Lessons Learned in Transitioning to an Open Systems Environment

Dillard E. Boland, David S. Green, Warren L. Steger
Computer Sciences Corporation
10110 Aerospace Road
Lanham-Seabrook, Maryland 20706

Abstract

Software development organizations, both commercial and governmental, are undergoing rapid change spurred by developments in the computing industry. To stay competitive, these organizations must adopt new technologies, skills, and practices quickly. Yet even for an organization with a well-developed set of software engineering models and processes, transitioning to a new technology can be expensive and risky. Current industry trends are leading away from traditional mainframe environments and toward the workstation-based, open systems world. This paper presents the experiences of software engineers on three recent projects that pioneered open systems development for the National Aeronautics and Space Administration's (NASA's) Flight Dynamics Division of the Goddard Space Flight Center (GSFC).

Introduction

How can an organization effectively accomplish technology transition? Introducing a new technology into an organization requires an investment. But what is the nature and size of that investment, and how long will it be before benefits are realized? How can one quantitatively define these benefits and measure the results? Whatever the ultimate reward of the technology, transition is a step into uncharted waters. Technology infusion requires managers to rethink the way they approach the ordinary project management challenges of developing effort estimates, achieving planned productivity, and dealing with evolving requirements.

The authors of this paper develop software systems under contract to the NASA/GSFC Flight Dynamics Division (FDD). For more than two decades, the FDD has successfully fielded software systems to support NASA spacecraft missions in a relatively stable mainframe/-minicomputer environment. This stability has allowed the FDD to optimize its software development process. During the first half of the 1990s, the authors worked on three projects in the forefront of the FDD's transition from its legacy environment to a workstation-based open systems environment. We discovered that our established development process had to transition as well, in unanticipated ways. Our experiences in this transition and our lessons learned are recorded here with some recommendations for managing technology transitions.

A model commonly used for technology transfer conceives of technology as moving from a producer to a consumer organization. The transition moves through the phases of early experimentation and exploration to technical maturity. The projects discussed in this paper fall primarily within the exploratory phase, where work has progressed from initial experiments to full-scale development, but the technology is still used by a

minority of the organization's staff. Marvin Zelkowitz defined these phases in a paper presented at the 18th Annual Software Engineering Workshop, "Software Engineering Technology Transfer: Understanding the Process."

This paper provides information on the software development organization, then summarizes our observations on each of the case study projects. We then organize the lessons learned and recommend elements of a technology transition plan and ways in which new technology projects might be better managed.

The FDD Software Development Organization

The FDD entered the transition with a mature software development organization that included the Software Engineering Laboratory (SEL), a research and process improvement group whose mature measurement program, cost and schedule estimation models, and management guidelines support software development and technology transfer in this environment.

The FDD had patterned its success on a basic scientific method of gradual, continuous improvement in software engineering technology in a stable computing environment. Controlled innovations were introduced to test new techniques and tools. Studies usually were conducted through pilot projects that applied the new technology under strict controls, with the results evaluated against the organization's norms. The FDD would then incorporate proven beneficial technologies into the standard technology suite.

The FDD had made little investment in exploring open systems technologies. The FDD's few projects outside the mainframe environment were considered out of the organization's mainstream. Developers collected few statistics, and few software engineering experiments were conducted on these projects. When the computing industry began to shift toward workstations, the C language, and open systems concepts, the FDD had little background in these technologies.

Since 1990, the FDD has been moving toward workstation computing platforms and open systems technology, driven primarily by factors external to the development organization. They have done so without the benefit or lead of SEL experiments. Figure 1 illustrates the FDD's

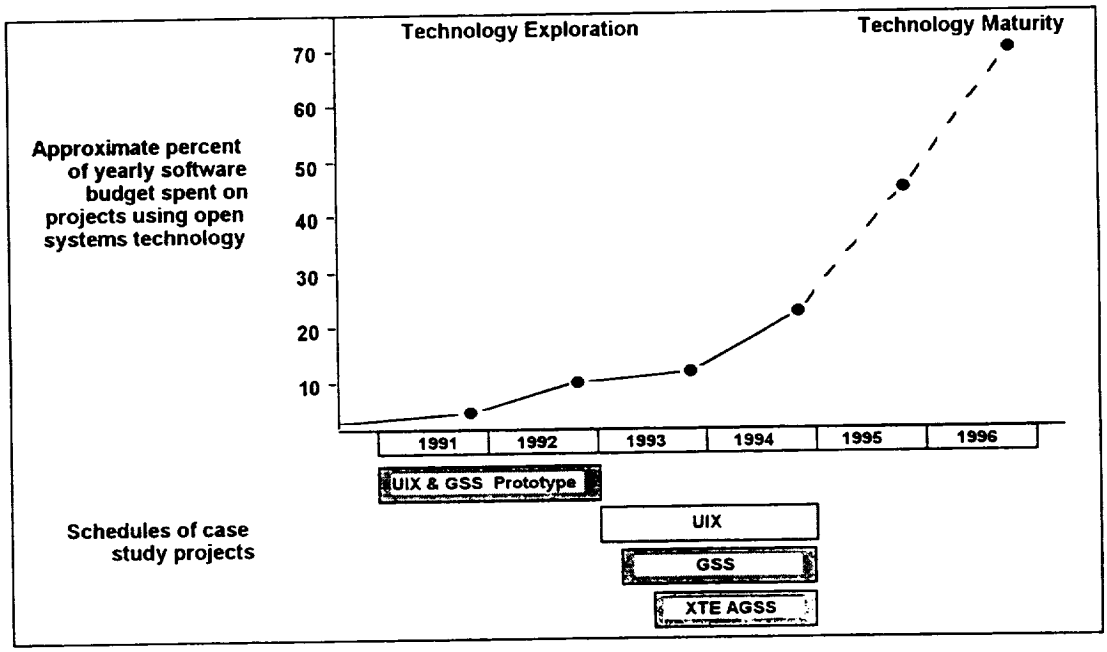


Figure 1. FDD Transition to Open Systems

investment in new technology exploration and the quickening pace of the transition. The case studies discussed in this report are shown at the bottom of the figure in their chronological context.

