Carnegie Mellon University
Research Showcase @ CMU

Department of Electrical and Computer Engineering

Carnegie Institute of Technology

1993

Software doesn't transfer, people do

D. M. Steier
Carnegie Mellon University

Robert F. Coyne

Eswaran Subrahmanian

Carnegie Mellon University.Engineering Design Research Center.

Follow this and additional works at: http://repository.cmu.edu/ece

This Technical Report is brought to you for free and open access by the Carnegie Institute of Technology at Research Showcase @ CMU. It has been accepted for inclusion in Department of Electrical and Computer Engineering by an authorized administrator of Research Showcase @ CMU. For more information, please contact research-showcase@andrew.cmu.edu.

NOTICE WARNING CONCERNING COPYRIGHT RESTRICTIONS:
The copyright law of the United States (title 17, U.S. Code) governs the making of photocopies or other reproductions of copyrighted material. Any copying of this document without permission of its author may be prohibited by law.

Software Doesn't Transfer, People Do

D. Steier, R. Coyne, E. Subrahmanian

EDRC 05-69-93

Software Doesn't Transfer, People Do

(And Other Observations from An EDRC Workshop On The Role Of Software In Disseminating New Engineering Methods)

David Steier, Robert Coyne^Eswaran Subrahmanian
Carnegie Mellon University

Abstract

On January 19, 1993, approximately forty people from EDRC and its affiliates participated in a workshop on the role of software in disseminating new engineering design methods. Several case studies of method dissemination involving software were presented and analyzed. This report summarizes the workshop talks and discussions, which revealed several "myths" regarding software technology transfer and strongly supported a model in which people, rather than disembodied software systems, are seen as the primary agents for technology transition. However, lest the role of software itself in technology transition be completely overshadowed, the report concludes with some reflection on the potential for improving the quality and usability of research software systems (in contexts where this is a desirable goal). The workshop suggests that an important step in this direction is better calibrating the *expectations* and *incentives* of the academic and industrial players in the technology transition partnership.

1. Introduction

On January 19,1993, the Engineering Design Research Center of Carnegie Mellon held a workshop on the role of software in disseminating new engineering design methods. Approximately forty people participated in the workshop, a group consisting of representatives of EDRC's industrial affiliates and EDRC faculty, staff and students. The original motivation for the workshop, in its essence, was to brainstorm on the question, How can we use the software that we produce in research as a more effective vehicle for transitioning the results of research?" (This entails the more pragmatic concern "How can we get others (mainly in industry) to use more EDRC software?[19]) As we soon discovered, framing the question in this way suggests some familiar but strong assumptions about our research software development:

1. that EDRC researchers develop software in a process that is perhaps sponsored by industry but conducted mostly independently of the intended users
2. that the software thus developed will be usable in a production setting
3. that the recipients of the software bear primary responsibility for incorporating the software into their engineering practice.

A possible motivation behind these "typical" assumptions is that on the surface they seemed to reflect an efficient division of labor. We had all seen these assumptions violated in our own individual experience, but were perhaps too inclined to regard these occurrences as 'aberations' with the possible interpretation that we were just victims of some unique circumstances. Therefore, going into the workshop a naive but widely shared view (hope) was that surely somebody out there had managed to do it "right," and all we needed to do was talk to them to extract the formula that would allow more EDRC software to be transferred.

We now believe the model of software transfer underlying these assumptions to be a myth. Like all myths, it no doubt serves some useful purposes— for instance, the prospect of a large user community is often a significant motivator for commercial software developers, and researchers who happen to develop software may be similarly motivated. However, as the workshop revealed, those who would use software to fKsy«"i"?!* innovative methods - such as those developed at EDRC - might want to use a different process raodeL At the heart of this (more realistic) model is a complex interaction between technology developers and users, a process that CMU's Software Engineering Institute and other organizations have termed "technology transition," as opposed to "technology transfer.[11] In the above usage, "transition" has become a verb, and while the resulting damage to standard English usage is noticeable, the word sense is a useful one: to "transition[11] a technology implies a gradual extended-duration, collaborative effort in contrast to the one-time hand-off implied by "transfer." In fact, this shift captures much of the spirit of the workshop. In discussing the case studies presented at the workshop, participants consistently described experiences that were best characterized as technology transition.

In support of these observations, this report will first present a more extensive list of the "myths" of software technology transition together with a summary of the more realistic observations concerning transition that emerged from the workshop. Next, the transition process for several case studies will be discussed in more detail, in domains that include electrical, mechanical, and chemical engineering. Then, we list a set of features for characterizing these and other case studies, and use our examples as data for some tentative findings based on these features. Finally, we describe some tradeoffs and recommendations that should be considered in light of our collective experience of software transition. While the structure of this report will not follow that of the workshop exactly, participants should recognize the information contained here.

2. Software technology transfer: myths and reality.

Myths of software technology transfer.

We list here (non-exhaustively) a typical set of assumptions (or perhaps more accurately prejudices) that contribute to the myth that technology transfer through software takes place readily on the basis of a simple, direct raodeL

1. Use of new technologies requires a simple transfer from the originator to recipient of the programs and documents.
2. Technology transfer takes place from universities to industry only.
3. Technology transfer is a purely technical issue and not organizational.
4. Technology transfer is a relatively cheap process.
5. Technology transfer is the job of programmers.

These myths arise from viewing the software development and deployment as a sequence of "over the wall" activities. This compartmentalization results in improper assignment of priorities and responsibilities and misaUocation of resources. The above observations are reinforced by the real experiences reported below.

The realities of experience.

We summarize the experiences of successful technology transfer from the case studies presented in the workshop in the following observations:

- **First-generation software** is **rarely used:** In all the case studies we examined at the workshop, the first generation of the software had to be rewritten, and sometimes several times, before that software could be used routinely in an industrial setting. On the basis of their experiences, many in industry would not use software developed at a university at all, even if it were free.

