Software Engineering Technology Infusion within NASA

Marvin V. Zelkowitz
Institute for Advanced Computer Studies and Department of Computer Science
University of Maryland, College Park, Maryland 20742

Abstract

Technology transfer is of crucial concern to both government and industry today. In this paper, several software engineering technologies used within NASA are studied, and the mechanisms, schedules and efforts at transferring these technologies are investigated. The goals of this study are: (1) to understand the difference between technology transfer (the adoption of a new method by large segments of an industry) as an industry-wide phenomenon and the adoption of a new technology by an individual organization (called technology infusion), and (2) to see if software engineering technology transfer differs from other engineering disciplines. While there is great interest today in developing technology transfer models for industry, it is the technology infusion process that actually causes changes in the current state of the practice.

1 Introduction

The ability to move a new technology from a development laboratory into general use in industry is of increasing concern as today's economic climate constantly reduces the time available for companies to develop new products. This process, generally called *technology transfer*, is of crucial concern as industry today needs to remain economically competitive in the global marketplace.

Technology transfer is a difficult and slow process:

"One reason why there is so much interest in the diffusion of innovations is because getting a new idea adopted, even when it has obvious advantages, is often very difficult. There is a wide gap in many fields, between what is known and what is actually put into use. Many innovations require a lengthy period, often of some years, from the time when they become available to the time when they are widely adopted. Therefore, a common problem for many individuals and organizations is how to speed up the rate of diffusion of an innovation.[1]"

The software development community is aware of this problem and the need to diffuse new innovations, i.e., transfer effective technology towards improving the process of developing software. This is a major goal towards achieving improvements in productivity and reliability of the resulting products. Concepts like

[0]This paper will appear in the August, 1996 issue of *IEEE Transactions on Engineering Management.* Copyright ©1996, IEEE Inc.
[1]E. Rogers [12] Page 1

the Software Engineering Institute's Capability Maturity Model [10] have grown in importance as a means for modifying the software development process. The Experience Factory concept of the National Aeronautics and Space Administration (NASA) Goddard Space Flight Center (GSFC) Software Engineering Laboratory (SEL) [4] has shown the value of process improvement.

However, all process improvement involves changes. Some of these may be relatively minor alterations to the current way of doing business (e.g., replacing one compiler or editor by another), while others may require major changes that affect the entire development process (e.g., using Cleanroom software development and eliminating much of the unit testing phase). In order for an organization to continually improve its process, it must be aware of how it operates and what other technologies are available that may be of use.

While much has been written on the general concept of technology transfer within an industry, there is not much which describes the processes which an individual organization undergoes to adopt a new technology. This change generally goes under the name of *technology infusion*. We can describe technology transfer as technology infusion which diffuses across a broad segment of a given industry. In order to investigate these issues, several software engineering technologies that have been adopted by NASA for use on various development projects are studied. In particular, we are interested in the mechanisms that were used to accomplish the infusion of the technology, the effort involved in performing that infusion, and the time that it took to accomplish. This work is part of an indepth study from 1993 through 1994 of software engineering technology within NASA [16]; however, only those results that seem to be applicable to a more general technological audience are presented in this paper.

In Section 2 we discuss the general problems of technology transfer and in Section 3, we discuss technology infusion at NASA. We show that software engineering seems to follow a technology infusion process that differs from other engineering disciplines. In particular, the lack of a culture in software engineering to experiment and measure results makes validation of new technologies difficult. Also, the major "products" of software engineering are processes, which makes the discipline behave more like a scientific than an engineering activity. Established technology transfer models to not address this well. We present these findings in Section 4.

2 Technology Transfer

By *technology transfer* we mean the insertion of a new technology into most organizations that perform similar tasks. The insertion must be such that the new organizations regularly use that technology if the appropriate conditions on its use should arise in the future. The organization that adopts the new technology is said to *infuse* that technology. We will call the creator of that technology the *producer* and the organization that accepts and uses the new technology the *consumer* of that technology. The process of moving the technology out of the producing organization will be called *exporting* the technology.

2.1 Models of technology transfer

Technology transfer has typically been identified as an importation process. It often follows the *product life cycle* (PLC) identified by Rogers via the S-curve [12]. The first few customers are the "oddballs" or "eccentrics" of society, who adopt a new product. Following them are the "opinion leaders," who then give their approval to the product. Society then follows these opinion leaders, and product growth follows rapidly. During the mature stage, as the market saturates, growth levels off, giving the characteristic S-curve.

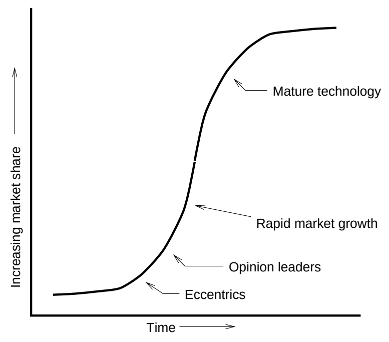

Figure 1: Product Life cycle S-curve growth cycle.

Gatekeepers. Technology transfer follows a similar process. One member of an organization, often called the *gatekeeper* [1], monitors technological developments, and chooses those that seem appropriate for inclusion in an organization; hence opens the "gate" to the new technology. Because this role is often informal, it may fall naturally to the most creative and technically astute individual in an organization. Since the gatekeeper is aware of technical developments outside of the organization, others in the group often look towards this person for guidance. This person often is known by the name "guru" or similar sounding monikers.

Transition models. However, the gatekeeper is not the only approach towards technology transfer. Other models of the process have been identified. In one study of 44 technology transfer efforts at one aerospace company, Berniker [5] identified four approaches towards technology transfer:

1. **People mover model.** In this approach, there is personal contact between the developer and the user of a technology. Typically there is some facilitator within the infusing organization that knows about the new technology and wishes to import it into the new organization (i.e., the gatekeeper mentioned above). This method was found to be the most prevalent and effective of all technology transfer methods.

   Nochur and Allen [9] investigated this model and discovered that it really consists of three separate subcategories, which we will call:

   1. **Spontaneous gatekeeper** role assumed by organization member.
   2. **Assigned gatekeeper** role imposed by management on some organization member.
   3. **Umbrella gatekeeper** role assumed by another organization to impose new technology on others.

2. **Communication model.** In this approach, the new technology has appeared in print and, as with the people mover model, some facilitator discovers the technology and wishes to infuse it into the new organization. The "print" mechanism may be internal documentation, conference reports or journal publications.

3. **On-the-shelf model.** This approach, relatively rare among the projects studied by Berniker, requires the new technology to be packaged so that non-experts can discover it and learn enough about it to begin the infusion process. It requires sufficient documentation so that others can easily pick it up and use it. Reading about the technology in a "parts catalog" is an example of this method.

4. **Vendor model.** This last method requires an organization to turn over the task to a vendor to sell them a new technology. It effectively turns the vendor into the agent of the People mover, Communication or On-the-shelf model.