The Case Study Projects

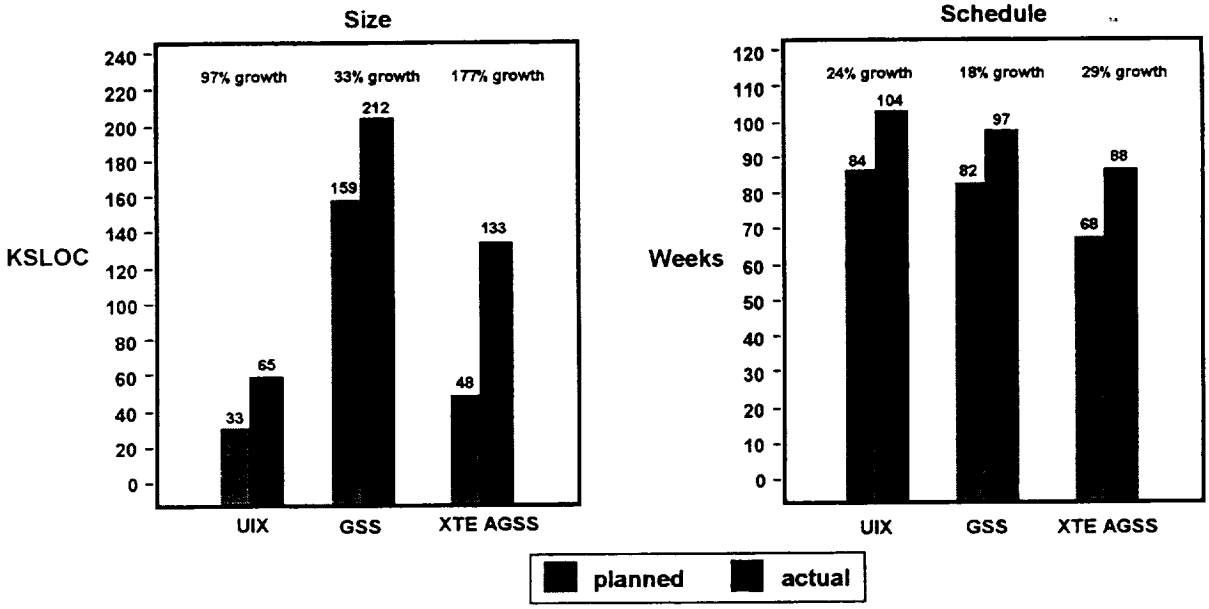
Table 1 provides an overview of the three case study projects, listing the size and language, operational computing environments, and development tools. The projects were planned by tailoring the domain-specific FDD cost and schedule models. The tailoring allowed for some training on specific new technologies. As work progressed, plans were revised to reflect the technology issues. Figure 2 summarizes the development results compared to the plan.

Case Study 1: User Interface Executive (UIX)

The FDD saw a need for a common framework in the new environment. The FDD planned the UIX as a common user interface and executive framework for distributed mission support systems. The decision to base the user interface on X/Motif was primarily driven by industry trends. The aim was to create a configurable system to be used by developers working in Ada, C, or FORTRAN to build application programs that shared a common set of interactive tools. The application developers would not be required to code in X/Motif or to use a GUI builder. The UIX would allow application users to control multiple, distributed processes in a platform-transparent manner. Finally, the FDD required that the UIX support existing hardware

Table 1. Case Study Project Characteristics

Project Descriptors	**Case Study 1: UIX**	**Case Study 2: GSS**	**Case Study 3: XTE AGSS**
Size in KSLOCs and Language	65,000 C	212,000 Ada	66,000 C 58,000 FORTRAN 9,000 User Interface Language (UIL)
Platform and Infrastructure Software	♦ 386 and 486 PCs ♦ Santa Cruz Operation (SCO) UNIX ♦ HP 9000/7xx series workstations ♦ HP/UX ♦ External Data Representation (XDR) ♦ X/Motif (X11R4, later R5)	♦ Digital Equipment Corp. (DEC) VAX 8820 (later Alpha AXP/4000), open VMS ♦ 486 PCs ♦ SCO UNIX ♦ HP 9000/7xx series workstations ♦ HP/ UX 9.0.3 or higher	♦ Hewlett-Packard (HP) 7xx workstations ♦ HP/UX ♦ X-terminal and VT2000 emulation ♦ X/Motif (X11R5)
Development Tools	♦ Intersolv PVCS version control ♦ SCO Open Desktop toolset	♦ DEC Configuration Management System (CMS) ♦ DEC VAXSet Development Toolset ♦ DEC Ada Compiler Version 2.2 ♦ Rational Software Corp. VADSelf Ada for 486 SCO, HP/UX	♦ Builder Xcessory (X Window GUI builder) ♦ HP full-screen editor ♦ HP desktop environment



Compare to size growth in 20% to 40% range and schedule growth in 5% to 35% range on recent maintenance and VAX development projects

Figure 2. Planned Versus Actual Size and Schedule

(the IBM mainframes and Intel-386 PCs) to the maximum extent possible.