- Disseminating new methods **through software** is **resource-intensive:** Our case studies show that the progression from an idea in a researcher's head to software product takes at least a decade (more than two Ph.D. cycles, to use academic time units) and a commitment by a team of people, at least 30 person-years of effort Because the process is hard work, the people involved need to be well motivated, which is most likely if organizational incentives encourage method dissemination and acquisition.

- **A wide variety of skills are needed for successful method dissemination:** In our case studies, we saw a variety of transition mechanisms, ranging from the use of internal research faculty to the creation of start-up companies. We found success turned out to be less dependent on the exact organizational arrangement used than on the distribution of the necessary skills. These skills include managerial and business expertise as well as the more obvious application domain and software skills.

- **Software doesn't transfer, people do:** Of course researchers try hard to write scientific reports, documentation, and interfaces that make the utility of their software self-evident. In spite of all that, papers and manuals go unread and software goes uninstalled. In the success stories we examined, there was always the element of a personal presence. People who had developed a technology, or been thoroughly trained in its use, at one organization often physically relocated to another organization. Graduating students and long-term visitors played an essential, not just incidental, role in the transition.

3. Case study: MICON

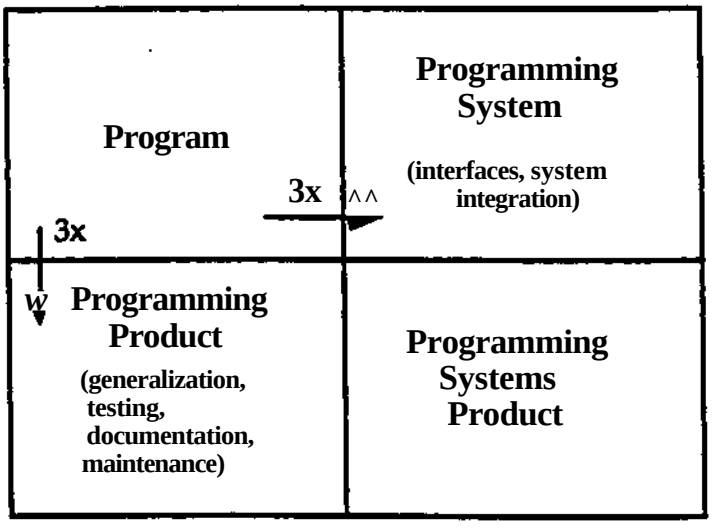
The first case study examined in the workshop was MICON, a prototyping tool that can produce electronic designs for a single board computer. The input to MICON is a set of requirements for processor, memory, and input/output components, and constraints on area, cost and power. The output of MICON can be (and has been) fed into commercial tools for physical design and manufacture of the computer so that a prototype can be built and tested. MICON uses a technique called "synthesis by composition[1]" to develop the design. The system applies a design knowledge base of several thousand rules to a database of six hundred parts to instantiate design templates, that, when composed, yield a complete computer design. By now, MICON has created dozens of realistic-scale designs, and incorporates several support tools, including a knowledge acquisition tool called CGEN, which allows domain experts to add their own design knowledge.

The development of MICON was supervised by Professor Dan Siewiorek of Carnegie Mellon, who gave the presentation at the workshop. Siewiorek made clear the extended duration and scope of the project as he described the evolution of MICON in its incarnations as MO, M0.5, and Ml. In developing MICON, 55,000 lines of code were written for the primary system, 19,000 lines for the support tools and 58,000 lines that embody designs (about 17,000 of the total have been made obsolete in later versions of the software). The project began in 1982, and has involved a total of about 38 person-years of effort External interactions have included nine workshops for industrial and academic audiences, and the software has been field-tested at eight companies for varying lengths of time.

In 1991, a start-up company called OmniView acquired a seven-year exclusive license to develop a commercial version of MICON. The CEO of OmniView is Charles Buenzli Jr., who followed Siewiorek's presentation at the workshop to give the perspective of an intermediary organization. Omniview's goal, as Buenzli remarked, was to "re-manufacture" the software for industrial use. The project, code-named Fidelity, used members of the original MICON development team to specify the software, but carried out the re-implementation on its own, developing some of the software off-shore. To obtain increased performance and maintainability, Fidelity replaced the rule-based and relational database technology of MICON with its own object-oriented and constraint technology. Additionally, the textual interfaces to MICON and the support tools were replaced with graphical Motif-based interfaces. In developing Fidelity, the dominant requirements, besides performance, were interoperability with other design automation tools, portability and extendibility, so industry standards were used wherever possible. The resulting 110,000 lines of code *does not re-use any of the original MICON software,* but Buenzli estimates that the availability of MICON as a prototype saved over a year in development time. The prototype guided the specification and analysis of performance bottlenecks before implementing FIDELITY.

Both Buenzli and Siewiorck agreed that the MICON/Fidelity situation was nearly an ideal one from a technology transition perspective. By the time OmniView began to turn MICON into a product, the CMU team had refined the original ideas through several prototypes (MO, M0.5 and M1). Those ideas had been tested extensively on realistic-sized problems, i.e., the design of a 386-based personal computer. CMU personnel developed MICON by taking advantage of free, readily available software such as the OPS83 production system language, which permitted rapid prototyping of new ideas. Once these ideas had been tested, OmniView provided the skills necessary to make the code comply with industrial standards, such software implementation in C or C++, and to make Fidelity available on multiple platforms. The process was also aided by extensive exposure of university and commercial personnel to one another's concerns: OmniView personnel went through extensive workshops run by CMU researchers, while CMU researchers were sensitized to the commercial implications of the technology and given incentives to see the commercialization succeed.

Despite the long gestation period for the ideas in MICON, the guidance from the original development team, and the near-ideal distribution of skills, Siewiorek and Buenzli still witnessed the essential truth of a software development reality embodied in a classification first described by Fred Brooks (Brooks, 1975, p. 5), termed by some "the father of the IBM/360". This classification is shown below, illustrating the differences between producing a working program, producing a programming system complete with tools and interfaces, and producing a programming product, complete with documentation, testing and multiple platform support Brooks estimates that creating a programming system from a program multiplies the original development effort by a factor of three, creating a programming product from a program also adds a factor of three. Creating a programming systems product involves both of these transitions, yielding a combined factor of nine times the original effort to create a program.