The communication model and the on-the-shelf model can be viewed as marketplace models. Innovations (in the form of reports, papers, products) are placed in the marketplace, and users will discover what they need. However, these appear to be very imperfect mechanisms for technology transfer:

> “Papers are usually written for peers and for posterity, rather than for anything approaching mass communication. The dissemination of knowledge in scientific disciplines is imperfectly understood, but it appears to require only a very small number of diligent readers to start the human networking process that eventually socializes the information in an important paper. Many other papers never get socialized at all and pass unnoticed into the archival purgatory.[2]”

While Nochur and Allen found that the assigned gatekeeper was somewhat effective in importing new technology, he or she could not continue the transfer by moving it to other internal organizations (in essence acting like the umbrella gatekeeper). The third of these gatekeeper roles was most ineffective. Technology generally has to be wanted by the new organization and cannot be dictated by outsiders. However, the umbrella gatekeeper was really misclassified by Nochur and Allen. It is not a people mover strategy, since it is not a technology importation process, but instead represents an exportation of technology from one organization to another. We will call this new model the rule model:

5. **Rule model.** This method uses an outside organization to impose a new technology on the development organization, which then infuses it into its own development process.

There are many examples within the government sector of this last technology transfer model. The mandating of the Ada language by the Department of Defense’s Ada Joint Program Office for system development, the use of the Software Engineering Institute’s Capability Maturity Model to evaluate developer’s qualifications for a Department of Defense contract, the similar process of using international standard ISO 9000 in Europe, and the use of Federal Information Processing Standards (FIPS) by the National Institute of Standards and Technology (NIST) are all examples of technology transfer imposed by an outside agency.

Industry is not immune to the rule model either. Organizations have imposed tool standards on their subunits (e.g., imposing particular hardware and operating system products, CASE tools, or database systems), and organizations need to react to rules imposed by government contracting organizations, such as Request for Proposals that mandate a particular language, such as Ada. But in practice, however, successful examples of this imposed technology are rare. (It is not even clear if the above instances are successful examples of imposed technology transfer.) However, it is a model of technology transfer that has tremendous impact and we must not lose sight of it in our study.

Advocates. Fowler and Levine at the Software Engineering Institute have been investigating technology transition and have identified an extension to the gatekeeper model [6]. In their model, technology transition is a push–pull process:

$$\textbf{Producer} \Rightarrow \textbf{Advocate} \Rightarrow \textbf{Receptor} \Rightarrow \textbf{Consumer}$$

The produce of the technology needs an advocate to export the technology outside of the development organization, while the consumer organization must have receptors agreeable to importing the technology. In many instances, however, both the advocates and receptors are part of the consumer organization, and in practice, this reduces to a model very much like Allen’s gatekeeper.

[2]R. Lucky [7] Page 119

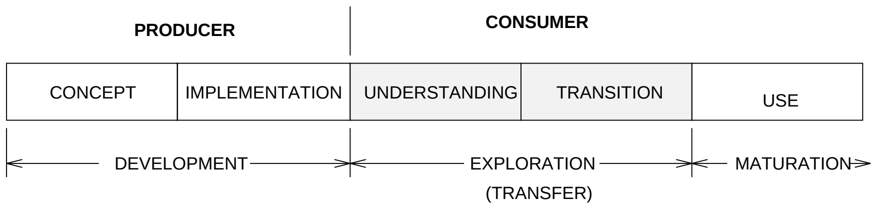

Figure 2: Technology Maturation Life Cycle

Successful technology transition. It should also be realized that the product life cycle S-curve is the result of the introduction of a successful product, plotted after its success. There is no guarantee that a new product will follow this curve and most new products do indeed fail. This is the problem of most forecasts about the predicted growth in a new technology [14]. Developers of a new technology are almost always too optimistic about the eventual success of that technology. One of the goals of our study is to understand the relationship between those early users of a technology and the methods used to transform it into a mature technology.

2.2 Technology maturation

In 1985, Redwine and Riddle [11] published the first comprehensive study of software engineering technology transfer, which they called *maturation*. Their goal was to understand the nature of technology maturation – what was the length of time required for a new concept to move from being a laboratory curiosity to general acceptance by industry. They defined maturation of a technology as a 70% usage level across an industry.

Technology maturation involves five stages, two by the producer of the technology and three by consumers of that technology (Figure 2):

1. The original *concept* for the technology appears as a published paper or initial prototype implementation.
2. The *implementation* of the technology involves the further development of the concept by the originator or successor organization until a stable useful version is created.
3. In the *understanding* stage, other organizations experiment, tailor, expand, modify, and try to use the technology.
4. In the later *transition* stage, use of the technology is further modified and expands across the industry.
5. The final *maturation* stage is reached when 70% of the industry uses the technology.

In their study, they looked at 17 software development technologies from the 1960s through the early 1980s (e.g., UNIX, spreadsheets, object-oriented design, etc.). Their results, most related to this current study, were:

- They were unable to clearly define “maturation” for most technologies, but were able to make reasonable estimates as to the length of time needed for new technologies to become widely available.
- Technologies required an average of 17 years to pass from an initial concept to a mature product.

	NASA Consumers	External Consumers

NASA Producers	Reuse(Kaptur), ∗AI(CLIPS)
	∗GUI(TAE), CASE tools
	Measurement(SME, GQM)
		Reuse(Kaptur), ∗AI(CLIPS)
		∗GUI(TAE)
		Measurement(SME, GQM)

External Producers	Rate monotonic scheduling, CASE tools Cost models, ∗Formal Inspections ∗Object oriented technology ∗Ada, C, C++, ∗Cleanroom	*Not relevant to this study*

∗ – technologies discussed in more detail

Table 1: Transferred technologies at NASA

- Technologies, once developed, required an average of 7.5 years to become widely available (i. e., the Exploration stage of Figure 2).

We view this current paper as the inverse of the Redwine and Riddle study. For each technology, how long did it take to infuse that technology into a given organization? That is, what was the Exploration stage within a given organization? The 7.5 years needed to penetrate an industry that was specified by the Redwine-Riddle study would be an upper bound, and we know that the lower bound is more than the gatekeeper simply declaring the new technology to be good.

3 Technology Infusion

In order to understand technology transfer within NASA, about 15 software engineering experts at several NASA center were interviewed to determine which software engineering techniques were being used effectively in the agency. To keep the scope of this problem manageable, the following two restrictions were imposed:

1. The technology had to clearly be software engineering. For example, successfully transferred programs, such as the widely-used modelling system NASTRAN available through the NASA Software Repository (COSMIC), were not included.
2. The technology had to have a major impact on several groups within NASA. With more than 4,000 software professionals affiliated with GSFC alone (including government and contractors), almost every software product has probably been used somewhere in the agency. While this was somewhat subjective, a list of transferred technologies was developed (Table 1). Technologies developed outside of NASA and not used within NASA were outside of the scope of this study, hence the *“Not relevant to this study”* section of the table.