Prototyping played a critical role. The ambitious goals of the UIX project were all the more challenging because it was the first to use open systems technology within the FDD. To learn the technology and refine the requirements, the development team built a prototype that covered all major facets of the proposed UIX. Development and evaluation of the prototype ultimately spanned a year and a half. In parallel with the prototype evaluation, the team began specifying the content of the actual UIX. The prototyping experience led to architectural and conceptual changes in the specified product, including abandoning the goal of supporting the IBM mainframe as an application host and deferring implementation of distributed process control until industry capabilities had further evolved.

Lack of a technical infrastructure and an organizational transition plan caused difficulties. Without a preestablished infrastructure ("middleware" such as a network file server), the traditional separation of concerns between the systems support and software development organizations was blurred. It was sometimes unclear whether responsibility for selecting an infrastructure product lay with the project that first needed the capability (in this case, the UIX) or with the support organization that maintained the FDD's institutional hardware resources. Although cross-organizational groups addressed these issues, the lack of an overall transition plan led to misunderstandings and organizational friction.

The FDD's traditional functional requirements and specifications methodology was not sufficient for establishing the infrastructure. Software developers, especially those from mainframe backgrounds, tend to take the existence of a computing system architecture for granted, but this was not the case with the UIX. The developers attempted to define the required software infrastructure using data flow diagrams and functional specifications, the method with

which they were familiar. Unfortunately, their limited knowledge of the technologies involved and the immaturity of available products muddied the development effort. One round of prototyping followed by one round of specification development was not sufficient, nor was the specification formalism conducive to iterative refinement.

Prototyping experience led to technical learning but not better planning. Although the prototyping experience clarified technical issues, it taught the developers little about planning the development project. They believed that the effort saved by rapid prototyping would offset the additional effort needed to come up the learning curve on the new technologies. In the actual project experience, there was still a substantial learning curve in spite of an overlap of development team members with the prototyping team. (For example, the complexity of X/Motif coding was underestimated.) The prototyping team achieved the organization's average productivity based on historical data. However, productivity on the actual UIX development was initially only half that of the prototype project, as the team faced continued technical learning as well as the documentation and inspection demands of a disciplined development methodology. Furthermore, the final system was larger (by a factor of about two) and more complex than indicated by the prototyping.

Case Study 2: Generalized Support Software (GSS)

The GSS project transitioned the post-integration development phase only. The GSS is a multiapplication flight dynamics support class library designed to interface with the UIX. The GSS project was the FDD's first Ada language software development project to make the transition to the open systems workstation environment. Unlike the other two case studies presented in this paper, the GSS was not developed in an open systems environment. The GSS was designed, coded, and integrated in the standard development environment for Ada-based software projects in the FDD, which was a DEC VAX system (later, a DEC Alpha system). The code was then ported to the SCO UNIX environment on PCs for integration with the UIX to create the operational system (an attitude telemetry simulator), with the UIX providing the user interface services. Thus, the technological "leap" taken by GSS was considerably smaller.

The infrastructure needed for a workstation-based development was underestimated. When the GSS project started production in January 1993, the FDD did not have sufficient workstations and associated Ada development tools to support a development the size of the GSS on workstations. The GSS project was not budgeted to procure the workstations and tools needed to develop the system totally in a workstation environment. FDD management decided that the most cost-effective approach would be to develop the GSS software on the institutional Ada development platform, a VAX 8820 minicomputer, until the build integration test phase. At that time, the software for the build would be ported to the workstation environment.

A familiar development environment helped control system growth. The growth in size of the GSS is fairly consistent with FDD projects over the past 5 years. The reasons for the relatively limited growth compared to the other case studies are

- GSS is developed in Ada, a language FDD software developers have been using for almost a decade.
- The developers were familiar with the GSS development environment and toolset, and only the latter phases of the life cycle (build integration through independent test) were performed on the workstation platforms.

The GSS project comprised pure computational applications software, not interactive software. The GSS project did not have to deal with user-system interface issues in the new open systems environment. Because the UIX system provides the GUI for GSS-based flight dynamics applications, the GSS project was "shielded"

from many of the technological hurdles and learning curve relating to building GUIs on workstation platforms. This experience suggests that scientific application development is less affected when moving to open systems platforms than is user interface software development.

Case Study 3: X-Ray Timing Explorer Attitude Ground Support System *(XTE AGSS)*

The FDD faced a new requirement to deliver software on workstations. On this project the FDD developed mission attitude ground support applications in an open systems workstation environment. The FDD had developed these types of applications before but only in an IBM mainframe environment. The FDD was required to deliver the applications to a separate GSFC organization, the Mission Operations Division (MOD), for integration into their operational system. Such applications had previously been installed and operated only within the FDD environment. The MOD systems use a locally developed package called Transportable Payload Operations Control Center (TPOCC) to provide the client-server framework.

Project planning was largely based on experience in the legacy environment. The project planners estimated size (in lines of code) of the applications based on previous FDD systems. The planners determined they could reuse a large amount of FORTRAN computational code being developed concurrently on the mainframe. Since XTE AGSS was a first-of-a-kind project, the planners lacked good comparisons to help estimate how the use of TPOCC and X/Motif graphics would affect the size. A productivity rate 20 percent lower than the FDD norm was used to account for the new technology learning curve.