4. Case study: CGS Interface

Professor Fritz Prinz and Andrew Jones of Inland Fischer Guide presented the second case study, the interface from General Motor's Corporate Graphics System (CGS) to several other projects including moidability and lighting design critics, a stereolithography apparatus (SLA), and finite element method software. The CGS interface is based on the CMU-developed NOODLES geometric modeling package. NOODLES is a non-manifold modeling system in which objects of varying dimensionality may be freely combined, unlike wireframe, surface-based, or solid modelers. A number of projects at EDRC that represent and manipulate geometric information have been based on NOODLES. The interface between CGS and the SLA is currently unique in GM; it currently drives an SLA-500 machine at the rate of about 40 parts per month. Future plans call for further fine-tuning and documentation of the program and providing support through Electronic Data Systems.

In contrast to the use of an intermediary start-up company for the commercialization of MICON, the development of the CGS interface and transfer of the technology to GM is the result of a direct partnership between Inland Fisher Guide and CMU, While it is too early to assess the prospects of long-term success for either system, a consensus emerged during the discussion that both arrangements can succeed if certain preconditions are met These preconditions include having people with the appropriate collection of skills available, and sufficient resources to ensure a successful transition. However, it was cautioned that very close, direct collaboration may impose excessive time demands on the developers (who in academia are usually already overloaded students and staff), and also often exposes users to untested and unstable software. Whatever the arrangement, workshop participants concurred that transition could not succeed without realistic expectations from all parties. These expectations could be set by maintaining an ongoing planning process, negotiating goals with time targets, and using feedback to evaluate progress towards goals.

In a discussion of the MICON and CGS interface case studies, Dr. Ted Giras of Union Switch and Signal pointed out that one of the reasons technology transition is so difficult is the need for recipients to manage risk while introducing high-impact technology. In the technology maturation cycle, risk decreases as time goes on because the prospects for a technology can be better evaluated. In contrast, the amount of money that must be committed increases as time goes on.

Deciding at what point in time the risk is sufficiently low to merit an increase in investment, Giras argued, is best done by well-informed people who have a long-term strategic view of how technology can be developed and used. In contrast to this ideal, the typical program manager may have a tactical planning horizon as short as 30 days. The ensuing discussion was wide-ranging, spanning topics from how to find a market for possibly idiosyncratic solutions to particular industrial problems, to the desirability of involving academics in efforts at developing standards. It was also noted that industry may have good reasons for not wanting to use university-developed software; the software may

not be sufficiently standardized or reliable for some applications, particularly those which are safety-critical.

5. Chemical engineering software case studies: ASPEN, ASCEND, DICOPT++, and GAMS

Dr. Jeff Siiroia of Eastman Chemical began the afternoon sessions by discussing the development of ASPEN, a process simulation package. ASPEN was developed under contract to the Department of Energy to solve process design problems associated with the development of coal liquefaction and gasification technologies. Algorithms for simulating distillation processes that had been developed by academics were incorporated into ASPEN. Although software engineering standards (of mid-1970s vintage) were used in the development of ASPEN, the original ASPEN software did not get transferred after the DOE contract was concluded. Siiroia listed a variety of reasons for this, including missing features and unfixed bugs in the software, and he pointed out that the original software developers may not have intended to transfer ASPEN to other users. Some of the users formed a company, ASPEN Technology, and managed to rewrite ASPEN so that it could be a viable commercial product Siiroia credited the success of this later effort to the close involvement of industrial users in the development of the software, and to the fact that the software was not too ambitious: that it was not at the forefront of technology. Vincent Vermeuil of Simulation Sciences (a competitor to ASPEN Technology) challenged Siiroia's interpretation of this success story, maintaining that ASPEN's competitive advantage derived from the DOE funding of its original development (no ASPEN Technology representative was present at the workshop to respond).

This case study was followed by a commentary from Dr. Jerry Robertson of Exxon. Robertson identified three categories of barriers to transition of technology from academia: funding, resistance to change, and low confidence in technology. In terms of funding, he noted that technology transition competes for resources with in-house development efforts and thus is one of the first expenditures to be sacrificed in difficult times. Like Ted Giras, he noted that resistance to change was often large because of the costs of failure and the difficulty of making the benefits of change apparent to all. He also listed several factors that needed to be considered in increasing confidence in technology, factors that are often accorded lesser priority in the development of new methods: usability by novices, traceability of changes, and compatibility with existing systems and methods. These factors are well known to consume a large amount of resources in most system development efforts - in many cases a multiple of the effort to produce the core system functionality itself.

One of the research efforts at the EDRC that has addressed this last set of concerns is the ASCEND project, directed by Professor Art Westerberg, who talked about ASCEND for the workshop. ASCEND is an environment that aids engineers in managing the development of mathematical models, which often grow to include tens of thousands of equations and variables. For chemical engineers, the use of the equation-oriented

simulation approach supported by ASCEND represents a significant paradigm shift from the more popular sequential modular approaches. Debugging ASCEND models is an anticipated and important part of the model development process, but little is known about how users can take advantage of the debugging possibilities provided by the ASCEND interface. Addressing usability concerns that might inhibit the adoption of the methods embodied in ASCEND thus became a cornerstone of the research. Therefore, Westerberg, and Peter Piela who developed the ASCEND software first as Westerberg's graduate student and later as member of the research faculty at the EDRC, made a special effort to solicit feedback from users on ASCEND. Both the hardware and the software needed to run ASCEND were shipped to seven companies, two of which then began to use ASCEND extensively and provided feedback. Despite the useful feedback, industrial users were more interested in software that was fully supported (e.g., with a technical support hotline to call) and operated robustly on multiple platforms (ASCEND was available only on the Apollo).