3.1 Examples of Technology Infusion

In this section we first discuss three technologies that were successfully infused into the state of the practice at GSFC (Ada, Object oriented technology, and Cleanroom) in greater detail. The details of transferring those technologies are summarized by Figure 3. Each represents the understanding and transition stages as

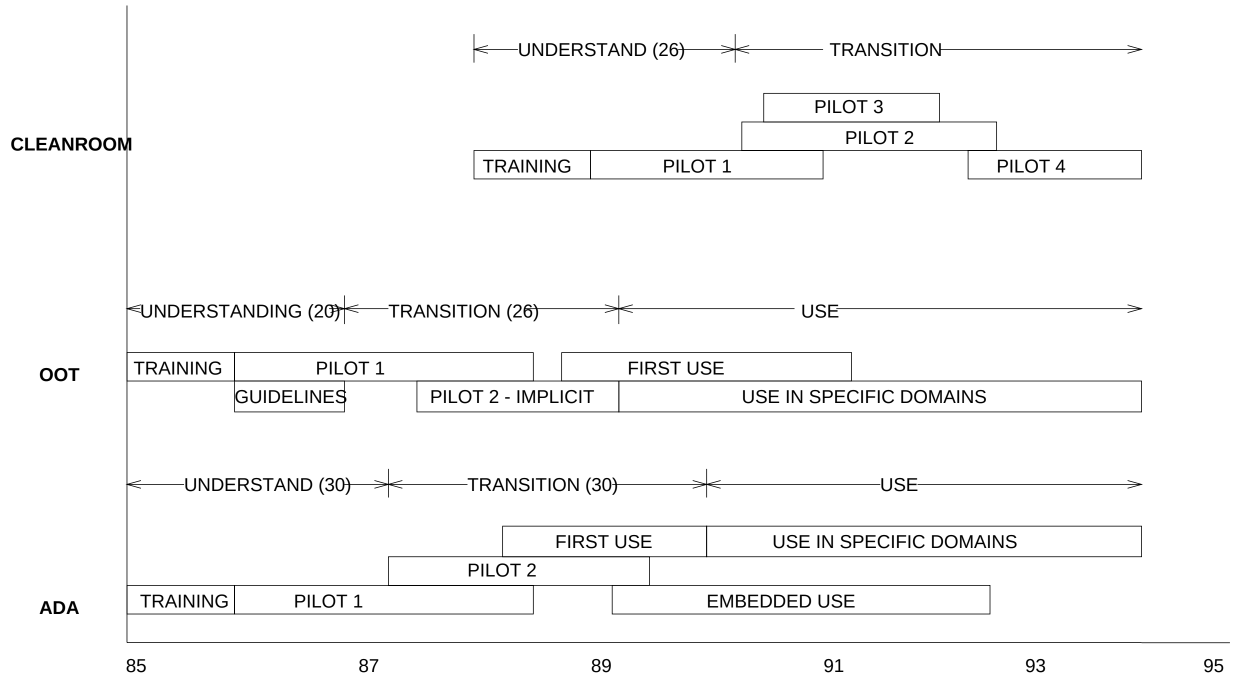

Figure 3: SEL technology transfer experience

NASA plays the role of consumer organization trying to adopt these new technologies. These technologies were studied by the Software Engineering Laboratory (SEL) at GSFC. In addition, we also include several technologies transferred by groups other than the SEL.

The NASA/GSFC SEL has been a major source of technology infusion at Goddard Space Flight Center. The SEL was organized in 1976 to study flight dynamics software, and since that time it has had a significant impact on software development activities within the Flight Dynamics Branch. Most of these technologies (e.g., measurement, resource estimation, testing, process improvement) have been reported elsewhere [2].

As a brief overview of SEL operations, the SEL has collected and archived data on over 125 software development projects. The data are also used to build typical project profiles against which ongoing projects can be compared and evaluated. The SEL provides managers in this environment with tools for monitoring and assessing project status. Typically there are 6 to 10 projects simultaneously in progress in the flight dynamics environment. Each project is considered an experiment within the SEL, and the goal is to extract detailed information to understand the process better and to provide guidance to future projects.

Projects range in size from approximately 10K lines of source code to 300K to 500K at the high end. Projects involve from 6 to 15 programmers and typically take from 12 to 24 months to complete. All software was originally written in FORTRAN, but Ada was introduced in the mid-1980s (see below), and there is now an increase in C and C++ programming.

Use of Ada

Ada is a language that was developed by the U.S. Department of Defense from 1976 until 1983 as a common language on which to build complex embedded applications. It is a general purpose programming language adaptable to any computing environment, although some claim that the language is too complex to use effectively. During the 1980s, as Ada compilers started to appear commercially, many organizations evaluated the language as a solution for their existing programming needs.

Technology Transfer Model. Use of Ada on flight dynamics projects was first considered in 1985. Because of Department of Defense interest in the language and because of NASA Johnson Space Center's decision to use Ada for Space Station software, the SEL desired to look at its applicability for other NASA applications. The initial stimulus for this activity, then, could be a mixture of the communication model (i.e., papers were written about Ada), on-the-shelf model (i.e., Ada products were being sold) and to some extent, the rule model (i.e., since Johnson Space Center adopted Ada, there was some pressure to do the same elsewhere within NASA).

Understanding phase of Technology Transfer. A training class and sample program was the first Ada activity. However. to truly evaluate the appropriateness of Ada within the SEL environment, a parallel development of an Ada (GRODY) and FORTRAN (GROSS) simulator was undertaken. GROSS, as the operational product, had higher priority and was developed on time. GRODY, as an experiment to learn Ada, had a much longer development cycle. In addition, since GRODY was known by all to be an experiment, the development team was not as careful in its design. However, the experiences of the GRODY team with the typical set of requirements NASA used for such products led to a greater interest in applying object-oriented technology instead as a model for future NASA requirements and design specifications. Although the development of this simulator continued until early 1988, by early 1987 it was decided that the initial project was sufficiently successful to continue the investigation of Ada on other flight dynamics problems. Elapsed time since start of Ada activity was 30 months.

Experiences at NASA Langley Research Center were similar to those of the SEL, but had a different conclusion. Understanding Ada began under the advocacy of one individual. A project was developed in both FORTRAN and Ada. Although the Ada project was deemed more successful than the FORTRAN version, the difference was not deemed great enough to enforce Ada on all projects.

Transition phase of technology Transfer. Because of the poor performance on the GRODY simulator and the problems with developing Ada requirements, the SEL undertook a second Ada pilot project (GOADA) as an experiment. However, there was sufficient confidence in Ada by this time to make GOADA an operational product, thus schedules and performance were more critical than with the previous GRODY experiment. In this case, the resulting product was comparable to performance of previous FORTRAN simulators. Between 1988 and 1990, four other simulators (one dynamic and three telemetry) were built successfully. In addition, one embedded application was developed beginning in 1989 and was not as successful due to the poor quality of the compilers for embedded applications that were available. By early 1990, Ada became the language of choice for simulators in the Flight Dynamics Division. Transition time was another 30 months.