The XTE development effort was significantly underestimated. As it turned out, the size of the applications was underestimated by a factor of three, primarily because

- Planners underestimated the size of the TPOCC and graphics-related code.
- Reused code was larger than expected.
- Requirement changes added major new functionality.

Productivity on the initial builds was considerably lower than expected. The main causes of the lowered productivity were underestimation of the complexity of the new technology, the lack of X/Motif expertise on the team, and skill mix problems. Productivity increased in the later builds as the team became more experienced with the technology and as the skill mix improved; some builds met or exceeded the FDD norm.

The traditional methodology had to change to incorporate iteration. Only about half the unit designs had been completed by the time of critical design review. (FDD methodology called for all unit designs to be complete at that point.) This indicated trouble, but the developers and their management did not realize the full extent of the effort underestimation until the coding phase. Then it became clear that they could not complete the project according to the original plans, and they had to renegotiate the delivery schedule and add staff. The new schedule was still highly compressed because of XTE mission deadlines, forcing the developers into an iterative approach of designing and coding build by build. For the most part the iterative approach worked well, though it made assessing progress difficult.

Requirements instability exacerbated problems. It is common in FDD development projects that software requirements evolve during the course of development. The XTE project encountered challenging, though not unprecedented, requirements instability, partly because the FDD analysts thought of ways to make the software more generic well after design and implementation were underway. System specifications were changed on several occasions to serve the best long-term interests of the FDD. The resulting perturbations were far more severe than they normally would have been because the project was in technology transition.

The development team needed immersion in the technology to come up to speed. One of the major challenges of the project was learning the TPOCC system. This amounted to technology transfer from the MOD to the FDD. The TPOCC system is large and complicated, and the XTE development team could find no single person who was expert in all aspects of the system. Early in the implementation phase, part of the development team relocated to the MOD development facility for 2 months. The relocation was very useful for promoting communications, though interaction was limited because the MOD developers were busy with their own projects. The early builds implemented the TPOCC interfaces and were kept relatively small to allow quick feedback. To get a testable framework in place, the team split the first build in two when it turned out to be far larger than planned.

Unrecognized technological assumptions created transition problems. The biggest problem encountered with TPOCC was not in implementing the application interfaces, but in installing TPOCC in the FDD. Differences between the MOD and the FDD computer environments and system administration approaches became evident. For instance, the FDD used network user accounts, with which TPOCC was not compatible. Other problems developed when the MOD moved to new releases of the HP operating system and Motif before these versions were available to the FDD. In retrospect, the memoranda of understanding between the FDD and the MOD, which only addressed XTE AGSS release dates, should have also specified TPOCC version delivery dates, versions of system and support software to be used, and all applicable standards.

Increasing personal interaction and emphasizing skill mix helped alleviate problems. After the FDD tested the releases in-house, the plan called for delivering them to the MOD for integration into the operational environment. Because of all the unexpected problems encountered thus far in the project, the FDD development team decided to work with the MOD developers informally to integrate the system before formal delivery. The main problems found during informal integration and testing were with installation instructions, not with the software itself.

A final factor very important to the success of the XTE AGSS was staffing. Once the true magnitude of the development effort was understood, project management committed highly experienced and motivated individuals to the team. They provided a good skill mix that included both software development and application domain knowledge and C and FORTRAN experience. In spite of the pressures, this commitment led to a very good team spirit and a successful product.

Lessons Learned

The complexity of open systems was much deeper than anticipated in all three case study projects. The developers learned that "industry standards" are often evolving or competing conventions, that COTS products are marketed before they are mature, and that interoperability does not always live up to advertised expectations. They discovered how much middleware it really takes to make a distributed system work. The organization realized how significant the choice of hardware is to the viability of the final system, how much hardware is needed to fully support a distributed development effort, and that the costs for support software and development environments can rival or exceed the cost of the hardware. They also had to find ways to overcome compartmentalization of open systems knowledge in their own and in interfacing organizations. We have grouped these lessons around organizational, technological, and managerial themes.

Organizational Lessons

Organizational transition plan. A planned transition for the entire organization, backed by management commitment, is needed. The case studies indicate that the FDD approached the transition on a project-by-project basis, not only

reducing coordination but also slowing the dispersion of knowledge. Management did attempt to coordinate activities at the top levels of the organization, but the staff on the individual projects received little information as to how their project fit into the plan. As a consequence, people focused almost exclusively on the challenges of using the new technology on their own projects, with little incentive to share their experiences with others in the organization.

Changing organizational roles. Changing technology can blur traditional roles, garble communications, and cause friction. No doubt this is part of what makes transition plans hard to create in the first place. Effects of technology change can ripple across organizations in ways they cannot readily accommodate. The leaders of the organization must define and communicate a vision for doing business using the new technology and help the staff make organizational changes stemming from it. Changing technology does not necessarily mean business reengineering, but if the organization is making a major technology change it should carefully evaluate the impact on its business model as well.

Outreach across organizational boundaries. Sharing experiences across project and department boundaries is critical during technology transition. "Department" here means any portion of the organization that traditionally practices "information hiding" from other portions. The case studies show that information barriers can exist even at the lowest levels. Groups of 5 or 10 people down the hall from each other may not share information even though they are engaged in parallel transitions. This may seem counterintuitive to anyone who has experienced the "office grapevine," but people do not grasp organizational plans through the grapevine. Personal contact works well for transferring detailed knowledge when people have a focus and goals, but it takes a special effort to find that focus. Management must provide forums, whether formal or informal, for sharing new technology experiences in real time without "turf" issues interfering.