According to Westerberg, another inhibitor to more widespread use of ASCEND is the steep learning curve. This is true despite the fact that once one becomes familiar with the system, there are significant benefits with respect to ease of model development. Apparently, learning time is a barrier even for other academics who would rather not spend very much effort learning to use someone else's software — a twist on the "Not Invented Here" syndrome most often associated with industry. The demonstrated successes of ASCEND come primarily from Westerberg's own students, some of whom, he claims, have developed elegant models that would be "publishable" if there were an appropriate journal in which to do so. Two of Westerberg's students, Piela and Oliver Wahnschafft have taken positions with ASPEN Technology, and this may ultimately lead to the transition of ideas from ASCEND and their incorporation into commercial products.

The final case studies of the workshop began with Professor Ignacio Grossmann's presentation on the creation of DICOPT++, a system for the efficient solution of mixed-integer non-linear programs (MINLP). Like the other systems discussed in the workshop, DICOPT++ is based on a series of developments spanning over a decade. These developments began with the idea of casting integrated process design as superstructure optimization of mixed-integer linear programs (MILP) in the early 80s, followed by the use of outer approximation to solve MINLPs and the use of equality relaxation that allowed solutions of larger problems. DICOPT++, developed over the last three years and maintained primarily by J. Viswanathan at EDRC, applies these ideas using the GAMS modeling environment and a number of solvers to solve non-linear programs and MILPs that are formulated as subproblems at different steps of an iterative process. DICOPT++ itself has served as a component of other systems, such as SYNHEAT, which creates optimal designs for heat exchanger networks. It has also been used independently to solve previously intractable problems, such as the cyclic scheduling of parallel continuous lines, in which an industrial scale problem may have on the order of a thousand integer variables, tens of thousands of continuous variables, and several thousand constraints.

The development of DICOPT++ was a collaboration between Carnegie Mellon and GAMS Development, the company responsible for the GAMS environment on which DICOPT++ relies. GAMS itself was the subject of the presentation by Alexander Meeraus, the president of GAMS Development GAMS is an interface between the user's formulation of a mathematical model and the range of solvers and hardware platforms on which the model could be solved. The system has its roots in software developed at the World Bank in the 1970s to help solve economic models used in policy analysis. In the late 1970s and early 1980s, the software was rewritten several times, with outside industry participation, to serve as a more general algebraic modeling system. In 1988, members of the original World Bank Development team formed the GAMS Development Corporation to position GAMS as a commercial product GAMS relies on DICOPT++ to provide a capability for solving MINLP problems, and the company is looking to incorporate the latest CMU results on integration of symbolic logic for solving MBLPs. In return, CMU obtains access to commercial grade software for building and solving models, including the support for multiple platforms that is so rare in an academic setting (it is absent in ASCEND for example). So the GAMS/CMU partnership is an instance in which both academia and industrial parties obtain benefits from the collaboration — the technology transition is bi-directional.

It may be instructive to compare the position of GAMS Development with that of OmniView, which is also positioning itself as an intermediary conduit for technology between academia and industry. As in MICON, the GAMS software had already been developed through several generations before the decision to commercialize it, and the total effort invested had been on the order of 30 to 40 person-years. The markets for the two products, however, seem substantially different Meeraus believes that after two decades of refinement, model solution technology is relatively mature. He thought that entry by a new company into the field would be very risky, based on a market potential for general purpose solution systems of only thousands, in contrast to the hundreds of thousands of computer systems designers that Buenzli of OmniView thought could benefit from a product like Fidelity. Meeraus believed that the primary market growth for the future would come from GAMS-based systems tailored to particular domains, and from providing interfaces to new solvers and platforms; the idea of producing customized versions of the core technology also figures into Omniview's plans. On the other hand, Buenzli estimates that half of Omniview's revenues in five years will come from selling standard libraries for particular design applications.

S. Analyzing the Case Studies

At the beginning of the workshop, Professor Steven Fenves presented a list of features that might be useful for classifying and therefore better understanding the case studies. The classification was refined during the course of the workshop, and summarized again in the concluding session. Below, we use a version of the list of features presented at the concluding session to organize some analysis of the case studies:

Directionality: Although we normally think of technology transition in terms of transition from academia to industry (e.g., MICON), it can take place, as we saw, in the reverse direction from industry to academia (e.g., GAMS), or from one company to another, or from one university to another. Additionally, government agencies can be consumers of technology (as discussed by a representative of the National Institute of Standards and Technology at the workshop), and developers, as in the case of the World Bank creating GAMS. Workshop participants seemed to agree in the end that no single uni-directionai transition is more likely to succeed than others. However, the cases from the workshop suggest that the best situation exists when technology transition is possible in two directions simultaneously, so that mutual benefits can sustain the collaboration.

Cardinality: The number of parties involved in the transition can vary, but we identified two important special cases. In the first case, there are two parties, an originator and recipient, and the transition is direct (e.g., the NOODLES-based CGS interface). In the second case, there are three parties, the originator, recipient, and an intermediary, such as a spin-off company or already established software vendor. As with directionality, we found that cardinality was important in a different way than we expected: the absolute number of organizations involved was not as important as ensuring the right skill mix and resources in the total process.

Messenger: The vehicle for transition may be a paper-based description of the method, software embodying the method, or an individual who champions the technology in a new organization. In every case discussed in the workshop, individuals played a key role as messengers. We found no instances of unaccompanied software or descriptions for very innovative methods finding their way into the practice of recipient organizations. It appears that if a new method is radical enough, recipients need long-term help from originators to adapt the method for use by their organizations, and this often means the physical relocation of originating personnel. The role of individuals was so ubiquitous and so striking that we decided to include this observation in the title of this report

Software: If software plays a role in the technology transition, the factors that influence its success include:

- System characteristics: program architecture, language(s), hardware requirements, interoperability with other software, performance, portability, etc. Another important characteristic is availability of documentation, which may range from a brief description or appendices in a thesis, to a full set of requirements specifications, user and maintenance manuals.
- Development process characteristics: software engineering standards adhered to, the degree of internal testing, etc.
- Distribution method: Software may be available at the originator's site, distributed to recipients, or installed at the recipient's site. The more work a recipient has to do to obtain the software, the less likely that software is to be used.