Comments on Technology Transfer. Total transfer time for Ada was approximately 60 months. Ada is now the language of choice for simulator projects using Ada on DEC VAX computers. Although Ada code costs more, line by line, than FORTRAN code (about 20%), the higher levels of reuse result in lower overall delivery costs for such projects.

Ada was also proposed as the implementation language for large mission ground support systems, but this was never tested. The inhibitors in this case were outside of the features of the Ada language, itself.

The operational systems at GSFC are IBM mainframe compatible, and no effective Ada compiler existed for this environment during the three times Ada was evaluated during the late 1980s. All of the successful simulator projects were implemented on DEC VAX computers, which did have an effective Ada system.

Presently, Ada is used on approximately 15% of the SEL's software. Eleven operational Ada projects have been completed to date. Another report gives a more complete analysis of the SEL's experiences with Ada [3].

One difficulty in evaluating the effects of Ada on software development within the SEL is due to side effects which may occur. The following study of object-oriented technology grew directly out of the early Ada experiences.

Object Oriented Technology

In traditional software design of the 1960s and 1970s, a program consisted of a series of functions grouped into a set of subprograms. Software design consisted of developing these subprograms via functional decomposition of the overall program requirements into smaller functional units.

In the 1980s, object-oriented design became an alternative to the earlier functional decomposition. A program now consists of a set of data objects and a set of operations that apply to these data objects. Software design now consists of developing these complex data objects (called data abstractions) and building a set of functions that manipulate the objects. Since design is more localized to the operations applying to a single data object, proponents of the method claim that the resulting product is more manageable, simpler, and more reliable.

Technology Transfer Model. Use of object-oriented technology (OOT) in the SEL was investigated at the same time as Ada was considered, although was not a primary goal of the original decision to study Ada. In modifying the "standard" requirements of the FORTRAN-implemented GROSS simulator for the GRODY experiment (the simulator to be written in Ada), it became apparent that the standard GSFC requirements document was oriented towards a FORTRAN functional decomposition and the use of these requirements on an Ada project would be very inefficient. We view this as an example of the communications model of technology transfer. The existing requirements were deemed inadequate, and papers in the literature were collected on an alternative technology (i.e., object-orientation) which might be applicable in this domain.

Understanding phase of Technology Transfer. Object orientation seems more natural an approach with the use of Ada packages and generic functions. Therefore the requirements for GRODY were rewritten to use a more object-oriented approach. Following this, an OOT guidebook for GSFC was developed (GOOD - General Object Oriented Software Development [13]) for use on future projects.

Elapsed time for these activities took from early 1985 until August, 1986, or a total of 20 months. Expenses for understanding this technology were high since this activity was wrapped up in the Ada evaluation which required parallel system development of GRODY with the FORTRAN equivalent GROSS.

Transition phase of technology Transfer. On a second project (UARSAGSS), object-oriented design was used implicitly. This was a FORTRAN ground support system, and experiences gained from the earlier GRODY effort allowed the programmers to better understand the design and use OOT. By the end of this project, it was sufficiently clear that OOT was an effective technique in some domains. Transition time was on the order of 26 months.

Comments on Technology Transfer. Total transfer time in this case was only 46 months. Although almost four years, this was relatively short since it did not require major changes in system development. The same set of tools could be used; object-oriented technology was mostly a change in approach towards system building that could be used with any underlying implementation language. Although initially considered as an Ada technique, the same methods would map easily to a FORTRAN development model. Since it fit within the usual development paradigm, tailoring the method and inserting it into the usual NASA development process was relatively easy.

It should also be mentioned, that although object-oriented technology was originally studied within the Ada domain, it has had a profound effect on productivity on FORTRAN projects as well. This provides an additional reason why adoption of Ada as the development language has not provided significantly better productivity within the SEL. Although overall productivity using Ada has greatly improved over FORTRAN productivity of the mid-1980s, FORTRAN productivity has also improved dramatically.

For example, productivity measurements in statements produced per hour of effort on four recent Ada and four recent FORTRAN projects show that FORTRAN is still easier to write and is easier to reuse than Ada code [15]:

Statements	Ada Stmts/hr	FORTRAN Stmts/hr
New statements	1.1	1.2
Reused statements	5.0	5.5

While not statistically significant, the data does indicate that both languages seem comparable.

Reuse has proven to provide the largest boost in productivity with both Ada and FORTRAN due to the effects of object-oriented technology:

Language	1988-90 % Reuse	1990-1994 % Reuse
Ada	4-17 (3 projects)	64-88 (4 projects)
FORTRAN	4-12 (7 projects)	75-90 (4 projects)

Although not every project achieves such high levels of reuse, the trend is certainly upwards. This shows the seredipity nature of technology transfer. You generally cannot dictate exactly what you want to change, and change may occur from unexpected sources.

Cleanroom

Cleanroom is a software development method that relies on *a priori* verification of a software product rather than the usual *a posteriori* testing of software for validation. All software designs are verified via both formal and informal proofs of correctness rather than relying on later testing to find errors. The claim is made that errors are easier to find within a design document before that design has been codified into a large source program. The method was developed by H. Mills while at IBM during the late 1970s.

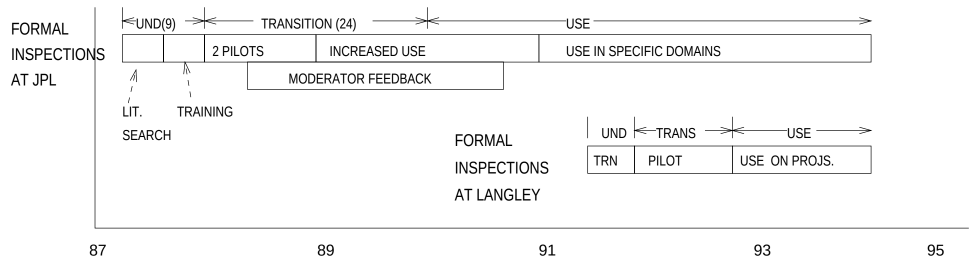

Figure 4: Formal inspections technology transfer experience

The immediate effect of this method is that a project spends more time and effort in the design phase as designs are verified than with more traditional development methods. The payoff is that less testing will be needed later, with a decrease in total project costs and an increase in product reliability.

Technology Transfer Model. Cleanroom was studied in the SEL via the people-mover model, at the instigation of V. Basili of the University of Maryland, who is also one of the SEL Directors. Previously, Basili had studied cleanroom within a student environment at the University of Maryland.