Disseminating lessons learned. The FDD has a tradition of writing good history documents after each project to capture lessons learned, but often they come out too late to help the project planners who really need them. Also, if a procedure for using them is not integral to the development methodology, the lessons may sit on the shelf unheeded. An organization should document lessons learned at points in the development process well before the project's end and should make producing and using them part of the development procedure. The lessons should be disseminated in a way that will make them easy to access (for example, in a cross-indexed on-line library). The goal should be to coalesce the lessons into an institutional knowledge base.

Technological Lessons

Cultivating market awareness. The competitive marketplace drives the evolution of open technologies, so using them effectively requires cultivating and maintaining market awareness. An organization coming from a stable mainframe environment that does not emphasize compatibility with the world beyond the vendor may be a "closed shop," especially if that organization produces a very specialized product (such as space ground support systems). The case studies suggest that the FDD was not prepared to deal with rapid market evolution. In the past, the organization usually had time to choose technologies carefully and experiment with "seed" projects. This approach was not geared to the pace of change the developers had to adopt to accomplish the transition to open systems. The transition forced a cultivation of market awareness, which in turn requires applying the discipline and resources to track all aspects of industry evolution. Management must actively encourage technical staff to follow market trends and pursue continuing education.

Training for front-line workers. Beware of unrealistic optimism on the part of both managers and technical staff regarding the ease with which staff can master the new technologies. The case studies revealed that people had a tendency to think in terms of distinct skills to be

learned, new, but similar to existing skills. In reality, the myriad interrelationships of a new suite of technologies, and the industry context in which they are evolving, are very complex. Our experience was that the amount of ramp-up time needed to learn new technologies, from least to most, was for UNIX, C, networking, and X/Motif (most difficult to acquire even using a GUI builder). When most of the team has to learn all the technologies together, the time invested is significant.

Technical compatibility. When a software development shop first adopts open systems technology, it may expect to easily interface with open systems in client and peer organizations. This expectation was not realized in the case study projects; "plug and play" is not yet the norm. Incompatibilities result if the organization does not have detailed knowledge of the technologies used by the interfacing organization. Open systems invite cooperation but do not guarantee compatibility. Interacting organizations should discuss and document their agreements on issues such as standards, COTS product versions, and configuration management assumptions.

Retooling the infrastructure. Organizations such as the FDD with long-standing stable computing environments have usually developed customized software development toolsets and a supporting infrastructure. When moving to a new technology, problems that were previously solved in the legacy environment may need to be solved again because the infrastructure and tools have changed. Even a technically mature organization may be unprepared for the extent to which it must develop new approaches to basic software engineering problems that it thought it had solved long ago. A mature organization may be at a disadvantage because of a high comfort level with its proven techniques.

System engineering. In all three cases studied, the transition to open systems caused the developers to shift from a purely software engineering viewpoint to more of a system engineering perspective. In the absence of a stable technical infrastructure, the developers had to devote considerable time and effort to understanding engineering topics for which their previous project experiences had not prepared them. Both hardware and software components had to be treated more or less equally. Emphasis shifted from crafting systems from lines of code to selecting and integrating the right combination of hardware and software components. When no established computing infrastructure exists, developers must perform systems engineering analysis at the start of the project to plan for and procure sufficient resources.

Project Management Lessons

Realistic expectations. Project managers cannot expect to achieve all the goals during a technology transition that the organization achieved in the stable technology. Aiming for these goals can lead to over-commitments and compromise the success of the transition. The project manager must be strategically aggressive but tactically conservative, and careful when making commitments.

Accurate effort estimation. Technology transition requires investment. The SEL *Manager's Handbook*, source of the FDD's project estimate models, recommends applying an additional effort multiplier of 2.3 when a project type and the technical environment are new to the organization. Had the case study projects followed this guidance, the UIX and XTE AGSS projects would have started with much more realistic effort estimates. The GSS project, which did not involve the same degree of transition as the other two, came closer to the standard model, and the effort multiplier may not have applied to it.

Staffing and skill mix. The manager in the legacy environment faces a particularly difficult staffing and training issue. The case study projects used "hot" technology, but because the FDD's existing technology was mainframe based, it did not tend to attract and retain people with expertise in new technologies. Those recruits who did have open systems experience generally were not experienced in either

application development or in the FDD's legacy systems and problem domain.

Training for technical managers. One problem with this technology transition was that the technical managers and senior technical people were reared in an older technology. The case studies show a tacit assumption that project managers would somehow "pick up" the open systems concepts sufficiently to competently plan and manage these projects. In fact, when project planners lack an understanding of the technology their team is using, they may not understand the real issues and cannot make good planning decisions. Open systems approaches bring significantly different problem-solving tools and techniques. Technical managers need training and hands-on experience. They need to know what they are up against when setting schedules and budgets.

Role of prototyping. Although useful for avoiding disaster, prototyping is not in itself a sufficient basis for project planning. A prototype does not confer organizational learning. Even a second-time use of a technology may not uncover all the possible pitfalls. Organizations have to assimilate information until they reach the point of "intuition."

Methodology. Methodology requirements oriented toward the routine design problem may actually impede learning, because they assume the problem-solving technology is already well understood. For example, the requirement that all unit designs be completed before any units are coded makes it impossible to feed lessons learned about the new environment into the design process. Although progress is harder to measure, iteration promotes learning the new environment. When introducing new technologies, a more appropriate approach may be to develop the system framework first and the application functionality later. The project can then be broken into numerous small builds and progress and expended effort assessed after each build. The development plan should be readjusted accordingly. To gain integration experience in the new environment, functionality should be slipped from early to later builds rather than delaying delivery of early builds.