- Cost: The cost of the commercial versions of some of the systems we examined ranged into the tens of thousands of dollars, but the academic organizations that developed the research prototypes did not charge recipients specifically for the software (as opposed to getting sponsorship for the research in general). Furthermore, getting user feedback on prototypes is easiest if the prototype depends only on easily available, preferably.free, components. Getting users to try out prototypes is so difficult that it seems to be essential to remove all possible financial obstacles to such experimentation.

Originator: The originating organization could be characterized in many ways:

- Intent: whether the originator wants to produce a feasibility demonstration, a prototype for use as a foundation for further development, or a production quality system. In all workshop case studies, a commitment to facilitating the creation of a production quality system was required on the part of an originator before the technology transition could occur, although this commitment was not always present at the very beginning of a project

- Openness to feedback: whether the originator is willing to consider feedback from recipients or potential recipients in refining the method. Successful transition seems to require openness to feedback, and the willingness to participate in an extended loop with users. This is especially true where the user interface is concerned, where the state-of-the-art does not allow prediction of user preferences and adequate handling 'of other usability issues without user participation and iterative development

- Resources available: time, money, personnel. Although we did not get estimates of dollar amounts involved, all the case studies we looked at involved over 30 person-years of effort and took over a decade for something approaching effective transition to occur. The arrangement for sponsorship of the research, whether it is internal funding, from government contract or otherwise externally funded is also important *

Recipient: These characteristics complement those of the originating organization:

- Intent: whether the recipient is going to use the new method to conduct further in-house research, or to use it in actual production. The intent is a component of setting reasonable expectations, whith are critical for obtaining management support and tolerance for risk.

- Willingness to provide feedback: necessary if the originating organization is to improve the software over time.

- Resources: how much money and effort can be spent to acquire new capabilities. If the methods are sufficiently different from the current practice of the recipient organization, there must be support for overcoming the learning curve.

Intermediary: If an intermediary is involved, then that intermediary will have to share characteristics of both originators and recipients. We found that even in the successful transition cases where there was not a separate intermediary organization, there was always

a person whose primary duties were neither research nor production so that they could take on the functions of an intermediary.

Shared context: A number of other features describe characteristics shared by originators, recipients, and intermediaries in that they describe the environment in which the transition takes place. These include:

- **Goal clarity:** the degree to which the goals of all parties have been set and examined, articulated within each organization and communicated to other parties. The workshop case studies bear out the intuition that the greater the degree of goal articulation and communication, the higher the probability of success.
- **Alternative methods:** in investigating the possibility of adopting a new method, a recipient organization compares it with alternatives, which may differ in their availability and desirability. The tradeoffs most often considered involve impact on product quality and on the speed and cost of the development process. If the new method is sufficiently unfamiliar, as in the case of ASCEND, assessing such tradeoffs is more difficult than in the case of a new MINLP solution method that can be shown to solve a benchmark model an order of magnitude faster than previous methods.
- **Number of people affected:** if a method is to be phased in slowly so that it affects only a few people at first, the recipient organization will be much more tolerant of unfamiliar (and hence riskier) methods than they would if large numbers of people would be affected by the change.
- **Distribution of skills:** whether the necessary software, application domain, and managerial skills exist and where they are located, are ail issues that are an important determinant of a successful transition.

We do not intend for this list of characteristics to be definitive or prescriptive for future technology transition processes. In particular, organizational issues in technology transition, which were not examined in any depth at the workshop, may be as important, or more so, as any technical issues.

Organizational issues: Organizational issues, especially for the transition of very innovative methods, do not yet seem amenable to classification in terms of a small set of features. Betty Deimel of the Software Engineering Institute gave a talk at the workshop that illustrated the complexity of these issues. While not specifically addressing the use of software as the agent of transition, Deimel noted that all transition involves change, which implies that there will be resistance to that change. One approach to overcoming that resistance is to "hammer" the initial state into the desired state, which may be easier in the short run, but incurs long-term costs of anger, burnout, even sabotage. Another approach manages the transition by "unfreezing[11] the present state, allowing for change but also initial confusion and stress that may decrease productivity in the short term. In the long term,

"refreezing" the practice of the organization in the desired state increases productivity. 1 During the transition stage, information, dialogue and pilot tests precede technology transition to foster awareness, understanding, and trial use, before the new methods are adopted and institutionalized.

During this freezing-unfreezing process, the people involved in the transition play a variety of roles within a recipient organization. Deimei follows Rogers (Rogers, 1983) in classifying these roles: *sponsors,* who provide the resources for the transition, authorizing and reinforcing the effort; *champions,* who influence the organization in the direction of technology adoption; *agents,* who are actually empowered to create the change; and *targets,* who are the behavioral focus of the transition effort

In fact there is a sequence of attitudes exhibited by targets that can be predicted, starting from initial optimism that the new method will solve all problems, followed by a pessimism and decreased energy as these expectations are not immediately fulfilled. Targets may then "check out" of the transition process, either in public or in private, or they may work through the problems, so that hope gradually increases and satisfaction is obtained. This sequence compares with the "grief curve[1]* proposed by Elizabeth Kiibler-Ross (Kübler-Ross, 1970) and others, in which the initial reaction to change is stunned paralysis, followed by denial, then rage, then unsuccessful bargaining, leading to depression. New beginnings start from this depression, first by a period of testing the waters, followed by acceptance. Being cognizant of such stages, Deimei argues, allows the managers of change to allow time for each stage to pass, and to use negative incidents as opportunities to transform resistance into support.