Understanding phase of Technology Transfer. To understand Cleanroom, a series of training courses was given in 1988 by H. Mills, original developer of the method. A pilot project was undertaken and proved to be very successful. All participants were converts to the method, even though several had reservations about it before they began. Time to understand the method (training until the start of the second Cleanroom project) was 26 months.

Transition phase of technology Transfer. Two follow-on Cleanroom projects were undertaken. A smaller in-house development was very successful, but a larger contracted project was not so successful. It was not as apparent that problems on the larger project were due to scaling up of Cleanroom to larger tasks or to a lack of training and motivation of the development team on this project. For this reason, a fourth larger Cleanroom project was undertaken. The fourth pilot was completed in early 1995 (after this technology transfer study officially ended), and the resulting system was considered to be extremely reliable.

Comments on Technology Transfer. Cleanroom technology appears to be an effective technology on smaller projects, but may lose some of its effectiveness on NASA projects involving more than 160K lines of code. Training and motivation of the staff were considered crucial for its success. Understanding time was 26 months and transition time was about 46 months. With the completion of the fourth pilot project, the Flight Dynamics Branch is now evaluating Cleanroom's role in future developments.

Formal Inspections

In addition to the techniques evaluated by the SEL, technology infusion was studied elsewhere within NASA. One such technique whose use is growing within NASA is the use of formal inspections (Figure 4).

A formal inspection is a verification method that has aspects similar to the cleanroom method mentioned previously. In a formal inspection, one software artifact (e.g., a module design, source code for a module, test data) is identified for inspection. The author of that artifact must present the design of that artifact to a meeting of fellow workers, who were previously given the artifact to study. The process of discussing the artifact for a given period of time (generally not more than 2 hours) has proven to be very effective in finding errors in the object of study.

Infusion of formal inspections We first studied the infusion of formal inspections within the Jet propulsion Laboratory.

Technology Transfer Model. Initial work at the Jet Propulsion Laboratory (JPL) on formal inspections began in February, 1987 in response to a need for higher quality software. J. Kelly of JPL was the initial gatekeeper who proposed studying this technology.

Understanding phase of Technology Transfer. After studying the literature, a decision was made in mid-1987 to tailor inspections for JPL use. A course was developed for use at JPL and the initial advocates for inspections sought other JPL managers (the receptors) who would use and benefit from the technology.

Transition phase of Technology Transfer. Approximately 2 or 3 pilot projects were started in 1988 using inspections. Bimonthly moderator meetings were held to spread the advocacy from the initial developers to other managers at JPL so that there would be others who “own” the process should the developers leave the organization. From 1989 through 1991 additional projects used the technology to help institutionalize the process. By late 1989 it was rapidly becoming a standard technology at JPL.

Comments on Technology Transfer. The elapsed time for developing the method was about 33 months and involved about 3 staff years of effort. Meetings and telephone contact with M. Fagan of IBM, developer of the technique, helped the JPL staff understand the process. It has been successfully transferred to JPL and between its initial use in February, 1987 through 1990, over 200 inspections were carried out. The use of moderator meetings greatly aided the “people mover” model as other managers became advocates of the technology to help infuse it at JPL.

Transfer of Formal Inspections Based upon experience at JPL, NASA tried to move the technology to other NASA centers.

Technology Transfer Model. The transfer of formal inspections demonstrates the difficulty of the assigned gatekeeper model described earlier. The need for this gatekeeper was quite apparent early in this process. For example, although meetings were held with management at Kennedy Space Center, no receptor for the technology was found and the infusion process never took hold (i.e., failure of the assigned gatekeeper role).

To avoid this problem at other centers, the transfer process first tried to identify the appropriate receptor who would promote the infusion process. At Langley Research Center, a course was offered and an advocate was identified who helped with the infusion. In May of 1991 the initial Formal Inspection course was offered, and by Fall, 1991 a pilot project was started. In August of 1992, inspections were declared a standard part of all developments. This process took only 16 months, because of the previous experiences

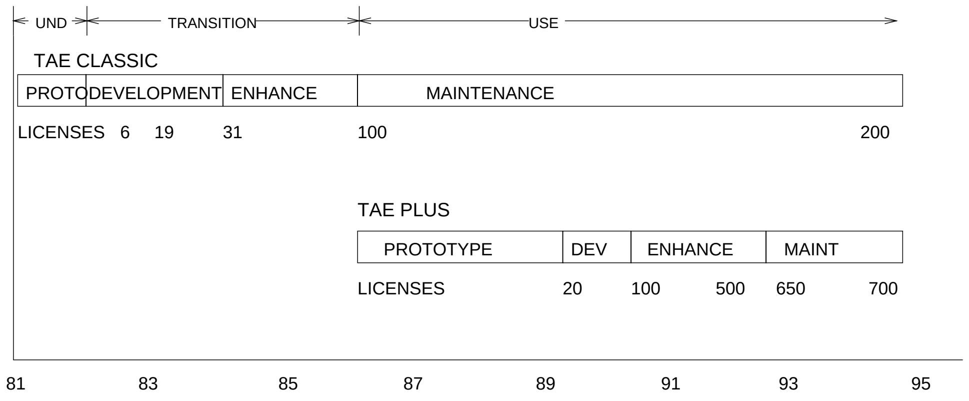

Figure 5: TAE development

at JPL. About 12 staff months of effort were required, but most of this effort was in "unpaid overtime." No NASA support was available for developing the technology. Once installed at Langley, it has been transferred to several contractors working at Langley.

Comments on Technology Transfer. Formal inspections were successfully transferred at JPL and Langley. Total time for transferring at both centers were 33 months and 16 months. These were relatively short since formal inspections cover only a relatively narrow and precise process in software development and can be inserted relatively easily into almost any mature development process. On the other hand, the process was not successful at other centers where no advocate was found.

TAE

TAE (Transportable Application Environment) probably represents NASA's major success in exporting software products outside of NASA. TAE is a graphical user interface useful as a front-end for other software products. It was developed between 1981 and 1990 at GSFC.

Initially TAE was a simple character-oriented interface between the user at a terminal and an application. With the development of the X Window System by the MIT X Consortium in the 1980s, TAE was rewritten to use the X Windows graphical interface.

TAE development as a producer. Originally designed in 1981 (See Figure 5) to support ASCII terminals, this product was renamed TAE Classic when an enhanced X-Windows and Motif compatible version (TAE Plus) was developed in 1986. It was originally distributed through the NASA software library COSMIC. Over 900 licenses have been obtained for the product. TAE represents one of the few commercial successes in software technology transfer for NASA. For the past two years non-NASA users have obtained TAE commercially via Century Computing, Inc. It is one of the few software engineering technologies that has been transferred via the On-the-shelf model of technology transfer, although the communication model was the primary vehicle for "getting the word out."