Software metrics. Metrics are critical to understanding the new technology. However, measurement programs established for the old technology may not be adequate for the new. Predictors based on source lines of code may not be meaningful when using GUI builders, code generators, and COTS packages.

Conclusions and Recommendations

A technology transition plan. While it is not our purpose to develop a model for technology transition planning, our observations do suggest issues that a transition plan should address. Table 2 presents our suggestions from the perspective of a fairly large organization with a mature and stable, but dated, technology infrastructure. (The ordering of topics does not imply a procedural sequence.)

Climbing the hills of technology infusion. Adopting a new technology is like climbing a hill representing the cost of the transition. Few computing professionals and managers are expert at estimating the height of the hill and the rate of progress over it. Yet as Figure 3 shows, the increasing pace of change brings whole ranges of hills to climb. The FDD, having successfully applied the SEL process improvement concepts in a stable environment, was unprepared for the rapid pace of the transition to open systems. Perhaps the FDD, with its stable environment and funding, had become accustomed to investigating technologies at its own pace. In the current technological environment, however, we may not have the luxury to control which technology hills we will climb or when.

Can we learn to adopt technologies faster and more efficiently? A common element in these case studies is a failure to realize that technology transition alters the essence of the design problem. The literature on the design process distinguishes between *routine* design and *variant,* or

innovative, design. In routine design, both the problem domain and the problem-solving process are well understood, and the main issue is accommodating an established solution to project-specific needs. But in variant design, while the problem domain may still be well understood, the problem-solving process is not. Approaching the variant design problem as if it were just a more difficult instance of the routine problem, to which slightly adjusted models and procedures can be applied, leads to problems.

Improving management models for "emerging technology" projects. Variant design problems can be expected whenever new technologies are adopted. The software industry needs to systematize its knowledge of them. Project planners must understand when the organization is going through a transition that fundamentally changes the problem-solving process so they can approach it the right way. Of course, the problem is compounded by the fact that technology drives organizational structures; as industry

Table 2. Recommended Content of Transition Plan

Topic	*Comments*
Vision	What is the purpose of adopting the new technology? Will it support and improve the organization's business position? Examine assumptions regarding these issues to reveal aspects of the transition that otherwise might go unnoticed.
Industry assessment	How mature is the technology? Look at the hardware/software solutions being adopted by other organizations. Attend trade shows and conferences. Challenge the assumption that your organization is unique in its needs or functions. Be proactive in defining business direction in terms of new technologies.
Organizational assessment	Consider the impact of technology on the whole organization, not just your department. How will the technology change the roles of supporting organizations? How will the organization operate after adopting the technology?
Hardware/ software tradeoffs	Challenge the assumption that existing equipment must be retained for cost-effectiveness. The cost of software development and development environments may outweigh equipment cost.
Standards, conventions	Make sure all the required standards are identified and followed. Understand the difference between established standards and industry conventions.
Pilot projects	Define realistic goals for pilot projects; avoid developing products best left to industry (such as distributed operating systems). Concentrate on using new technology to bolster the organization's traditional strengths. Keep initial transition projects small.
Staffing	Staff transition tasks carefully. Experienced, motivated, capable people are essential to the transition project. Articulate the special skill needs to management, back the request with evidence, and be aggressive about resolving staffing problems and retaining team members.
Personal contact	Expedite personal contact across department boundaries. Establish mechanisms such as cross-department working groups, but avoid too much structure. Allow teams flexibility to discover what areas need focus and how to work together.
Training plan	Ensure that training is available; err on the side of too much. Project planners need exposure to the new technology at the time of planning; developers need it when assigned to the project. Support staff also need training.
Methodology	An iterative approach promotes learning. Use numerous small builds to gain integration experience in the new environment. Slip functionality rather than delay delivery. Challenge methodology requirements focused on the routine design problem.
Metrics	Challenge the assumptions of measurement programs established for the routine design problem. The new technology may demand different metrics.

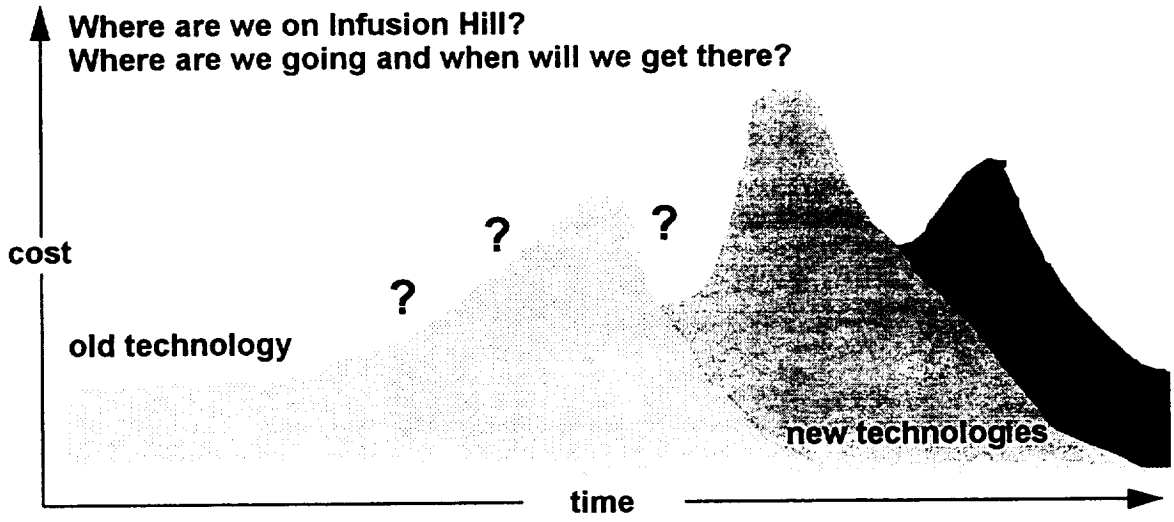


Figure 3. The Hills of Technology Infusion

retools, organizations discover possibilities that prompt them to reexamine their missions. Although this exploration can be guided only in broad outline, the need to steer projects through these uncharted waters remains.