6. Workshop discussion and afterword.

The workshop concluded with a discussion moderated by Robert Coyne and a summary presented by David Steier. We began by returning again to the emergent theme of the workshop: that the simple model of method dissemination via direct transfer of software from academia to industry faced a number of problems. These problems are due to a fundamental gap in the *incentives* and *expectations* between those producing software in academic research settings and potential recipients and users in industry. For example, there is usually an gap between the software interest, resources and possibly capabilities of researchers, who are best at rapidly prototyping small-scale demonstrations of new concepts, and the requirements of users, who want "usable" software to be thoroughly tested on realistic-sized problems, well-engineered to industrial standards and supported for a variety of platforms - but do not always expect to share in the cost of time and resources required. We also recognized that researchers often do not intend for their systems to go beyond the feasibility demonstration stage. Furthermore, in developing research software, all the classic problems of software development, including unpredictability of resource

---

^ Thus organizations must be different from ice cream* which must be whipped after unfreezing to achieve the desired state after refreezing.

requirements and difficulty of maintaining up-to-date documentation, are compounded. In concluding the discussion, a "joke" with a serious point was suggested by the moderator software customers, who are accustomed to being able to pick two out of three trade-off criteria used for software products: "robust, fast and cheap", in the context of research software get to pick only one.

The Role of the EDRC.

Given this situation, we began to ask what role academic research centers with strong ties to industry could play in all of this. In theory, the nation's Engineering Research Centers (of which EDRC is one) could address these issues. An important component in setting the goals, direction and focus of strategic research at the EDRC is an industrial planning committee which is composed of senior engineers and managers from a variety of industries. On average, between one third and one half of EDRC research projects are conducted with the participation in some form of industry, many of these with full or partial industry funding, and some with close collaboration and continuous exchange with specific industry personnel or groups. EDRC performs much of its research by embodying new methods in software prototypes, and therefore already has significant software skills among its faculty, staff and students. Furthermore, EDRC is different from other academic organizations in that long-lived, multi-generational systems, such as MICON and ASCEND, are much more common, more like the norm rather than the exception. These systems are used for iterative development and refinement of ideas, and some EDRC systems are used as platforms on which other systems are built So there is an incentive to maintain high software quality, even if only for internal maintainability and understandability. One might expect that these circumstances would lead to a situation where a greater proportion of EDRC software is suitable for transfer to external use.

The workshop case studies showed us that the situation is never quite that simple, and this report has already discussed a number of reasons for this. In the discussion and summary, we returned to the software development process and expanded on several of these reasons. We first noted how important it is for software to be usable to be accepted. Usability requirements are the ones most likely to vary between research and external use, especially for systems embodying new methods. Unfortunately, developing and refining a new user interface is often very expensive, with several studies of software development showing that half the effort and code in many systems is devoted to the user interface (Myers and Rosson, 1992). In addition to the interface itself, new research software systems (such as ASCEND) also offer users a new paradigm in the practice of their (engineering) discipline. This generally implies that there are multiple possible interaction scenarios by which users might utilize the underlying functionalities of the system and integrate these into their work flow (Jacobson et al, 1992). Exploring, prototyping and validating a useful set of these interactions with the system - each of which may have many interface alternatives - requires iterative development, user participation and testing (Floyd et al, 1989). To build good system interaction cases and their interfaces, researchers must collaborate closely with users, but in ways they may feel does not contribute directly to the

core technology they are developing. Aside from issues of interest and capability, ultimately there is the question of who will pay for all of the design effort involved in making software "usable", and the necessary impiementational "builetproofing" and testing, given that no technology transition will occur in the absence of these development efforts.

Another point relates to the software technology used to implement these systems. A variety of tools facilitate rapid prototyping of software, ranging from languages such as Lisp and the various production systems, to UNDC™-based tools such as awk and perl, to the interpreted command language Tel and its associated XI1 toolkit, Tic, which allow very rapid construction of graphical user interfaces. For a variety of reasons, such as efficiency, or the necessity to conform to accepted standards, industrial users will often refuse to touch software constructed with these tools. Yet the flexibility these tools permit is essential in a prototyping context, so it may be best to accept that software developed in one technology may have to be rewritten for another, as it was in the transition from MICON to Fidelity. In the words of Fred Brooks (Brooks, 1975), "Plan to throw one away; you will anyhow." So there are tradeoffs to consider, which may indicate that aiming for direct transfer of software is typically not as desirable an option as other alternatives.

We do not list these factors to argue that organizations like the EDRC should stop producing software completely, nor should they stop aiming to improve the quality and usability of research software systems. The usefulness of such systems is important, in conjunction with people, as vehicles for transitioning new methods. Embodying new methods within software, and refining those methods over several generations of systems, teaches us more about the implications of those methods than hand simulations. Furthermore, usability and realistic scaling of systems are becoming increasingly important in exploring higher and deeper level issues (in engineering design methods and practice) that can only be investigated when there is the opportunity to build upon already existing systems and in conjunction with real industrial applications (Coyne et al, 1993). In the case of interactive systems, there may not be a way to determine an appropriate design for the interface without building one (or more) and evaluating the results. Educationally, EDRC students benefit from constructing multi-generational software systems and interacting with software experts. They acquire practical skills in this way that might be difficult to obtain in their home engineering departments. And in turn, if they acquire more knowledge of what software has already been developed by others, they might avoid reinventing the wheel (which unfortunately still occurs far too often), improving their research productivity, and even helping them to graduate faster.

What the workshop does suggest to us, however, is that we may benefit from the recognition that people, rather than software, have been the primary successful messengers for technology transition. One could say that what researchers do is produce knowledge in the form of theories, in EDRC's case new engineering methods. Both publications and

software systems are descriptions of methods, but they are incomplete descriptions.[1] The missing part is an understanding of how the method can be used and the process by which a method might be adapted to meet a given organization's needs and integrated into the work practice. People are necessary to supply that understanding, particularly when there is a substantial mismatch between the current practices of the recipient organization and the practices needed to use the new method. Peter Naur (Naur, 1985) from his study of programming points out that programming is like theory construction where most of the theory is in the heads of the people and the documentation is a poor reflection of the theory. This is precisely why, he argues that the transfer of software requires that members of the team be involved because the underlying theory not being explicit, results in patched up pieces that lead eventually to failure.