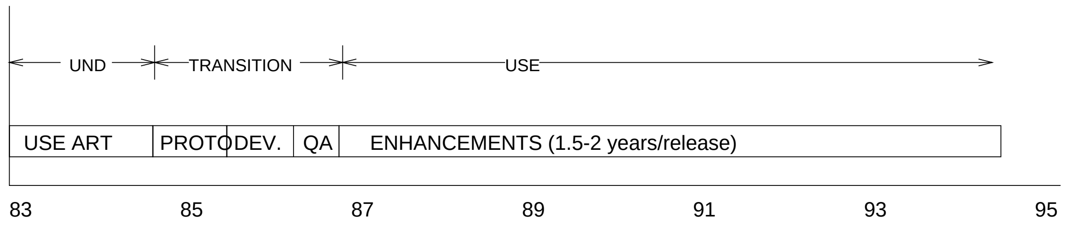

Figure 6: CLIPS development

Comments on Technology Transfer. The TAE developers support its users via: (1) A help desk in lieu of training and consulting; (2) Over 1500 pages of reference documentation; (3) Newsletters giving advice on how to use the system; and (4) TAE user conferences held approximately every 18 months.

Exporting TAE encountered problems quite similar to those encountered in other technology transfer environments:

1. The cultural bias against anything new made other organizations reluctant to try this product. The "Not Invented Here" syndrome is strong in software development organizations. Discovering potential gatekeepers from the outside is difficult to accomplish.
2. The mechanisms to make potential users aware of the benefits of TAE were inadequate.
3. Mandating use of an immature technology (the Rule model) may be counterproductive if it causes an effective technique to be discarded before it is ready to be used. TAE Classic was not quite ready in the mid-1980s and its mandated use on some GSFC projects led to less than optimum performance and a lack of advocacy among some development groups. It took the development of TAE Plus to get the product to its full potential. Therefore, TAE use within GSFC generally lags behind use of TAE at other NASA centers.
4. There was a reluctance to accept a government-developed product as legitimate and of quality design. However, two-thirds of all TAE licenses are at non-NASA sites.

TAE development as a consumer. TAE use spread through NASA. TAE usage within NASA was similar to infusion of any other technology. Reports about TAE were read, and since the group was already committed to an X-Windows and Motif interface, use of TAE Plus seemed appropriate. Total understanding and transition time was approximately 42 months before it became operational.

CLIPS Expert System

The C Language Integrated Production System (CLIPS) was developed at NASA/JSC (Johnson Space Center) as an expert system to solve problems that JSC was having with previous products [8]. Using a rule-based syntax similar to ART (Automated Resoning Tool), CLIPS was implemented in C to be efficient and portable.

CLIPS development began around 1985 as a prototype proof-of-concept batch implementation that would use some of the syntax from the earlier ART product (Figure 6). The initial goal was simply a training exercise to make JSC an "intelligent customer" for similar products. However, after completing the prototype, in May, 1985 it was decided to build a product for internal use for the HP workstation. This product was completed around May, 1986. Only after completion did the developers consider the possibility that others may want to use it, and a Quality Assurance phase (QA) was instituted to eliminate remaining errors and make the system portable and useful by others. The software was submitted to COSMIC in August, 1986.

CLIPS had a relatively short 15 month development cycle. Since its rules were based upon the previous ART expert system, there was little need for design trade-off studies. The basic rule-based algorithms were coded in C for efficiency and portability. Since its initial release, versions of CLIPS have been ported to DEC VAXs, mainframes, UNIX workstations, Macintoshes and PCs; its portability enhanced by being coded in C.

Comments on Technology Transfer. CLIPS was transferred outside of NASA/JSC by the following mechanisms:

- **NASA Technology Transfer facilities.** CLIPS was submitted to COSMIC for general distribution. Since 1986, CLIPS has over 5,000 users in government, academia and industry. COSMIC was viewed as just a vehicle for distributing the source program, but not in how to spread the word about the product. CLIPS represents a second example, like TAE, of an “on-the-shelf” product being exported outside of NASA.
- **Conferences.** Papers on CLIPS appeared at many professional conferences. Papers started to appear that referenced the use of CLIPS to solve various application problems, thus increasing interest in the product.
- **Technology transfer model.** The people mover model via “word of mouth” was the most effective means for distributing information about CLIPS. The JSC development group distributed copies to other NASA centers, where NASA project managers became advocates who wanted their contractors to use CLIPS for relevant applications. Thereafter, non-space-related divisions of these contractors learned about CLIPS and started to use it on non-NASA applications. Discussions over the Internet spread the word about the product. Being written in C, its efficiency led to rapid growth in its use.

Both CLIPS developers and some users claimed that one of the reasons for its spread was that it was the only expert system shell generally available in source program format. This enabled users to tailor the product for their own local environment and made understanding aspects of the system easier.

3.2 Software Engineering Processes

Software processes are generally not embodied in a product. Transfer of processes turns out to be very difficult. For example, the most successfully transferred products found in this study of NASA were TAE and CLIPS, both executable products. Transfer of processes (e.g., Cleanroom, object-oriented technology, formal methods) was found to be much more limited in this study.

SEL measurement model. To address this problem, the SEL has developed the quality improvement paradigm (QIP) for the diffusion of new innovation within an organization. Figure 7 briefly summarizes this process as an evolution of SEL activities since the inception of the SEL in 1976. The process improvement model involves understanding a technology, assessing its applicability within a new environment, and packaging it for routine use.