New Directions for the FDD

Despite transition problems, the software developed by the projects we studied appears to be of good technical quality. The XTE systems were proved reliable in testing and are being reused by other projects for upcoming missions. New projects are using the UIX and the GSS in their designs. The FDD itself is embarking on full-scale conversion to a distributed system, porting or replacing up to 6 million lines of legacy software. A stable infrastructure for open systems is beginning to evolve within the FDD, improving prospects for success.

Moving a large organization from a mainframe legacy to a new environment of open systems is a complex technology transition problem. The transition involves much more than a simple switch of tools and techniques. Transitions that cause sudden shifts from routine to variant design problems are likely to become more common in the future. Our challenge is to apply organizational learning techniques in staying abreast of industry developments, and to effectively incorporate them in our experience base.

Acknowledgments

The authors gratefully acknowledge the help of Marvin Zelkowitz of the University of Maryland and Myrna Regardie of CSC in clarifying our concepts and consulting on the presentation. The original inspiration for this report was Zelkowitz's paper on the technology transfer process.

References

Landis, L., S. Waligora, F. E. McGarry, *Recommended Approach to Software Development (Revision 3)*, Software Engineering Laboratory, SEL-81-305, June 1992

Landis, L., F. E. McGarry, S. Waligora, et al., *Manager's Handbook for Software Development (Revision 1)*, Software Engineering Laboratory, SEL-84-101, November 1990

Zelkowitz, M. V., "Software Engineering Technology Transfer: Understanding the Process," *Proceedings of the Eighteenth Annual Software Engineering Workshop*, Software Engineering Laboratory, SEL-93-003, December 1993

Lessons Learned in Transitioning to an Open Systems Environment

Dillard Boland
Dave Green
Warren Steger

Purpose and Method

- Problem: Transition to a new technology requires investment before benefits are realized — how can we plan and manage efficient transitions in the midst of rapid industry evolution?
- Method: Study three projects in the GSFC Flight Dynamics Division (FDD) moving from a mainframe environment to "open systems" workstation technology
- Goal: Improve our understanding of technology transition and identify lessons learned

Background: The FDD Software Development Organization

- Through the SEL process, the FDD has achieved a track record of continuous improvement in reuse, error rates, and other software characteristics
- Stable development environment: IBM mainframe with FORTRAN, and DEC VAX with Ada
- Focused SEL experiments: OO, Ada, Cleanroom, IV&V, resources usage
- Computing environment held relatively constant while process and products evolved

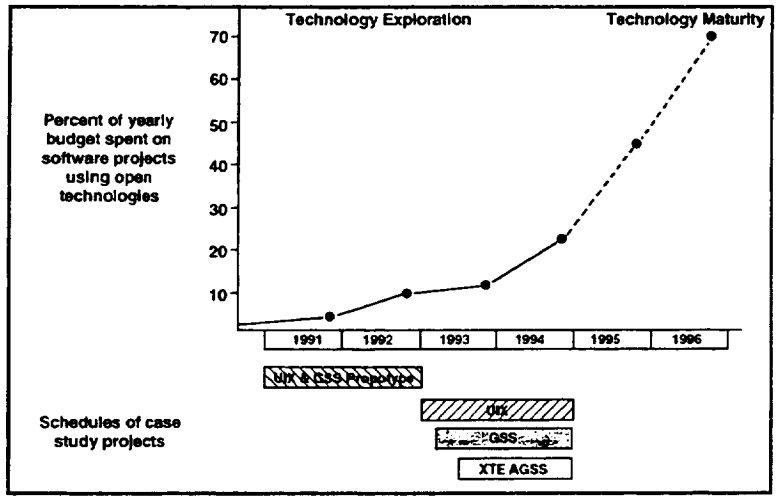
10022890 3

Background: Transition of the FDD to Open Systems

- FDD/SEL achievements were within the context of stable mainframe and VAX computing environments
- Now FDD is moving toward open systems
  - Workstation computing platforms, industry standards, and conventions
  - Use widely available COTS products; emphasize portability and interoperability
  - Goals are economic and technical: less vendor dominance, more competing solutions, "more bang for the buck"
- How will this dramatic change in computing environment affect our products and processes?