Accepting this characterization of transition would have several implications. One would be a changed role for students: rather than rely on software as the primary transition vehicle, with students providing technical support back at school, it may be best for students to facilitate transition as interns, and eventually as graduates in new jobs. Courses and internships that provided some of the necessary skills would enhance career prospects, yet at the same time minimize conflicting demands on students[1] time. An *IEEE Spectrum* article (Curran, 1993), published after the workshop, supports this idea. The article reports that half a dozen CMU electrical engineering undergraduates visited semiconductor companies in the summer of 1990; they carried with them a program called AWEsim (short for asymptotic waveform evaluation) and their goal was to get industrial reaction to the program. Comments from industrial participants on the AWEsim software itself mirrored the experiences we found in our case studies: some never used the software because it did not fit their needs, while others took ideas from the program and incorporated them into their in-house software. All participants though, liked the the idea that students familiar with the program were available to answer questions; the students could do a much better job of explanation than the program and documentation alone. The article reports that the experiment only ran into trouble after the summer, when enhancements were made to AWEsim, and the undergraduates were too busy to transfer those enhancements remotely during the academic year.

More generally, of course, there is no reason that students are the only people who can facilitate technology transition; they are just the most natural candidates, because (presumably) they are going to graduate and move on. But non-students can also be effective in transition activities and might be even more effective if they possess more skills and experience. Thus, the organizations that wish to facilitate technology transition might wish to look closely at their incentive structure to see how they reward transition-related activities. Here EDRC *can* play a special role. The culture of the university at large changes slowly. It may be a long time before traditional academic departments reward method

---

[1] HOW incompteff! fflKf 1 " wha * ray* thft dncnmentalifln of research - as a mliahnrarivp design and work process- may become more complete and integrated with software systems products and communication processes integral to successful transition arc issues discussed below in the Afterword

dissemination activities as well as they do archival journal publications. But EDRC may be able to create a culture in which the incentive structure is different There would have to be rewards for enhancing the "usability" and "reuse" of research products - for making users' lives easier, for incorporating the work of others, and for exploration of domains outside one's field to understand and encourage applications for new methods. In general, the people who would engage in these activities would not be teaching faculty, who would otherwise have to juggle in technology transition along with their research and teaching activities. Rather, these people would be research staff and others for whom the retention and promotion criteria would be very different than those for teaching faculty. The key to success seems to be ensuring that technology transition activities are provided with adequate resources and rewarded (in terms of job security, salary, etc.) in accordance with the stated goals of the organization. Expectations should be explicit for all those involved, and if the expectations must change as resource availability changes, then the process for adjusting those expectations needs to be agreed upon as well.

Of course, here we have been speaking as academics, and academics alone cannot do all that is necessary. Government and industrial organizations also need to examine their own expectations and incentive structures to decide how they wish to promote technology transition. They are also in the best position to evaluate how the forces of public policy and markets, as well the attitudes of their own employees, will affect the prospects for adoption of new methods. The discussions at the workshop seem to be a good beginning.

Afterword: People alone don't transfer either.

Most engineering tasks today involve the use of software in some form or another, and, the majority of new methods coming out of the EDRC are software tools (such as NOODLES or ASCEND) or algorithms or methods embodied in software. In the work setting of today, whether it be in industry or in academic research, it takes people, working with software, to transition and employ technology effectively.

Though the workshop very strongly suggested the fundamental importance and roles of people in the transition of research, we do not wish to leave the impression that there is an "all or nothing" verdict as to the choice or usefulness of software vs. people as a medium for technology transition. The workshop started with the rather skewed perception or expectation that if we could get the software right, it could "do it all" - that is, accomplish technology transfer by just throwing the software over the wall to eager industry recipients. Guided by the realities of the case studies, and for the sake of making a critical point, we then shifted the relative balance to the view - as caricatured in the report title - that "people transfer", or for the most part, they "do it all" in accomplishing technology transition (as if the quality of the research documentation - including software prototypes - mattered little if atalL)

What this ail or nothing view obscures is the importance of and opportunity for moving toward improving the quality and usability of (research) software systems. Sharing a

more insightful and pragmatic set of incentives and expectations between academia and industry is part of the transition puzzle, as we have seen, and may lead to the production of more usable software.. We suggest that the understanding and knowledge now exist, and the time is ripe, for adopting a more comprehensive approach to software development - even within a research setting - and for establishing a better infrastructure for software design, maintenance and reuse.

This more comprehensive approach is based on considering software development as a "design" activity, and software engineering as a collaborative social process. Transition of knowledge between participants within a (software) design project, or across design projects, depends on reaching a shared understanding of many concepts, terms and issues. The clarity of the shared understanding achieved depends, in turn, on the structure of the organizations involved, the dynamics of the communication infrastructure, and the quality of the tools utilized for the establishment of a shared external record or memory of the development - in short a design history and rationale. In this sense, software engineering involves many of the same issues and problems as engineering design in industry (or generic design activity anywhere).(Garg and Scacchi, 1987; Minneman and Leifer, 1991; Levy et al, 1992) These problems revolve around the difficulties and the large overhead involved in information access, sharing, negotiation, indexing, integration and retrieval, and the management and communication breakdowns between people separated in time or space. These issues are accentuated when there are multi-disciplinary teams involved and where the participants may have vastly different expectations and goals - such as software users and developers, or industry and academic' researchers. However, as with engineering design in general, the same potential exists within software engineering for transforming these issues into new opportunities to improve the infrastructure for information integration and to examine and re-engineer practice to leverage collaboration and transition of knowledge.