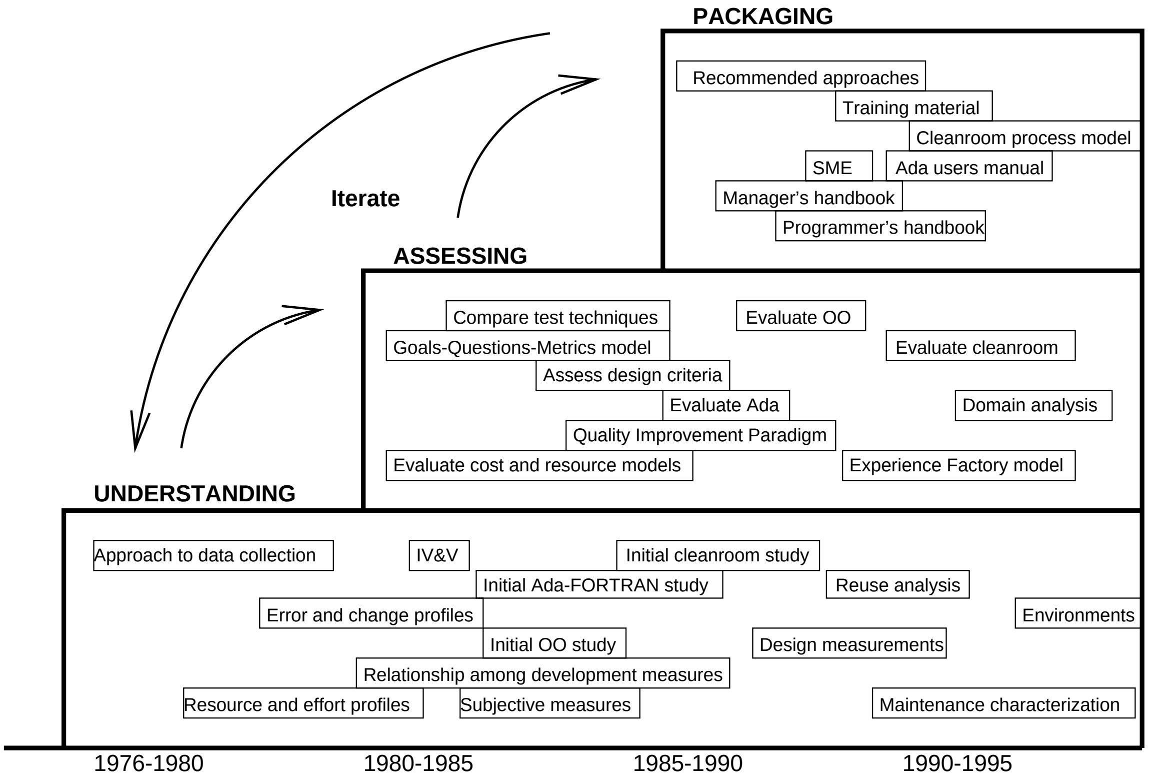

Figure 7: Summary of SEL Studies

The *understanding* step baselines process and product characteristics (e.g., cost, reliability, software size, reuse levels, error classes). The *assessment* step incorporates potential improvements into development projects that are to be evaluated. Quantified SEL experiences (e.g., most significant causes of errors) and clearly predefined goals for the software (e.g., decrease error rates) drive the selection of candidate process changes. After the changes are selected, training is provided and experiment plans are produced. Then the processes are applied to one or more production projects from which detailed measurements are taken. The new process is assessed by comparing these measures with the continually evolving baseline. As a result of the analysis, processes are adopted, discarded, or tailored for ensuing efforts depending on the observed impacts. The *packaging* step infuses identified improvements into the standard SEL process. This includes updating and tailoring standards, handbooks, training materials, and development support tools.

Process improvement applies to both the individual project or experiment level (observing 2 or 3 projects) as well as to the overall organization level (observing trends of numerous projects over many years). In the early years, the SEL emphasized building a clear understanding of the process and products within the environment. This led to the development of models, relations, and general characteristics of the SEL environment. Most of the experiments (process changes) consisted of the study of specific, focused techniques (e.g., program design notations, structure charts, reading techniques), but the major enhancement was the infusion of measurement, process improvement concepts, and the realization of the significance of the process as part of the software culture.

Measurement and experimentation elsewhere. One of the original goals of this study was to also collect data on the costs of technology infusion. However, outside of the SEL, very little data was collected on which to make any conclusions. While total project costs are usually collected by development organizations, there is usually no real breakdown into the various activities required for exploring a new technology. Any results here would be mostly unreliable guesses.

4 Conclusions

In studying technology transfer we believe that we have identified the five models of technology transfer at work within NASA, the four models (people-mover, communications, on-the-shelf, and vendor) of Berniker as well as our fifth rule model.

Infusion of technology generally took from two to four years to accomplish. An initial study, training course, or prototype development took from six months to a year. Once the technology was deemed appropriate, two to four pilot studies were undertaken as the method was tailored to the local environment. After several of these studies, the method (either implicitly or explicitly) became state-of-the-practice within that organization.

One limitation to this study is that NASA, as a government agency, is not driven by the same set of market forces as in a profit-making organization. While pressure to downsize, lower costs, increase reliability, and shorten the development cycle are as true for NASA as they are for most other organizations, the demands to do so are driven more by management (and Congressional) demands and less by loss of market share and lack of profit.

From this study, we can make several observations about technology transfer mechanisms. While these conclusions apply solely to NASA, we believe that the results are fairly general and should apply to other comparable technological organizations.

4.1 Differences between software engineering and other technologies

This study of NASA exhibited several attributes of software engineering which differ from other engineering disciplines. As such, software engineering technology infusion follows a process that differs somewhat from other technology transfer models:

1. **Infusion mechanisms do not address software engineering technologies well.** Software has a crucial difference that separates programming from other engineering disciplines. Just as software differs from other products in that it really does not “age,” “decay,” or “cease to function” (and hence engineering concepts like mean-time-to-failure are not truly relevant with software), the development of software is much more process-driven and less product-oriented. Software engineering is currently very dependent upon programmer expertise and less upon implemented technology than with many other forms of engineering, as we describe below.

   There are few integrated systems that effectively aid the software engineer to build complex systems. Most software engineering technology are processes, a set of rules to be followed in the development of systems. How to package and transfer processes as a corporate asset must be handled better. For example, within the GSFC Software Engineering Laboratory, the following processes have been studied over the past few years:

   (a) Object Oriented Technology,

   (b) Goals/Questions/Metrics paradigm of software development,

   (c) The Experience Factory model of development, and
   (d) Cleanroom software development.

   (Only (a) and (d) were described in this paper. See [2] for other SEL studies.)

   None of these processes is embodied in a product. One cannot buy a “Cleanroom” program. Instead one buys some books, a training course and some guidance on using the technique. Although NASA does not explicitly address the packaging of such processes as assets to be transferred, NASA in not unique in this regard. Much of industry does not understand the unique role that processes play in software development compared to most engineering processes. It is imperative for industry to understand this distinction and to address the transfer of processes as well as products.

   This observation makes software engineering appear more like Allen’s view of science rather than technology [1]. In science, the desired output is “verbally encoded information” in the form of published papers, whereas in technology, the desired result is “physically encoded information” in the form of hardware products. “Verbally encoded information” in the form of product documentation or published papers on the technology is not viewed as important as the product itself. Current technology transfer organizations are attuned to the technology model of technology transfer and have not adapted (or needed to adapt) to the *science* model of software engineering technology.

2. **Quantitative data is crucial for understanding software development processes.** Like other engineering disciplines, without quantitative results, it is impossible to fully understand what is happening and what are the effects of instituted changes. However, outside of the SEL at NASA/GSFC, few organizations (both inside NASA and outside) collect effective data on their development practices. In this study, while we were able to track the time to accomplish several instances of technology infusion, any details about the costs of such technology transfer and on their bottom-line effects on project costs, reliability, or schedules were mostly educated guesses. Measurement and experimentation are not part of the software development culture.

3. **Technology infusion is not free.** Organizations already understand that without the appropriate advocate already in place in some infusing organization, the ability to export a new technology to that organization depends upon a significant marketing effort to make the new organization aware of the benefits of the new technology. For example, NASA already spends considerable funds running a Technology Utilization Office, COSMIC, and other components of its technology transfer program. On the other hand, technology infusion is rarely supported.