10022890 4

Background: Transition of the FDD to Open Systems

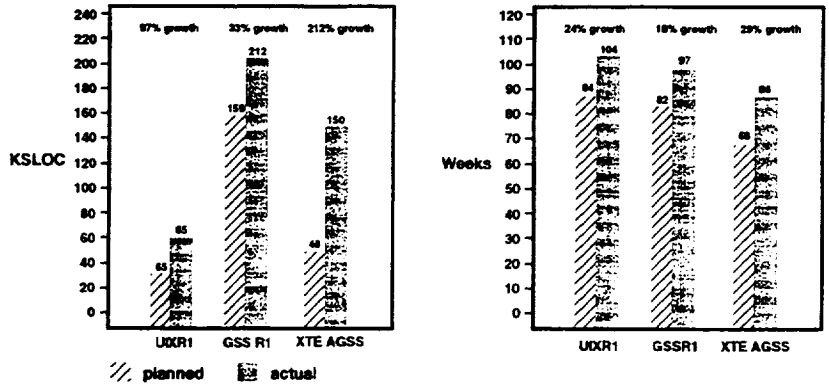


The Case Study Projects

Project	Platform	Size (SLOC)	Purpose
UIX	PC (SCO UNIX), HP	65,000 C	Multiapplication user interface system
GSS	DEC Alpha, HP	212,000 Ada	Multiapplication attitude support components
XTE AGSS (two subsystems)	HP	150,000 C and FORTRAN	Mission attitude ground support applications

Planned using SEL models based on local mainframe and VAX experience with adjustments for new technology

Case Study Projects: Comparision of Results



Compare to size growth in 20% to 40% range and schedule growth in 5% to 35% range on recent maintenance and VAX development projects

CSC

10022690 7

Case Study 1 — UIX

- We wanted to develop a common user interface and executive framework for interactive, distributed mission support systems
- We did the logical thing: up-front prototyping
  - Led to necessary architectural and conceptual changes
  - Not a good basis for project planning: final system is much larger and more complex than prototype indicated
- Lack of a preestablished system architecure ("middleware") proved to be a significant technical and organizational stumbling block
- The project was refocused on the user interface and extended: wait for industry middleware to evolve before attacking distributed executive

CSC

10022690 8

Case Study 2 — GSS

- We wanted a class library of flight dynamics capabilities from which we could build our systems; we prototyped it along with the UIX
- We wanted to transition Ada development to workstation environment, but have not been able to except for integration and test phases
- We discovered that matching development toolset capabilities available on DEC/Alpha/Open VMS is not yet cost effective on our target platforms
- Current plan is to phase in development tools as market forces drive the costs of Ada development systems down (this is already happening)

CSC

10022690 9

Case Study 3 — XTE AGSS

- We needed to integrate with client/server software developed by another group at GSFC, and to provide our first interactive X/Motif system for mission support (UIX was not ready)
- We assumed we could achieve our current norms: compressed development schedules and reusable software
- We severely underestimated the complexity and functionality required to meet these goals in a new environment
- We underestimated the difficulties of interfacing with other group's software (same "open" technologies, but environment differences such as COTS products at different version levels)
- Technology transfer facilitated by relocating developers to the other organization's site to infuse their technology, and by adopting highly iterative implementation approach

CSC

10022690 10

Lessons Learned: Organizational

- A coordinated organizational transition plan, with management commitment, is essential
- Changing technology can blur traditional roles, garble communications, and cause friction, because the "old ways" do not always adapt well to new technology
- The organization must find ways to cooperate and share lessons learned across departmental boundaries; technology transition is not the time for information hiding!

Lessons Learned: Technological

- Open systems and rapid industry change demand we cultivate market awareness to replace our "closed shop" outlook
- Open systems invite cooperation but do not ensure compatibility: stress coordination and communication
- Early training is important for both the technical managers and the frontline workers
- Problems previously solved in legacy environment (e.g., CM, reuse) often must be solved again in the new environment

Lessons Learned: Project Management

- Open systems require open minds: awareness of market trends, continuous organizational learning, structured feedback of lessons learned
- Use prototyping to avoid disaster but not as a basis for project planning
- Don't expect to achieve the goals of a technologically mature organization while you are transitioning
- We need a better management model for "emerging technology" projects

10022690 13

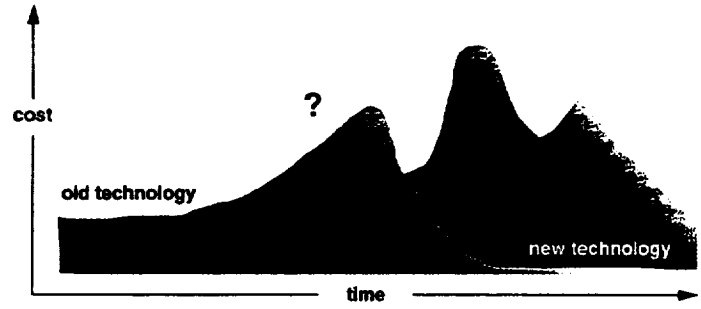
Conclusions and Recommendations

- We need scientific data about technology transitions: The industry needs to honestly appraise successes and failures and learn from them
- Our existing SEL models are not adequate for technology transitions: Upgrade them
- Open systems concepts and decreasing hardware costs force a systems (not just software) engineering approach
- Personal contact is the most effective means of information sharing on technology transition – need an institutional mechanism
- We must plan for continuously infusing technology and commit resources to that end

10022690 14

The Hills of Technology Infusion

- In a rapidly evolving industry and an open marketplace, we must learn better skills for evaluating and adopting new technologies



Where are we on Infusion Hill?
Where are we going and when will we get there?

New Directions for the FDD

- XTE AGSS subsystems are being reused for upcoming missions
- EOSTGSS project just completed PDR
  - Up-front emphasis on system engineering
  - Using UIX as part of infrastructure
- Flight Dynamics Distributed System
  - Port or replace the 6 million SLOC of our mainframe and VAX legacy
  - Will use GSS and UIX
  - Infrastructure is now coming into place