Currently, there are several research groups investigating requirements and methods for enhancing collaboration in team-based design activities - such as software engineering. They have begun prototyping environments to provide computational support for collaboration in the form of electronic designers[1] notebooks, or uniform information modeling environments to capture shared explicit "memories[1]* (external records) of system development (Toye et al, 1993, Subrahmanian et aU 1993). The motivation and goals of their research efforts indicate the potential for experimenting with improved "information integration" for software engineering within research organizations such as the EDRC - entailing both organizational issues (which influence and are influenced by expectations and incentives) and technical issues. In this context, software engineering of research software systems will aim at producing much more than just code (or partially documented code). It will endeavor to also produce and preserve a history of system development in the form of a navigable web of information that interlinks requirements and a wealth of system design and development information and documents, both formal and informal, such as

negotiation of concepts and terras, system architecture issues, decisions, alternatives, code versions, testing sets, trial runs, and so on.

In addition to the potential benefit of general infrastructure improvements implied by information integration for software engineering, there are more specific methods and techniques of current software engineering practice, such as object-oriented modeling, that can be adapted and more broadly applied in research software development One such method that is growing in importance and achieving widespread acceptance is the behavior modeling of system requirements - known by various terms such as "use cases", "scenarios", "scripts", etc. (Jacobson et al* 1992; Rubin and Goldberg, 1992). Behavior modeling may be a particularly useful starting place for development of research systems, that are innovative in the sense that there is no precedent in current practice for the method or approach embodied in the system (e.g., ASCEND) (Coyne et al, 1993). A current project at the EDRC called SEED (Hemming et al, 1993) has benefitted from behavioral modeling; it is prototyping innovative generative design methods for recurrent building design and it integrates two streams of multi-generational research and software systems - ABLOOS, a hierarchical framework for layout design (Coyne, 1991) and GENESIS, a solids grammar development system (Heisserman, 1991).

In conclusion, research results embodied in software may transfer best when the adaptable and flexible capabilities of people for transitioning knowledge and experience across contexts is combined with computational support for capturing and preserving more software design information and for evoking a shared understanding among all participants in system development. A continuing consideration of these issues will be important to all of the following goals articulated at the the workshop:

- better supporting multi-generational research projects which build-on or extend existing research software systems*
- using research software systems within courses which can serve as testbeds for the methods embodied in the software, can drive the usability of such systems and provide tight feedback loops with a target community of users. The courses also benefit by enabling students to experience and to help shape the "cutting edge" of the methods within their field.
- the more effective transition of new methods, via experienced developers, users and comprehensive software systems (in the sense described above) embodying those methods.

We believe that the workshop reported on here is only the first step needed toward the development of a shared understanding of research experiences, the dissemination of case studies and tutorials on successful projects, and the continuous evolution and evaluation of research software development as an important accessory to people in the transition of research results and methods.

References

Brooks, F. P. *The Mythical Man-Month.* Reading, MA: Addison -Wesley, 1975.

Coyne, R.F.,"ABLOOS: An Evolving Hierarchical Design Framework,[11] Ph.D. dissertation. Department of Architecture, Carnegie Mellon University, Pittsburgh, PA, 1991; also available as technical report EDRC-02-15-9L

Coyne, R. F., Hemming, U., Piela, P. and Woodbury, R., "Behavioral Modeling in Design System Development," Proceedings of CAAD Futures '93, Carnegie Mellon University, Pittsburgh, PA, USA, July, 1993.

Curran, L. J. "'A[1] for Effort," *IEEE Spectrum,* February 1993, p. 50-52.

Hemming U., Coyne R. and Woodbury, R., "SEED: A Software Environment to Support the Early Phases in building Design," Proceedings of ARECDAO (93, Barcelona, Spain, March-April, 1993.

Hoyd, C, Mehl, W.-M., Reisin, F.-M., Schmidt, G., and Wolf, G., "Out of Scandinavia: Alternative approaches to software design and system development," ***Human Computer Interaction* 4, 4 (1989), 253-350.**

Garg, P. and Scacchi, W, "A Hypertext System to Manage Software Life Cycle Documents", Proceedings of 21st Hawaii International Conference in System Sciences, Vol 2, pp. 337-346.

Heisserman, J., "Generative Geometric Design and Boundary Solid Grammars," Ph.D. dissertation, Department of Architecture, Carnegie Mellon University, Pittsburgh, PA, 1992.

Jacobson, I., Christerson, M., Jonsson, P. and Overgaard, G., *Object-Oriented Software* ***Engineering, A Use Case Driven Approach,* Addison-Wesley, NY, 1992.**

Kübler-Ross, E., *On Death and Dying,* New York: Macmillan, 1970.

Levy, S., Subrahmanian, E., Konda, S., Coyne, R., Westerberg, A. and Reich, Y., "An Overview of the n-dim Environment", Technical Report, EDRC-05-65-93, Carnegie Mellon University, January, 1993.

Minneman, S., "The Social Construction of a Technical Reality," Proceedings of NSF Workshop on Information Capture and Access in Engineering Design Environments, Cornell University, Ithaca, NY, 1991.

Myers, B.A. and Rosson, M. B., "Survey on User Interface Programming," in *Human Factors in Computing Systems,* Proceedings of SIGCHI '92, Monterey, CA, May 1992.

Naur, P., "Programming As Theory Building", 1985, in *Computing: A Human Activitty,* Naur, P., ACM Press, Addison-Wesley, NY, 1992.

Rogers, E. C, *Diffusion Of Innovations,* New York: Free Press, 1983.

Rubin, K.S. and Goldberg, A., "Object Behavior Analysis," Communications of the ACM, September, 1992, Volume 35, Number 9, pages 48-62.

Subrahmanian, E., Coyne, R., Konda, S., Levy, S., Martin , R., Monarch , L, Reich, Y. and Westerberg, A., "Support System for Different-Time Different Place Collaboration for Concurrent Engineering", To appear in: Proceedings of WET-ICE (Workshop on

Enabling Technologies In Concurrent Engineering), CERC, West Virginia, USA, 1993.

Toye, G., Cutkosky, L., Leifer, L., Tenenbaum J., and Glicksman J., "SHARE: A Methodology and Environment for Collaborative Product Development", To appear in: Proceedings of WET-ICE (Workshop on Enabling Technologies In Concurrent Engineering), CERC, West Virginia, USA, 1993.