   Unlike other engineering disciplines, rarely are IRAD (Internal Research And Development) funds available for developing new software technology. New technology is often procured out of existing project funds and not capitalized over the life of the product. This may be related to the previous comment that software engineering behaves more like a science, and IRAD funds typically are used for technology developments.

   Organizations generally keep track of total project costs, but separating them into individual tasks (pilot project development, tool use training, new method experimentation, etc.) is generally not done. The cost of innovation as a part of an operational project becomes a severe inhibitor to using new innovations. Costs of such innovation have to be borne by project development budgets, greatly increasing the risk of a cost overrun. Conservative fiscal planning makes innovation an even higher risk than is necessary.

4.2 Similarities between software engineering and other technologies

To a great extent, we reconfirmed within the software engineering domain many of the issues concerning technology transfer found by others:

1. **Most software professionals are resistant to change.** One manager at NASA referred to the "cultural block" to new technology by those who were used to mainframe computers, while another manager was more pragmatic in stating that his staff needed to see the technology demonstrated in a meaningful way in their own environment before they would consider accepting the technology.

   "Activities in this [technology transition] life cycle must attend to the culture of the organization in which the technology is being implemented" [6] is an aspect of technology infusion that is often ignored. The motto of "one size fits all" should not apply in this domain.

2. **Technology transfer is more than simply understanding the new technology.** Technology transfer takes time. Understanding the technology has been shown to take upwards of 2.5 years, and it usually involves multiple instances of training and pilot projects. The transition time when the organization is exploring, tailoring and modifying the technology for its own use often takes more than the understanding time, with a total transfer time on the order of five years not being unusual.

3. **Technology infusion is part of the total environment of the consumer organization.** Technology infusion does not occur in a vacuum. The SEL experience with Ada is such an example. Ada proved to be successful with flight simulators. However, the operational system for flight dynamic software was the IBM mainframe, and no effective Ada compiler was available during the 5 years (from 1985 to 1990) when Ada was under evaluation. The "window of opportunity" on using Ada has passed and FORTRAN remained the language of choice for such applications. Had an effective mainframe Ada compiler been available, then the result of evaluating Ada for large systems might have been different[3].

4. **The government can have an impact on technology infusion.** As others have shown, the imposition of rules mandating certain technologies (i.e., the rule model) is generally not very effective. However, the experiences with inspections and CLIPS demonstrates that the government can employ an effective people mover strategy to infuse technology. In both of these cases, advocates were recruited from NASA centers different from the development group. Those advocates started to use these technologies with contractors working for them. The transfer process occurred as other development groups within the contractor organization noticed the technologies that their colleagues were using and then voluntarily decided that they were effective for solving certain problems. Project by project, these technologies gradually spread among the contractors without the need for mandates.

5. **People contact is the main transfer agent of change.** As many others have observed, technology transfer occurs best when the developers of a technology are involved in the technology transfer process. In our study, that happened in order for Cleanroom to be effectively used at GSFC, for inspections to be brought first to JPL and then to Langley, and for CLIPS to develop an initial set of users. Finding the appropriate advocate to act as gatekeeper for the technology is a crucial component of any technology transfer mechanism.

6. **Timing is a critical decision.** This can be either a positive or a negative influence. When to enforce a decision is important for its adoption. The TAE experience at GSFC shows that early mandating of an immature technology may have the paradoxical consequence of delaying an effective technology even more than by not mandating it at all.

   On the other hand, the early studies of Ada led to the observation that object-oriented technology might have an impact on the organization. The result of this observation was an improvement in the use of Ada as well as vastly improved FORTRAN code being produced.

Technology infusion today is generally ignored and left up to the individual engineer to discover what is needed and available. With today's shrinking budgets and the need to work "Better, Faster, Cheaper," management needs to address this issue and help in the search for new effective technology to use.

[3]For the past several years the use of C and C++ has been growing within this community, and it now looks like FORTRAN may be replaced by C++ as the language of choice for flight dynamics appplications.

5 Acknowledgment

This research supported in part by grant NSG-5123 from NASA Goddard Space Flight Center to the University of Maryland, College Park, Maryland. Frank McGarry of Computer Sciences Corporation, formerly of NASA/GSFC, and Kellyann Jeletic of NASA/GSFC helped in collecting the data described in this paper.

References

[1] Allen, T. J., *Managing the Flow of Technology*, MIT Press, Cambridge, MA, 1977.

[2] Basili V., M. Zelkowitz, F. McGarry, J. Page, S. Waligora, and R. Pajerski, SEL's software process-improvement program, *IEEE Software*, (November, 1995) 83–87.

[3] Bailey J., S. Waligora and M. Stark, Impact of Ada in the Flight Dynamics Division: Excitement and Frustration, 18th *Software Engineering Workshop*, SEL-93-003, NASA/GSFC, (December, 1993), 422-449.

[4] Basili V. R., The experience factory: Can it make you a 5?, 17th NASA/GSFC Software Engineering Workshop, Greenbelt, MD (December, 1992) 55-64.

[5] Berniker E., Models of technology transfer (A dialectical case study), IEEE Conference: The New International Language, Portland, OR, (July, 1991), 499-502.

[6] Fowler P. and L. Levine, A conceptual framework for software technology transition, Software Engineering Institute, Carnegie Mellon University, SEI-93-TR-31 (December, 1993).

[7] Lucky R. W., *Silicon Dreams: Information, Man and Machine,* St. Martin's Press, New York, (1989).

[8] Mettrey W., A Comparative evaluation of expert system tools, *IEEE Software* 24, 2 (February, 1991) 19-31.

[9] Nochur K. S. and T. J. Allen, Do nominated boundary spanners become effective technological gatekeepers?, *IEEE Trans. on Eng. Management* (39)3 (1992), 265-269.

[10] Paulk M. C, B. Curtis, M. B. Chrissis, and C. V. Weber, Capability Maturity Model for Software, Version 1.1, Technical Report SEI-93-TR-24, Software Engineering Institute, Pittsburgh, PA (1993).

[11] Redwine S. and W. Riddle, Software technology maturation, 8th IEEE/ACM International Conference on Software Engineering, London, UK, August, 1985, 189-200.

[12] Rogers E., *Diffusion of Innovation*, The Free Press, New York, (1983).

[13] Seidewitz E. and M. Stark, General Object Oriented Software Development, SEL-86-002, NASA/GSFC, August, 1986.

[14] Schnaars S., *Megamistakes*, The Free Press, New York, (1989).

[15] Waligora S., J. Bailey, and M. Stark, The impact of Ada and object-oriented design in the Flight Dynamics Division at Goddard Space Flight Center, NASA Goddard Space Flight Center Technical Report SEL-95-001 (March, 1995).

[16] Zelkowitz M. V., Assessing Software Engineering Technology Transfer Within NASA, Software Engineering Program Technical Report NASA-RPT-003-95, NASA, (January, 1995).