders and individual contributors across a number of software porting and service teams, this thesis demonstrates the application of this framework for structured assessment of knowledge transfer in software development organizations. This thesis asserts that the assessment output provides managers and technical leaders with a system view of barriers and enablers to effective knowledge transfer. This information is then available to form the basis of prescriptive action plans.

D. System Design and Management Principles

The generation of the research approach applied for this thesis, coupled with the development of the framework for knowledge transfer assessment, has provided a unique opportunity to apply the core teaching concepts of the SDM program. This research effort draws heavily on system architectural principles for the development of the framework. System engineering principles where combined with the three lens analysis used to understand organizational processes and behavior to develop the research approach and the assessment framework. This combined effort was essential in developing the comprehensive nature of the assessment framework.

E. Engineering and Management Content

The research approach applied in this thesis drew heavily on over 19 years of personal experience in the software development industry. The thesis framework addresses a broad scope of engineering content spanning the range from software operating system development environments to system architecture axioms.

The research required a strong knowledge of hands-on software development tools (compilers, debuggers, library systems, optimizers, etc...) across a wide range of software operating systems and hardware platforms. In addition, the research draws heavily on knowledge of software engineering techniques required for effective development of cross-platforms software applications.

F. Statement of Authorship and Originality

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

Bibliography

[1] Noel Gallagher, *Stand By Me, recorded by Ben E. King. 1961.*
[2] Brooks, Fredrick P., *Silver Bullet – Essence and Accident of Software Engineering,* Information Processing 1986, the Proceedings of the IFIP Tenth World Computing Conference, Elsevier Science B.V., Amsterdam, The Netherlands, 1986
[3] Berkman, Eric, *When Bad Things Happen to Good Ideas*, Darwin, April 2000.
[4] Thompson, James D. 1967 *Organizations in Action,* New York: McGraw-Hill
[5] Carlile, Paul , *Managing for the Future: Organizational Behavior & Process*, South Western College Publishing, 1999
[6] Ibid.
[7] Senge, Peter, *The Fifth Discipline,* Currency Doubleday, New York, 1990
[8] Anacona, Kochan, Scully, Van Maanen, Wstney, et.al. *Managing for the Future: Organizational Behavior & Process*, South Western College Publishing, 1999
[9] Schein, Edgar, *The Corporate Culture: A Survival Guide*, Josey-Bass Inc, 1999
[10] Schein, E.H., *Organizational Culture and Leadership.* (1st ed.). San Fransisco: Jossey-Bass, 1985
[11] Schein, Edgar, *The Corporate Culture: A Survival Guide,* Josey-Bass Inc, 1999
[12] Bach, James, Satisfice, What Software Reality is Really About, Computer, December 1999
[13] Joel Moses is a computer scientist and Institute Professor at MIT.
[14] Crawley, Ed, *MIT System Architecture Class notes*, September 2001
[15] Tuchman, Bruce W. *"Developmental Sequences in SmallGroups"* ,Psychological Bulletin 63, 1965
[16] Carlile, Paul R.,*A Pragmatic View of Knowledge and Boundaries: Boundary Objects in New Product Development,* MIT Sloan School of Management, Auguest 15, 2000.